Logic Design

Dinesh Sharma
EE Department, IIT Bombay

(Design Examples for Logical Effort have been taken
from the book by Sutherland, Sproull and Harris)

September 9, 2019

Contents

List of Figures

Chapter 1

Transistor Models

We shall use simple analytical models for MOS transistors. We use a sign convention according to which, voltage and current symbols associated with the pMOS transistor (such as V_{Tp}) have positive values. Then, the n channel formulae can be used for both transistors and we shall assign signs to quantities explicitly.

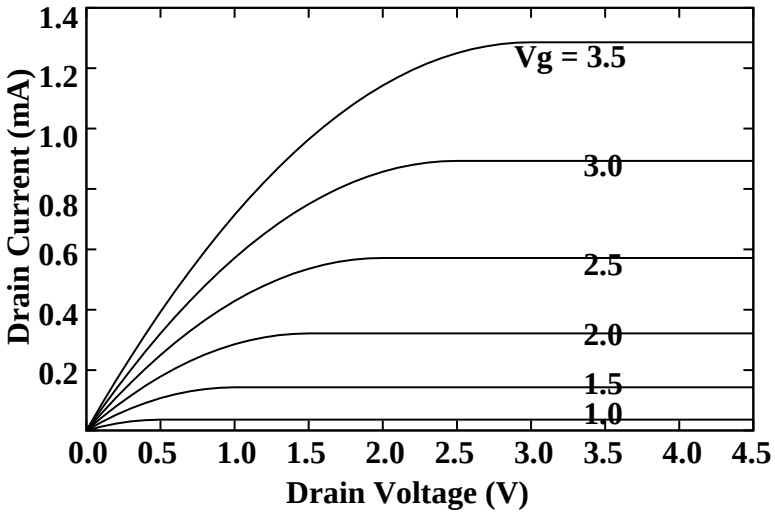


Figure 1.1: MOS characteristics according to the simple analytic model

The model we use is described by the following equations:
for $V_{gs} \leq V_T$,

$$I_{ds} = 0 \tag{1.1}$$

for $V_{gs} > V_T$ and $V_{ds} \leq V_{gs} - V_T$,

$$I_{ds} = K\left[(V_{gs} - V_T)V_{ds} - \frac{1}{2}V_{ds}^2\right] \tag{1.2}$$

and for $V_{gs} > V_T$ and $V_{ds} > V_{gs} - V_T$,

$$I_{ds} = K\frac{(V_{gs} - V_T)^2}{2} \tag{1.3}$$

The saturation region equation is somewhat oversimplified because it assumes that the current is independent of V_{ds}. In reality, the current has a weak dependence on V_{ds} in this region.

In order to model the saturation region more accurately, we adopt an "Early Voltage" like formalism.

