Logical Effort: Designing for Speed on the Back of an Envelope

Ivan E. Sutherland
Robert F. Sproull
Sun Microsystems, Inc.
Mountain View, CA 94043

Designers know to use strings of inverters with geometrically increasing sizes to drive large capacitive loads. But they are unsure how to optimize arbitrary logic networks so as to achieve least delay without resorting to trial-and-error circuit simulations. The method of logical effort, introduced in this paper, is a simple method that optimizes networks for speed.

The method of logical effort shows how many stages of logic are required for the fastest implementation of any given logic function. The effort of computing a logic function requires amplification stages just the same as the effort of driving large capacitive loads. The method reveals the proper transistor sizes in each stage to realize the fastest overall operation. It also provides a guide that can be applied in the early "back of the envelope" stages of design to choose among major alternative structures without extensive simulation work.

The method assigns a *logical effort* to each logic function. The logical effort of an inverter is taken to be one. The logical effort for any other logic function describes how much worse it is than an inverter at producing output current, given an equivalent amount of input capacitance. The logical effort of a logic function depends mainly on its circuit topology and slightly on the electrical properties of the fabrication process used to build it. In CMOS the logical effort of each input of common two-input logic functions ranges from about 4/3 for NAND to 4 for XOR. The logical effort of functions with more than two inputs is generally higher.

Logical efforts for individual stages of logic can be combined to find the logical effort of networks. Where several stages of logic drive each other in a string, the overall effort involves the product of their individual efforts. Where several logic devices are driven from a common source, the overall effort involves the sum of the efforts of the driven devices. Compound circuits with smaller overall logical effort can be made to run faster than logically equivalent circuits with larger logical effort.

1 Delay in a single logic gate

The method of logical effort reformulates a simple conventional *RC* model of delay in a CMOS logic gate and introduces some new terminology. The delay in a logic gate can be expressed as the sum of two components, a fixed part called the *parasitic delay*, p, and a part proportional to the load on the gate's output, called the *effort delay*, f:

$$d = f + p \tag{1}$$

Delay expressions such as this one express delay in units of τ, a time unit that characterizes the actual semiconductor process being used.

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

$$f = gh \tag{2}$$

The logical effort 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 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:

$$h = C_{out}/C_{in} \tag{3}$$

where C_{out} is the capacitance that loads the logic gate and C_{in} is the capacitance presented by the logic gate at one of its input terminals.

Combining Equations 1 and 2, we obtain the basic equation that models the delay through a single logic gate, in units of τ:

$$d = gh + p \tag{4}$$

This equation shows clearly that logical effort and electrical effort both contribute to delay in the same way. This formulation separates τ, g, h, and p, the four contributions to delay. Note that p and g are independent of the size of the transistors in the logic gate, while h relates directly to transistor sizes.

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. Logical effort is defined so that an inverter has a logical effort of one. Thus τ is the delay of an ideal inverter with no parasitic delay that drives another identical inverter.

It is interesting but not surprising to note from Table 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. 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. The method of logical effort allows you to choose among such alternatives.

Gate type	Number of inputs					
	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
Muller C		2	3	4	5	n
XOR (parity)		4	6–12	16–32		
majority			4			

Table 1: Logical effort for inputs of static CMOS gates. These figures assume that p-type transistors have half the conductance of n-type transistors of identical geometry.

Gate type	Parasitic delay
inverter	p_{inv}
n-input NAND	np_{inv}
n-input NOR	np_{inv}
n-way multiplexer	$2np_{inv}$
n-input Muller C	$2np_{inv}$
XOR, XNOR	$4p_{inv}$
3-input majority	$6p_{inv}$

Table 2: Estimates of parasitic delay of various logic gate types. A typical value of p_{inv}, the parasitic delay of an inverter, is 0.6.

Electrical effort is usually expressed as a ratio of transistor widths rather than actual capacitances. 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, both C_{in} and C_{out} can be expressed 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.

The parasitic delay of a logic gate is fixed, independent of the size of the logic gate and of the load capacitance it drives. This delay is a form of overhead that accompanies any gate. The principal contribution to parasitic delay is the capacitance of the source/drain regions of the transistors that drive the gate's output. Table 2 presents 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 0.6 delay units, which is used in the examples in this paper.