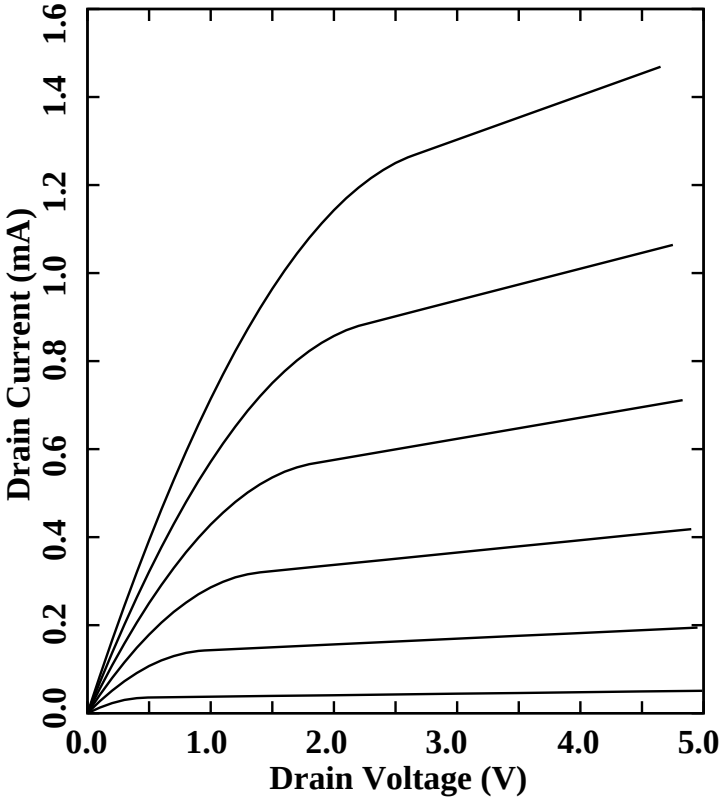


Figure 1.2: MOS characteristics with non zero conductance in saturation

It is assumed that the current increases linearly in the saturation region. All linear characteristics in saturation can be produced backwards towards negative drain voltages and will intersect the drain voltage axis at a single point at $-V_E$. (This is, at best, an approximation). Because the conductance in saturation is now non zero, the onset of saturation has to be redefined, so that the current and its derivative are continuous at the boundary of linear and

saturation regimes. The current equations are given by:
For $V_{gs} > V_T$ and $V_{ds} \leq V_{dss}$,

$$I_{ds} = K\left[(V_{gs} - V_T)V_{ds} - \frac{1}{2}V_{ds}^2\right] \tag{1.4}$$

and for $V_{gs} > V_T$ and $V_{ds} > V_{dss}$,

$$I_{ds} = I_{dss}\frac{V_d + V_E}{V_{dss} + V_E} \tag{1.5}$$

Where V_E is the 'Early Voltage'. Here V_{dss} and I_{dss} are saturation drain voltage and drain current respectively. Since the current values must match at either side of $V_{ds} = V_{dss}$, we must have:

$$I_{dss} \equiv K\left[(V_{gs} - V_T)V_{dss} - \frac{1}{2}V_{dss}^2\right]. \tag{1.6}$$

For the curve to be smooth and continuous at $V_d = V_{dss}$, the value of the first derivative should match on either side of V_{dss}. Therefore,

$$K(V_{gs} - V_T - V_{dss}) = \frac{I_{dss}}{V_{dss} + V_E}$$

So,

$$K(V_{gs} - V_T - V_{dss})(V_{dss} + V_E) = K\left[(V_{gs} - V_T)V_{dss} - \frac{1}{2}V_{dss}^2\right] \tag{1.7}$$

This leads to a quadratic equation in V_{dss}

$$\frac{1}{2}V_{dss}^2 + V_E V_{dss} - (V_{gs} - V_T)V_E = 0 \tag{1.8}$$

Solving this quadratic, we get

$$V_{dss} = V_E\left(\sqrt{1 + \frac{2(V_{gs} - V_T)}{V_E}} - 1\right) \tag{1.9}$$

For $V_E >> V_{gs} - V_T$ this reduces to

$$V_{dss} \simeq (V_{gs} - V_T)\left(1 - \frac{V_{gs} - V_T}{2V_E}\right) \tag{1.10}$$

Characteristics of a MOS transistor using this model are shown in fig.1.2. While accurate modeling of the output conductance is essential for linear design, the simpler model assuming constant I_d in saturation is often adequate for preliminary digital design. In any case, final designs will have to be validated with detailed simulations. In this booklet, we shall use the simple model for MOS devices to keep the algebra simple.

Chapter 2

Static CMOS Logic Design

Static logic circuits are those which can hold their output logic levels for indefinite periods as long as the inputs are unchanged. Circuits which depend on charge storage on capacitors are called dynamic circuits and will be discussed in a later chapter.

2.1 Static CMOS Design style

The most common design style in modern VLSI design is the Static CMOS logic style. In this, each logic stage contains pull up and pull down networks which are controlled by input signals. The pull up network contains p channel transistors, whereas the pull down network is made of n channel transistors. The networks are so designed that the pull up and pull down networks are never 'on' simultaneously. This ensures that there is no static power consumption.