1.1 Deriving the delay model

The delay model in Equation 4 is derived from a simple RC model of a logic gate, shown schematically in Figure 1. The delay d is proportional to the

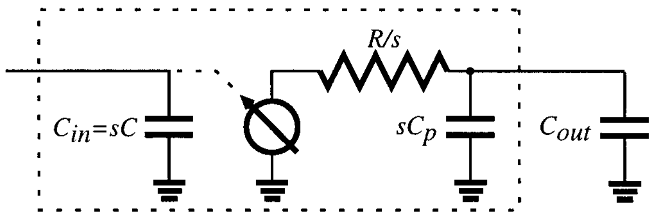

Figure 1: Equivalent circuit of a logic gate with scale s.

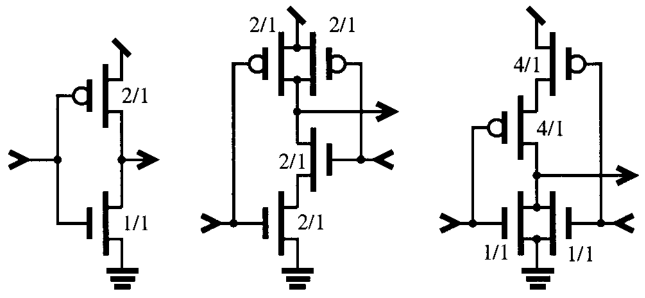

Figure 2: Designs for an inverter, a NAND gate, and a NOR gate that have approximately the same drive characteristics. Transistor size ratios, given as W/L, can be scaled uniformly to obtain larger or smaller gates.

RC delay of the pullup or pulldown resistance charging the load capacitance C_{out} and stray capacitance C_p:

$$d \propto \frac{R}{s}(sC_p + C_{out}) \tag{5}$$

Transistors in the circuit are assumed to have minimum length and a width proportional to the overall scale, s, of the logic gate. Input and stray capacitances thus scale up with s, while resistances scale down. By normalizing Equation 5 by R_{inv} and C_{inv}, the parameters of a reference inverter, Equation 4 emerges:

$$d \propto \left(\frac{RC}{R_{inv}C_{inv}}\right)\left(\frac{C_{out}}{C_{in}}\right) + \frac{RC_p}{R_{inv}C_{inv}} = gh + p \tag{6}$$

1.2 Determining the logical effort of a logic gate

Equation 6 offers a definition of logical effort: *The logical effort of an input of a logic gate is the ratio of the gate's input capacitance to that of an inverter that delivers equal output current.* To compute the logical effort of an arbitrary logic gate, design a circuit that performs the intended logic function with the same drive characteristics as an inverter. Then the logical effort is the ratio of the input capacitance of one of the logic gate's inputs to the input capacitance of the inverter.

Figure 2 shows designs for an inverter, a NAND gate, and a NOR gate. In order to achieve equal rising and falling delays in the CMOS process used to fabricate these circuits, the effective width of pullup transistors is twice that of pulldowns. Using the sum of transistor gate areas as a measure of capacitance, we find that the input capacitance is 3 units for the inverter, 4 units for the NAND gate, and 5 units for the NOR gate. Thus the logical effort of an input of the NAND gate is 4/3 and that of the NOR gate is 5/3; these are the numbers recorded in Table 1. In these circuits, it is the series-connected transistors, which must be wider than those of an inverter to achieve comparable drive, that increase the logical effort.

2 Multi-stage logic networks

The method of logical effort is applied in two ways to design fast multi-stage logic networks. It reveals the best number of stages to use in the network and it shows how to get least overall delay by equalizing the effort delay in each stage of the path.

The notions of logical and electrical effort generalize easily to multi-stage networks. The logical effort along a path compounds by multiplying the logical efforts of all the logic gates along the path. We use the upper-case symbol G to denote the *path logical effort.* 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 upper-case symbol, H, to indicate the *path electrical effort.*

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 to obtain an electrical effort. 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 the path. We define the branching effort b at the output of a logic gate to be:

$$b = \frac{C_{onpath} + C_{offpath}}{C_{onpath}} \tag{7}$$

where C_{onpath} is the load capacitance of the next logic gate along the path we are analyzing and $C_{offpath}$ is the capacitance of fanout connections that lead off the path. Note that if the fanout is one, 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.

Armed with definitions of logical, electrical, and branching effort along a path, we can define the *path effort*, F. The equation that defines path effort is reminiscent of Equation 2, which defines the effort for a single logic gate:

$$F = GBH \tag{8}$$

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 delay, D, is the sum of the delays of each of the N stages of logic in the path. As in the expression for delay in a single stage (Equation 4), we shall distinguish the *path effort delay,* D_F, and the *path parasitic delay,* P:

$$D = \sum d_i = D_F + P \tag{9}$$

where the subscripts index the logic stages along the path. The path effort delay is simply $D_F = \sum g_i h_i$ and the path parasitic delay is $P = \sum p_i$.

Optimizing the design of a logic network proceeds from a very simple result: *The path delay is minimized when each stage in the path bears the same stage effort, f.* This result is obtained by minimizing D in Equation 9 by varying the h_i subject to the constraint that the path electrical effort, H, is fixed. This minimum delay is achieved when the stage effort is

$$\hat{f} = g_i h_i = F^{1/N} \tag{10}$$

A hat over a symbol indicates 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:

$$\hat{D} = NF^{1/N} + P = N(GBH)^{1/N} + P \tag{11}$$

From a simple computation of logical, branching, and electrical efforts we obtain an estimate of the minimum delay achievable for a logic network.

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. Equations 10 and 2 combine to require that each logic stage be designed so that

$$\hat{h}_i = F^{1/N} / g_i \tag{12}$$

This relationship is used to compute transistor sizes, starting at the beginning of the path and applying this equation at each logic gate along the path.

p_{inv}	ρ	$\ln \rho$	$d = \rho + p_{inv}$
0	$2.718 = e$	1.000	2.718
0.2	2.91	1.069	3.11
0.4	3.09	1.129	3.49
0.6	3.27	1.184	3.87
0.8	3.43	1.233	4.23
1.0	3.59	1.278	4.59
2.0	4.32	1.463	6.32
3.0	4.97	1.604	7.97
4.0	5.57	1.718	9.57

Table 3: Optimum effort per stage, ρ, as a function of p_{inv}. The last column gives the stage delay obtained when the number of stages in a path is optimum. Calculated from Equation 15.

3 Choosing the length of a path

Although equalizing the effort borne by each stage in a path minimizes delay for a given path, the delay can sometimes be reduced further by adjusting the number of stages in the path. This optimization is also a straightforward result of our delay model.

Consider a path of logic gates containing n_1 stages, to which we append n_2 additional inverters to obtain a path with a total of $N = n_1 + n_2$ stages. We will assume that the original n_1 stages cannot be altered except by scaling because they perform necessary logic functions, while the number of inverters can be altered if necessary to reduce delay. Although preserving the correct logic function requires that an even number of inverters be used, we will assume that an odd number of inverters can be accommodated by changing the logic function as necessary. We will assume that the path effort $F = GBH$ is known: the logical and branching efforts are properties of the n_1 logic stages that will not be altered by adding inverters, and the electrical effort is determined by the input and load capacitances required.

The minimum delay of the N stages is the sum of the delay in the logic stages and in the inverter stages:

$$\hat{D} = NF^{1/N} + \left(\sum_{i=1}^{n_1} p_i\right) + (N - n_1)p_{inv} \tag{13}$$

The first term is the delay obtained by distributing effort equally among all N stages, as shown in the preceding section. The second term is the parasitic delay of the logic stages, and the third term is the parasitic delay of the inverters. Differentiating this expression with respect to N and setting the result equal to zero, we obtain:

$$\frac{\partial \hat{D}}{\partial N} = \frac{-F^{1/N-1}}{N^2} + F^{1/N} + p_{inv} = 0 \tag{14}$$

Now define the solution to this equation to be $\hat{N}$, the number of stages to use to obtain least delay. If we define $\rho = F^{1/\hat{N}}$ to be the effort borne by each stage when the number of stages is chosen to minimize delay, the solution of the equation can be expressed as:

$$p_{inv} + \rho(1 - \ln \rho) = 0 \tag{15}$$

In other words, the fastest design is one in which each stage along a path bears an effort equal to ρ, where ρ is a solution of Equation 15. Thus we call ρ the *optimum stage effort.*

It is important to understand the relationship between ρ and $\hat{f}$, both of which appear to specify the stage effort required to achieve least delay. The expressions for $\hat{f}$, such as Equation 10, determine the best stage effort when the number of stages, N, is known. By contrast, the value ρ, which is a constant independent of the properties of a path, represents an ideal stage delay, which may not be achievable in an actual path.