2.2 CMOS Inverter

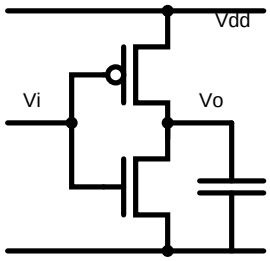


Figure 2.1: The basic CMOS inverter

The simplest of such logic structures is the CMOS inverter. In fact, for any CMOS logic design, the CMOS inverter is the basic gate which is first analyzed and designed in detail. Thumb rules are then used to convert this design to other more complex logic. The basic CMOS inverter is shown in fig. 2.1. We shall develop the characteristics of CMOS logic through the inverter structure, and later discuss ways of converting this basic structure more complex logic gates.

2.2.1 Static Characteristics

The range of input voltages can be divided into several regions.

nMOS 'off', pMOS 'on'

For $0 < V_i < V_{Tn}$ the n channel transistor is 'off', the p channel transistor is 'on' and the output voltage = V_{dd}. This is the normal digital operation range with input = '0' and output = '1'.

nMOS saturated, pMOS linear

In this regime, both transistors are 'on'. The input voltage V_i is $> V_{Tn}$, but is small enough so that the n channel transistor is in saturation, and the p channel transistor is in the linear regime. In static condition, the output voltage will adjust itself such that the currents through the n and p channel transistors are equal. The absolute value of gate-source voltage on the p channel transistor is V_{dd}- V_i, and therefore the "over voltage" on its gate is V_{dd}- V_i- V_{Tp}. The drain source voltage of the pMOS has an absolute value V_{dd}-V_o. Therefore,

$$I_d = K_p \left[(V_{dd} - V_i - V_{Tp})(V_{dd} - V_o) - \frac{1}{2}(V_{dd} - V_o)^2 \right] = \frac{K_n}{2}(V_i - V_{Tn})^2 \tag{2.1}$$

Where symbols have their usual meanings.

We define $\beta \equiv \mathrm{K_n/K_p}$. We make the substitution $V_{dp} \equiv V_{dd} - V_o$, where $V_{dp}$is the absolute value of the drain-source voltage for the p channel transistor. Then,

$$(V_{dd} - V_i - V_{Tp})V_{dp} - \frac{1}{2}V_{dp}^2 = \frac{\beta}{2}(V_i - V_{Tn})^2 \tag{2.2}$$

Which gives the quadratic

$$\frac{1}{2}V_{dp}^2 - V_{dp}(V_{dd} - V_i - V_{Tp}) + \frac{\beta}{2}(V_i - V_{Tn})^2 = 0 \tag{2.3}$$

Solutions to the quadratic are:

$$V_{dp} = (V_{dd} - V_i - V_{Tp}) \pm \sqrt{(V_{dd} - V_i - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{2.4}$$

These equations are valid only when the pMOS is in its linear regime. This requires that

$$V_{dp} \equiv V_{dd} - V_o \leq V_{dd} - V_i - V_{Tp}$$

Therefore, we must choose the negative sign. Thus

$$V_{dd} - V_o = (V_{dd} - V_i - V_{Tp}) - \sqrt{V_{dd} - V_i - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{2.5}$$

Therefore,

$$V_o = V_i + V_{Tp} + \sqrt{(V_{dd} - V_i - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{2.6}$$

Since V_o must be $\geq V_i + V_{Tp}$, the limit of applicability of the above result is given by

$$(V_{dd} - V_i - V_{Tp})^2 = \beta(V_i - V_{Tn})^2$$

That is, the solution for V_o is valid for

$$V_i \leq \frac{V_{dd} + \sqrt{\beta}V_{Tn} - V_{Tp}}{1 + \sqrt{\beta}} \tag{2.7}$$

In the case where we size the n and p channel transistors such that

$$K_n = K_p; \text{ so } \beta = 1$$

we have

$$V_o = (V_i + V_{Tp}) + \sqrt{(V_{dd} - V_{Tn} - V_{Tp})(V_{dd} - 2V_i + V_{Tn} - V_{Tp})} \tag{2.8}$$

with

$$V_i \leq \frac{V_{dd} + V_{Tn} - V_{Tp}}{2}$$

nMOS saturated, pMOS saturated

At the limit of applicability of eq. 2.7, when the input voltage is exactly at

$$V_i = \frac{V_{dd} + \sqrt{\beta}V_{Tn} - V_{Tp}}{1 + \sqrt{\beta}} \tag{2.9}$$

both transistors are saturated. Since the currents of both transistors are independent of their drain voltages in this condition, we do not get a unique solution for V_o by equating drain currents. The currents will be equal for all values of V_o in the range

$$V_i - V_{Tn} \leq V_o \leq V_i + V_{Tp}$$

Thus the transfer curve of an inverter shows a drop of $V_{Tn}+ V_{Tp}$ at a voltage near $V_{dd}/2$. This is actually an artifact of the simple transistor model chosen for this analysis, which assumes perfect saturation of drain current. In a real case, the drain current does depend on the drain voltage (albeit weakly) in the saturation region. If the model incorporates an Early Voltage like effect, the drop near the middle of the characteristic is more gradual.

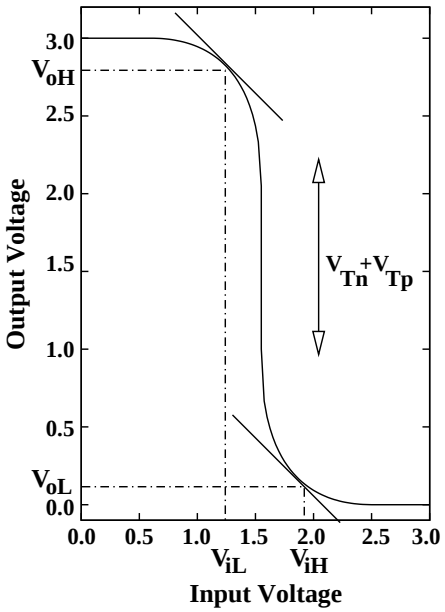


Figure 2.2: Transfer Curve of a CMOS inverter

nMOS linear, pMOS saturated

At the gate voltage given by eq. 2.9, both transistors are saturated. As we increase V_i beyond this value, such that

$$\frac{V_{dd} + \sqrt{\beta} V_{Tn} - V_{Tp}}{1 + \sqrt{\beta}} < V_i < V_{dd} - V_{Tp}$$

both transistors are still 'on', but nMOS enters the linear regime while pMOS gets saturated. Equating currents in this condition,

$$I_d = \frac{K_p}{2}(V_{dd} - V_i - V_{Tp})^2 = K_n \left[(V_i - V_{Tn}) V_o - \frac{1}{2} V_o^2 \right] \tag{2.10}$$

From this, we get the quadratic equation

$$\frac{1}{2} V_o^2 - (V_i - V_{Tn}) V_o + \frac{(V_{dd} - V_i - V_{Tp})^2}{2\beta} = 0 \tag{2.11}$$

This has solutions

$$V_o = (V_i - V_{Tn}) \pm \sqrt{(V_i - V_{Tn})^2 - \frac{(V_{dd} - V_i - V_{Tp})^2}{\beta}} \tag{2.12}$$

Since the equations are valid only when the n channel transistor is in the linear regime ($V_o < V_i - V_{Tn}$), we choose the negative sign. This gives,

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_i - V_{Tn})^2 - \frac{(V_{dd} - V_i - V_{Tp})^2}{\beta}} \tag{2.13}$$

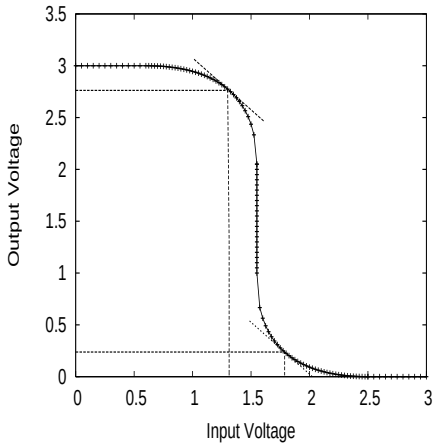
Again, in the special case where $\beta = 1$, we have

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_{dd} - V_{Tn} - V_{Tp})(2V_i - V_{dd} - V_{Tn} + V_{Tp})} \tag{2.14}$$

nMOS 'on', pMOS 'off'

As we increase the input voltage beyond V_{dd}- V_{Tp}, the p channel transistor turns 'off', while the n channel conducts strongly. As a result, the output voltage falls to zero. This is the normal digital operation range with input = '1' and output = '0'.