Equation 15 shows that the optimum effort, ρ, is a function of the parasitic delay of an inverter. This result has an intuitive explanation. The stray capacitance of the logic gates in the network is fixed—you can't do much about it, and it simply adds a fixed delay to the path. Adjusting the sizes of the logic gates will change their effort delay, but not the delay contribution due to their parasitic delay. But when you add an inverter as a "gain" element in the hope of speeding up the circuit, you need to compare the improvement offered by its gain to the delay added by its parasitic capacitance. As p_{inv} grows, it becomes less advantageous to add inverters because the stray load is excessive, and the optimum number of stages diminishes.

It is not hard to solve Equation 15 for values of ρ given values of p_{inv}. Table 3 shows the solution for several values of the inverter's parasitic delay. Note that if we assume that the parasitic delay of an inverter is zero, then $\rho = e = 2.718$; this is the familiar result when parasitic delay is ignored [5]. In our examples, we shall assume that $p_{inv} = 0.6$ and thus that $\rho = 3.27$. The quantity ρ is sometimes called the *optimum step-up ratio,* because it is the ratio of the sizes of successive inverters in a string of inverters designed to drive a large capacitive load.

Actual designs will require us to choose a step-up ratio that differs somewhat from ρ because the design must use an integral number of stages. Given the path effort F, we must find the number of stages $\hat{N}$ that gives the least delay; this result will have a stage effort delay close to ρ. Table 4 shows how to select $\hat{N}$, given the effort F and the value of the parasitic delay of an inverter. Values in the table are calculated by finding those values of F for which $\hat{N}(F^{1/\hat{N}} + p_{inv}) = (\hat{N}+1)(F^{1/(\hat{N}+1)} + p_{inv})$. These are the values of path effort for which the best $\hat{N}$-stage design provides just as much delay as the best $(\hat{N}+1)$-stage design. As F gets large, $\hat{N} \approx \ln F / \ln \rho$, so the stage delay approaches $\rho + p$.

It is interesting to ask how much the delay for a properly optimized circuit is changed by using the wrong number of stages. The answer, as

$\hat{N}$	$p_{inv} = 0.0$	$p_{inv} = 0.6$	$p_{inv} = 0.8$	$p_{inv} = 1.0$
	0	0	0	0
1				
	4.0	5.13	5.48	5.83
2				
	11.4	17.7	20.0	22.3
3				
	31.6	59.4	70.4	82.2
4				
	86.7	196	245	300
5				
	237	647	848	1090
6				
	648	2130	2930	3920
7				
	1770	6980	10100	14200
8				
	4820	22900	34700	51000
9				
	13100	74900	120000	184000
10				
	35700	245000	411000	661000
11				
	97300	802000	1410000	2380000
12				
	265000	2620000	4860000	8560000
13				
	720000	8580000	16700000	30800000
14				
	1960000	28000000	57400000	111000000
15				
	5330000	91700000	197000000	398000000

Table 4: Table of ranges of path effort, F, and the optimum number of stages, $\hat{N}$. For example, when $p_{inv} = 0.6$ and $F = 205$, $\hat{N} = 5$ because 205 lies between 196 and 647.

$N/\hat{N}$	$D/\hat{D}$
0.25	7.42
0.5	1.46
0.7	1.09
1.0	1.00

$N/\hat{N}$	$D/\hat{D}$
1.4	1.06
2.0	1.24
3.0	1.62
4.0	2.01

Table 5: The relative delay of a network, $D/\hat{D}$, as a function of the relative error in the number of stages used, $N/\hat{N}$. Assumes $p_{inv} = 0.6$.