The figure below shows the transfer curve of an inverter with V_{dd}= 3V, V_{Tn}= 0.6V and V_{Tp}= 0.5V, and $\beta = 1$.



The plot produced by SPICE for this circuit with realistic models is quite similar.

2.2.2 Noise margins

The requirement from a digital circuit is that it should distinguish logic levels, but be insensitive to the exact analog voltage at the input. This implies that the flat portions of the transfer curve (where $\frac{\partial V_o}{\partial V_i}$ is small) are suitable for digital logic. We select two points on the transfer curve where the slope ($\frac{\partial V_o}{\partial V_i}$) is -1.0. The coordinates of these two points define the values of (V_{iL},V_{oH}) and (V_{iH},V_{oL}). Robust digital design requires that the output high level be higher than what is acceptable as a high level at the input ($V_{oH} > V_{iH}$). The difference

between these two levels is the 'high' noise margin. This is the amount of noise that can ride on the worst case 'high' output and still be accepted as a 'high' at the input of the next gate. Similarly, we require $V_{oL} < V_{iL}$. The difference, $V_{iL} - V_{oL}$ is the 'low' noise margin. Obviously, it is of interest to evaluate the values of these noise margins. For the discussion which follows, we shall use the expressions derived earlier for $\beta = 1$ to keep the algebra simple.

Calculation of V_{iL} and V_{oH}

from eq. (2.8)

$$V_o = (V_i + V_{Tp}) + \sqrt{(V_{dd} - V_{Tn} - V_{Tp})(V_{dd} + V_{Tn} - V_{Tp} - 2V_i)}$$

From this, we can evaluate $\frac{\partial V_o}{\partial V_i}$ and set it = -1.

$$\frac{\partial V_o}{\partial V_i} = -1 = 1 - \sqrt{\frac{V_{dd} - V_{Tn} - V_{Tp}}{V_{dd} + V_{Tn} - V_{Tp} - 2V_i}} \tag{2.15}$$

This gives

$$V_{iL} = \frac{3V_{dd} + 5V_{Tn} - 3V_{Tp}}{8} \tag{2.16}$$

Substituting this in eq.(2.8), we get

$$V_{oH} = \frac{7V_{dd} + V_{Tn} + V_{Tp}}{8} = V_{dd} - \frac{V_{dd} - V_{Tn} - V_{Tp}}{8} \tag{2.17}$$

Calculation of V_{iH} and V_{oL}

When the input is 'high', we should use eq.(2.14).

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_{dd} - V_{Tn} - V_{Tp})(2V_i - V_{dd} - V_{Tn} + V_{Tp})}$$

Differentiating with respect to V_i gives

$$\frac{\partial V_o}{\partial V_i} = -1 = 1 - \sqrt{\frac{V_{dd} - V_{Tn} - V_{Tp}}{2V_i - V_{dd} - V_{Tn} + V_{Tp}}} \tag{2.18}$$

From where, we get

$$V_{iH} = \frac{5V_{dd} + 3V_{Tn} - 5V_{Tp}}{8} \tag{2.19}$$

and

$$V_{oL} = \frac{V_{dd} - V_{Tn} - V_{Tp}}{8} \tag{2.20}$$

Calculation of Noise Margins

The high noise margin is given by

$$V_{oH} - V_{iH} = \frac{V_{dd} - V_{Tn} + 3V_{Tp}}{4} \tag{2.21}$$

Similarly, the Low noise margin is

$$V_{iL} - V_{oL} = \frac{V_{dd} + 3V_{Tn} - V_{Tp}}{4} \tag{2.22}$$

The two noise margins can be made equal by choosing equal values for V_{Tn} and V_{Tp}.

2.2.3 Dynamic Considerations

In this section, we analyze the dynamic behaviour of the inverter. For the calculation of rise and fall times, we shall assume that only one of the two transistors in the inverter is 'on'. (Notice that this is more conservative than the input high and low conditions determined by slope considerations in eq.2.19 and 2.16). We shall continue to use the simple model described at the beginning of this booklet.

Rise time

When the input is low, the n channel transistor is 'off', while the p channel transistor is 'on'. The equivalent circuit in this condition is shown in fig. 2.3. From Kirchoff's current law at

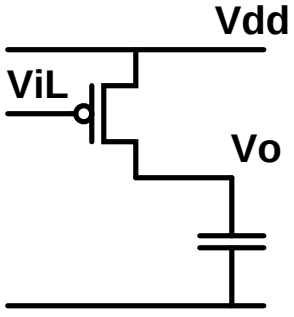


Figure 2.3: CMOS inverter with the nMOS 'off'

the output node,

$$I_{dp} = C\frac{dV_o}{dt}$$

so,

$$\frac{dt}{C} = \frac{dV_o}{I_{dp}}$$

This separates the variables, with the LHS independent of operating voltages and the RHS independent of time. Integrating both sides, we get

$$\frac{\tau_{rise}}{C} = \int_0^{V_{oH}} \frac{dV_o}{I_{dp}}$$

Till the output rises to $V_{iL}+ V_{Tp}$, the p channel transistor is in saturation. Since the current is constant, the integration is trivial. If $V_{oH} > V_{iL} + V_{Tp}$ (which is normally the case), the integration range can be broken into saturation and linear regimes. Thus

$$\begin{aligned}\frac{\tau_{rise}}{C} &= \int_0^{V_{iL}+V_{Tp}} \frac{dV_o}{\frac{K_p}{2}(V_{dd} - V_{iL} - V_{Tp})^2} \\ &+ \int_{V_{iL}+V_{Tp}}^{V_{oH}} \frac{dV_o}{K_p\left[(V_{dd} - V_{iL} - V_{Tp})(V_{dd} - V_o) - \frac{1}{2}(V_{dd} - V_o)^2\right]}\end{aligned}$$

We define $V_1 \equiv V_{dd} - V_o$ and $V_2 \equiv V_{dd} - V_{iL} - V_{Tp}$, so $dV_o = -dV_1$.
We get

$$\frac{K_p\tau_{rise}}{2C} = \frac{V_{iL} + V_{Tp}}{V_2^2} - \int_{V_2}^{V_{dd}-V_{oH}} \frac{dV_1}{2V_1V_2 - V_1^2}$$

The integral can be evaluated as

$$\begin{aligned}I &\equiv -\int_{V_2}^{V_{dd}-V_{oH}} \frac{dV_1}{2V_1V_2 - V_1^2} \\ &= \frac{1}{2V_2}\int_{V_{dd}-V_{oH}}^{V_2} \left(\frac{1}{V_1} + \frac{1}{2V_2 - V_1}\right) dV_1 \\ &= \frac{1}{2V_2}\left[\ln\frac{V_1}{2V_2 - V_1}\right]_{V_{dd}-V_{oH}}^{V_2} \\ &= \frac{1}{2V_2}\ln\frac{2V_2 - V_{dd} + V_{oH}}{V_{dd} - V_{oH}}\end{aligned}$$

Therefore,

$$\frac{K_p\tau_{rise}}{2C} = \frac{V_{iL} + V_{Tp}}{V_2^2} + \frac{1}{2V_2}\ln\frac{2V_2 - V_{dd} + V_{oH}}{V_{dd} - V_{oH}}$$

or

$$\frac{K_p\tau_{rise}}{2C} = \frac{V_{iL} + V_{Tp}}{(V_{dd} - V_{iL} - V_{Tp})^2} + \frac{1}{2(V_{dd} - V_{iL} - V_{Tp})}\ln\frac{2V_2 - V_{dd} + V_{oH}}{V_{dd} - V_{oH}}$$