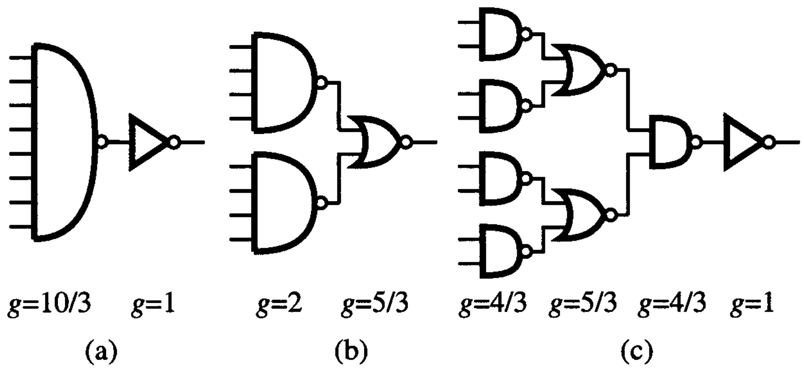

Figure 3: Three circuits that compute the AND function of eight inputs.

shown in Table 5, is that delay is quite insensitive to the number of stages, provided the deviation from optimum is not too large. As the table shows, doubling the number of stages from optimum increases the delay only 24%. Using half as many stages as the optimum increases the delay 46%. Thus one need not slavishly stick to exactly the correct number of stages. It is slightly better to err in the direction of using too many stages than too few. A stage or two more or less in a design with many stages will make little difference, provided proper transistor sizes are used. Only when very few stages are required does a change of one or two stages make a large difference.

4 Examples

4.1 An 8-input AND network

When a large number of inputs must be combined, there are several options for the structure of the circuit. Figure 3 shows three possibilities for computing the AND function of eight inputs. Which one is best?

Recalling that the path logical effort, G, is the product of the logical efforts of the logic gates along the path, we find that $G = 10/3 \times 1 = 3.33$ for case a, $6/3 \times 5/3 = 3.33$ for case b, and $4/3 \times 5/3 \times 4/3 \times 1 = 2.96$ for case c. These figures can be used in the delay equation, Equation 11, to find the minimum delay that can be obtained from each circuit. These equations also include an estimate of parasitic delays, obtained by summing the parasitic delays of each of the logic gates along the path:

$$\text{Case } a \qquad \hat{D} = 2(3.33H)^{1/2} + 5.4 \qquad (16)$$

$$\text{Case } b \qquad \hat{D} = 2(3.33H)^{1/2} + 3.6 \qquad (17)$$

$$\text{Case } c \qquad \hat{D} = 4(2.96H)^{1/4} + 4.2 \qquad (18)$$

It is clear from these equations that case b will always be better than a.

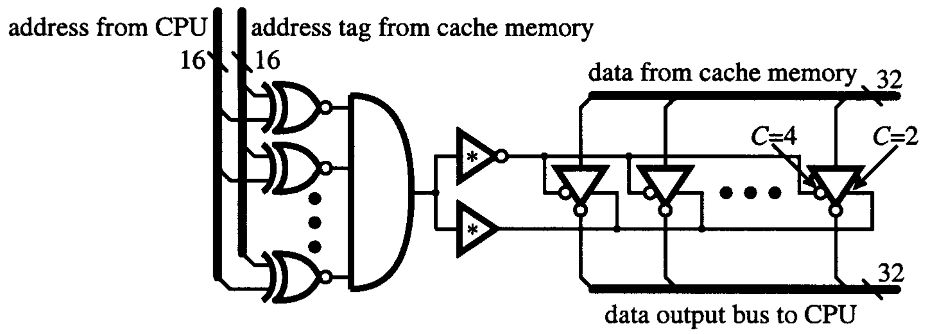

Figure 4: Block diagram of a cache comparator.

Choosing between cases *b* and *c* depends on the electrical effort, H, that must be borne by the network. When $H = 1$, case *b* will be best, but for $H = 12$, case *c* will be best. The equations show that for high electrical effort, case *c* yields least delay because the $H^{1/4}$ factor dominates.