Thus,

$$\begin{aligned}\tau_{rise} &= \frac{C(V_{iL} + V_{Tp})}{\frac{K_p}{2}(V_{dd} - V_{iL} - V_{Tp})^2} \\ &+ \frac{C}{K_p(V_{dd} - V_{iL} - V_{Tp})}\ln\frac{V_{dd} + V_{oH} - 2V_{iL} - 2V_{Tp}}{V_{dd} - V_{oH}}\end{aligned} \tag{2.23}$$

The first term is just the constant current charging of the load capacitor. The second term represents the charging by the pMOS in its linear range. This can be compared with resistive charging, which would have taken a charge time of

$$\tau = RC \ln \frac{V_{dd} - V_{iL} - V_{Tp}}{V_{dd} - V_{oH}}$$

to charge from V_{iL}+ V_{Tp} to V_{oH}.

Fall time

When the input is high, the n channel transistor is 'on' and the p channel transistor is 'off'. If

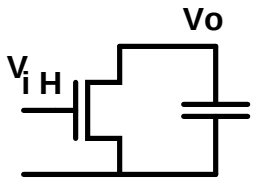


Figure 2.4: CMOS inverter with the pMOS 'off'

the output was initially 'high', it will be discharged to ground through the nMOS. To analysis the fall time, we apply Kirchoff's current law to the output node. This gives

$$I_{dn} = -C \frac{dV_o}{dt}$$

Again, separating variables and integrating from the initial voltage (= V_{dd}) to some terminal voltage V_{oL} gives

$$\frac{\tau_{fall}}{C} = -\int_{V_{dd}}^{voL} \frac{dV_o}{I_{dn}}$$

The n channel transistor will be in saturation till the output voltage falls to V_i- V_{Tn}. Below this voltage, the transistor will be in its linear regime. Thus, we can divide the integration range in two parts.

$$\begin{aligned} \frac{\tau_{fall}}{C} &= -\int_{V_{dd}}^{V_i - V_{Tn}} \frac{dV_o}{I_{dn}} - \int_{V_i - V_{Tn}}^{V_{oL}} \frac{dV_o}{I_{dn}} \\ &= \int_{V_i - V_{Tn}}^{V_{dd}} \frac{dV_o}{\frac{K_n}{2}(V_i - V_{Tn})^2} \\ &\quad + \int_{V_{oL}}^{V_i - V_{Tn}} \frac{dV_o}{K_n[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2} \end{aligned}$$

Therefore

$$\begin{aligned}\frac{K_n\tau_{fall}}{2C} &= \frac{V_{dd}-V_i+V_{Tn}}{(V_i-V_{Tn})^2}+\int_{V_{oL}}^{V_i-V_{Tn}}\frac{dV_o}{2V_o(V_i-V_{Tn})-V_o^2}\\ &= \frac{V_{dd}-V_i+V_{Tn}}{(V_i-V_{Tn})^2}+\frac{1}{2(V_i-V_{Tn})}\int_{V_{oL}}^{V_i-V_{Tn}}dV_o\left(\frac{1}{V_o}+\frac{1}{2(V_i-V_{Tn})-V_o}\right)\end{aligned}$$

Which gives

$$\begin{aligned}\frac{K_n\tau_{fall}}{2C} &= \frac{V_{dd}-V_i+V_{Tn}}{(V_i-V_{Tn})^2}+\frac{1}{2(V_i-V_{Tn})}\left[\ln\frac{V_o}{2(V_i-V_{Tn})-V_o}\right]_{V_{oL}}^{V_i-V_{Tn}}\\ &= \frac{V_{dd}-V_i+V_{Tn}}{(V_i-V_{Tn})^2}+\frac{1}{2(V_i-V_{Tn})}\ln\frac{2(V_i-V_{Tn})-V_{oL}}{V_{oL}}\end{aligned}$$

and therefore

$$\tau_{fall}=\frac{C(V_{dd}-V_i+V_{Tn})}{\frac{K_n}{2}(V_i-V_{Tn})^2}+\frac{C}{K_n(V_i-V_{Tn})}\ln\frac{2(V_i-V_{Tn})-V_{oL}}{V_{oL}} \tag{2.24}$$

Again, the first term represents the time taken to discharge at constant current in the saturation regime, whereas the second term is the quasi-resistive discharge in the linear regime.