To illustrate the computation of transistor sizes to achieve least delay, consider a case where $C_{in} = 4$ units and $C_{out} = 48$. The preceding equations show that case *c* should be selected, and Equation 10 gives the stage effort $\hat{f} = F^{1/N} = (2.96 \times (48/4))^{1/4} = 2.44$. Let us work forward along the path, starting with the 2-input NAND gate at the left. We know that $C_{in} = 4$, $\hat{f} = g_0 h_0 = 2.44$, and that the logical effort of the NAND is $g_0 = 4/3$. We solve for $h_0 = 1.83$ and then because $h = C_{out}/C_{in}$, we solve to obtain $C_{out} = 7.33$. This, then, is the input capacitance of the NOR gate in the second stage. Proceeding analogously for the other stages, we find that the input capacitance of the third stage is 10.73 and of the fourth stage is 19.66.

4.2 A cache comparator

As a second example let us consider the key circuit of a cache memory controller, namely its address comparator. This circuit compares two 16 bit addresses, and if they match delivers 32 bits of data to an output bus. Such a circuit is difficult to design because it presents not only the large logical efforts of the bit-by-bit XOR function and the 16-input AND that together detect the match, but also the large electrical effort imposed by the control lines for the 32 bus switches.

An outline of the circuit is shown in Figure 4. This figure is somewhat schematic; it shows only a few of the 16 XNOR gates that do the bit-by-bit comparison; it uses a single 16 input AND gate symbol to represent what will ultimately be a tree of simpler NAND and NOR gates; and it uses inverting and non-inverting amplifier symbols as drivers for the 32 bus selection gates at the output. The star notation in the amplifier symbols indicates that they may contain more than one inverter stage; of course the non-inverting amplifier will contain an even number of stages and the inverting amplifier an odd number. We call such a structure of inverting and non-inverting amplifiers a *fork.*

The main task in designing this circuit is to choose a topology for the AND gate. Each candidate topology has an easily calculated logical effort. For example if we use a tree of two-input NAND and NOR gates four stages deep to implement the 16 input AND, its logical effort will be $4/3 \times 5/3 \times 4/3 \times 5/3 = 4.94$; note that this is lower than $g = 6$ for a single 16-input NAND gate. The logical effort of an XNOR gate implemented by three NAND gates ($\overline{a \oplus b} = nand(nand(a,b), nand(\overline{a}, \overline{b}))$) is $4/3 \times 4/3 = 16/9$. Thus the path logical effort of the entire comparator is $16/9 \times 4.94 = 8.78$. If we assume that the true and complement inputs to the XNOR gates may be loaded with 8 units of capacitance, and the total output load as shown in the figure is $2 \times 32 + 4 \times 32 = 192$ units, then $H = 24$. Thus the path effort is $F = GBH = 210.7$. Table 4 shows that for $p_{inv} = 0.6$, we should use a five-stage design. Unfortunately, the design we have chosen uses a minimum of six stages on one leg of the fork and seven on the other. A bushier and less deep tree will be better, e.g., a two-level tree with a 4-input NAND followed by a 4-input NOR. Because the AND gate is followed by a fork, the tree may implement either an AND or NAND function with possible interchange of the fork outputs. Because both branches of the fork have logical effort of one, such an interchange has no effect on the path logical effort.

Because both true and complement outputs are required to control the bus switches, there must somewhere be a branch point in the path between the circuit inputs and the outputs. The method of logical effort shows that this branch point may be placed anywhere without hurting circuit performance. If we put the branch point near the input, we will make two duplicate AND trees, one to drive the true signal and one to drive its complement. These two trees will each, of course, use smaller transistors than the single tree illustrated in the figure, since each drives a lesser load. Together they will impose as much load on the inputs as the single tree. If we choose to use a single tree, it is because we believe that it will reduce the circuit area, not because it is faster.

4.3 An array multiplier

A full adder has three inputs of equal arithmetic value and two outputs called *sum* and *carry.* One way to implement such a circuit uses a three-input parity gate to generate the sum and a three-input majority gate to generate the carry. Such a circuit is suggested in Figure 5.

One implementation of such an adder circuit uses single-stage parity and majority circuits, each consisting of transistors in series-parallel connections. The single-stage implementations of these circuits require both true and complement inputs. Because the logical effort of the fork of amplifiers that will precede the parity or majority gates is one, we can consider the true

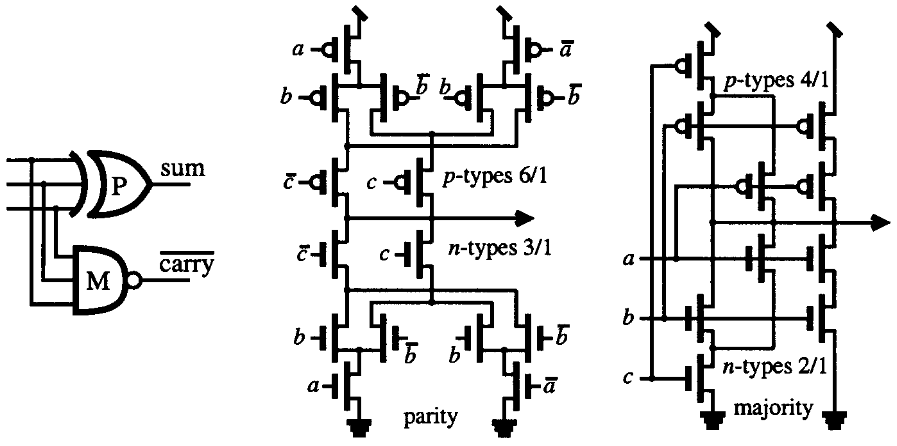

Figure 5: Parity and majority circuits for building an array multiplier.

and complement inputs as a *bundle* and assign a logical effort to the entire bundle. The logical effort of parity and majority circuits is not the same on each input bundle. Some inputs are easier to drive than others because they are connected to fewer transistors. The logical efforts of the inputs of the majority gate are 2, 4, and 4, and the logical efforts of the bundled inputs of the parity gate are 6, 12, and 6.

A group of such full adders can be combined into an array multiplier. One input of each adder comes from an AND gate that combines multiplier and multiplicand bits. The other two inputs of each adder come from the sum and carry outputs of two previous adders. The design of such an array adder poses two topological questions: First, how should the three inputs of the two gates be connected? Should the easiest-to-drive input of the majority gate ($g = 2$) be connected to one of the easy-to-drive inputs of the parity gate ($g = 6$), producing inputs with relative drive requirements of (8, 10, 16), or should the easiest-to-drive input of the majority gate be connected to the hard-to-drive input of the parity gate, producing inputs with relative drive requirements of (10, 10, 14)? Second, which of the resulting three inputs should be driven by a sum output, which by a carry output, and which by the AND gate that combines multiplier and multiplicand bits?

There are six possible topologies. The best one makes one of the inputs as easy to drive as possible; it is the 8, 10, 16 connection. The sum output of a previous stage should drive this easy-to-drive input. The carry output of a previous stage should drive the intermediate input, and the AND gate combining multiplier and multiplicand values should drive the hardest-to-drive input. The sum should be assigned the easiest drive task because the logic gate that produces the sum has the highest logical effort.

Within the optimum topology the best design balances the relative sizes of the transistors in the parity and the majority gates. The delay in the two gates should match, because if either output is produced prematurely because its gate has transistors that are too big, it will take more current from its inputs than necessary. Balancing the speed of the two gates produces the least delay in the slowest of them. The method of logical effort reduces this design problem to a few expressions that can be minimized by taking partial derivatives.

5 Refinements

The method of logical effort leads to a number of related results that are covered in a more complete treatment [9], including:

- Aggregating logical effort into bundles provides an easy way to reason about complex circuits with wide or double-rail datapaths.
- Logical effort shows that forks always have designs in which the two paths differ in length by one inverter. Moreover, the number of inverters to use in a fork can be obtained from a table similar to Table 4.
- Logic gates can be deliberately designed to reduce the logical effort of certain inputs at the expense of others, e.g., to favor a carry path in a sum circuit. Any attempt to favor an input results in raising the combined logical effort of all inputs, however.
- The method of logical effort can also be used to determine path length and transistor sizes that minimize area subject to a delay constraint.
- Test chips can be designed to measure directly values for τ and the logical effort and parasitic delay of different logic gates. Alternatively, a circuit simulator can be used to find these values.
- While the logical effort of an n-way multiplexer is independent of n (see Table 1), it is not wise to make multiplexers arbitrarily big. The parasitic delay of common designs turns out to predict that 4-way multiplexers are the best size to use. This result applies to buses as well.
- Although the examples in this paper show gates designed to produce roughly the same delays on rising and falling transitions, the method of logical effort can be applied when these delays differ.
- The method of logical effort works for logic families other than fully static gates as in Figure 2, including precharged and domino logic.
- Although the method of logical effort does not handle pass gates directly, pass gates can often be melded with a surrounding logic gate for analysis.

6 Related work and challenges

The method of logical effort arises not from a new delay model, but rather from factoring and normalizing a simple model that is used frequently [2]. Our model is less accurate than those that include terms to account for differences in rise times of input signals [1]. This omission is perhaps not serious in applications of logical effort, however, because equalizing effort delay usually results in roughly equalizing rise times.

Nemes modeled a stray capacitance that scales with the size of a logic gate and showed the implications for optimum step-up ratio [6].

Many works treat transistor sizing as a numerical optimization problem [3, 4, 7], but these techniques are hard to apply by hand. Also, most fail to determine whether the right number of stages is used in a network. The method of logical effort can be used to obtain good starting designs for these optimization programs.

Logical effort holds promise as a measure of computational complexity that accounts for the switching required to implement a required logic function. Can we find bounds on the logical effort of certain logic functions or subsystems such as adders and multipliers? Can we use logical effort to evaluate different design features such as asynchronous handshakes or testability? Logical effort may be able to express the "cost" of testability or of adding a new instruction to a decoder in a RISC machine.

7 Conclusions

The method of logical effort developed as we attempted to design asynchronous circuits for maximum speed. The circuits were sufficiently large and complex that the conventional RC model and our intuition did not lead to the best designs. However, the symmetry of CMOS circuits, and especially the forms that occur in asynchronous designs, led us to compare them to simple inverters, which are also symmetric: the equations of logical effort followed naturally.

The method of logical effort finds the circuit with the least delay, 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 and increase 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 method can sometimes be improved by careful simulation with a circuit simulator and subsequent adjustment of transistor sizes. However, we have evidence that the method of logical effort alone obtains designs that are within 10% of the optimum.

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 the 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 fanin 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 tri-state drivers. Logical effort unifies all of these design problems.

8 Acknowledgements

We wish to thank six companies that sponsored the Asynchronous Systems project at Sutherland, Sproull, and Associates, which led to the method of logical effort: Apple Computer, Austek Microsystems, Digital Equipment Corporation, Evans and Sutherland Computer Corporation, Floating Point Systems, and Schlumberger. Our colleagues Charles E. Molnar, Ian W. Jones, Erik L. Brunvand, and Peter Single [8] contributed in several ways to the work.

References

[1] M. Horowitz. Timing Models for MOS Circuits. PhD thesis, Stanford University, December 1983. TR SEL83-003.

[2] A. Kanuma. CMOS Circuit Optimization. *Solid-State Electronics*, 26(1):47–58, 1983.

[3] C. M. Lee and H. Soukup. An algorithm for CMOS timing and area optimization. *IEEE J. Solid-State Circuits*, 19(5):781–787, 1984.

[4] D. P. Marple and A. El Gamal. Optimal Selection of Transistor Sizes in Digital VLSI Circuits. *Proc. Stanford Conf. on Advanced Research in VLSI*, March 1987.

[5] C. A. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980, p. 12.

[6] M. Nemes. Driving Large Capacitances in MOS LSI Systems. *IEEE J. Solid-State Circuits*, SC-19:159–161, 1984.

[7] J. Shyu, J. P. Fishburn, A. E. Dunlop, and A. L. Sangiovanni-Vincentelli. Optimization-Based Transistor Sizing. *IEEE 1987 Custom Integrated Circuits Conf.*, 417–420, 1987.

[8] P. Single. The Theory of Logical Effort and Overhead. *Proc. 7th Australian Microelectronics Conf.*, May 1988.

[9] I. E. Sutherland and R. F. Sproull. *Logical Effort: Designing Fast* MOS *Circuits.* Monograph in preparation.