2.2.4 Trade off between power, speed and robustness

As we scale technologies, we improve speed and power consumption. However, as we can see from the expression for noise margins, (eq 2.21 and eq 2.22) the noise margin becomes worse. We can improve noise margins by choosing relatively higher threshold voltages. However, this will reduce speeds. We could also increase V_{dd}- but that would increase power dissipation. Thus we have a trade off between power, speed and noise margins.

This choice is made much more complicated by process variations, because we have to design for the worst case.

2.2.5 CMOS Inverter Design Flow

The CMOS inverter forms the basis of most static CMOS logic design. More complex logic can be designed from it by simple thumb rules. A common (though not universal) design requirement is symmetric charge and discharge behaviour and equal noise margins for high and low logic values. This requires matched values of K_n and K_p and equal values of $V_{Tn}$and V_{Tp}. For a constant load capacitance, rise and fall times depend linearly on K_n and K_p. Thus it is a straightforward calculation to determine transistor geometries if speed requirements and technological parameters are given. However, as transistor geometries are made larger, self loading can become significant. We now have to model the load capacitance as

$$C_{Load}=C_{ext}+\alpha K_n$$

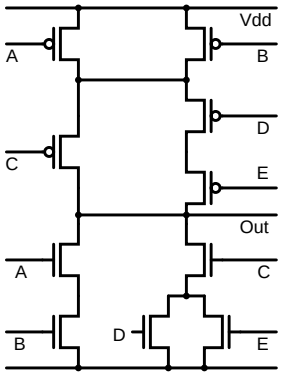


Figure 2.5: CMOS implementation of $\overline{A.B + C.(D + E)}$

where we have assumed that $\beta = K_n/K_p$ is kept constant. α is a technological constant. We use the expressions for $K\tau/C$ which depend only on voltages. Once these values are calculated, the geometry can be determined.

In the extreme case, when self capacitance dominates the load capacitance, K/C becomes constant and τ becomes geometry independent. There is no advantage in using wider transistors in this regime to increase the speed. It is better to use multi-stage logic with tapered buffers in this regime. This will be discussed in the module on Logical Effort.

2.2.6 Conversion of CMOS Inverters to other logic

Once the basic CMOS inverter is designed, other logic gates can be derived from it. The logic has to be put in a canonical form which is a sum of products with a bar (inversion) on top. For every '.' in the expression, we put the corresponding n channel transistors in series and the corresponding p channel transistors in parallel. for every '+', we put the n channel transistors in parallel and the p channel transistors in series. We scale the transistor widths up by the number of devices (n or p) put in series. The geometries are left untouched for devices put in parallel. Fig.2.5 shows the implementation of $\overline{A.B + C.(D + E)}$ in CMOS logic design style.

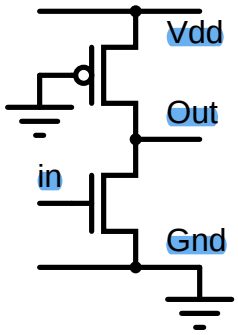
Chapter 3

Beyond Static CMOS

3.1 Pseudo nMOS Design Style

CMOS design style ensures that the logic consumes no static power. This is because the pull down and pull up networks are never 'on' simultaneously. However, this requires that signals have to be routed to the n pull down network *as well as* to the p pull up network. This means that the load presented to every driver is high. This fact is exacerbated by the fact that n and p channel transistors cannot be placed close together as these are in different wells which have to be kept well separated in order to avoid latchup.

Pseudo nMOS design style reduces dynamic power (by reducing capacitive loading) at the cost of having non-zero static power by replacing the pull up network by a single pMOS transistor with its gate terminal grounded. The pseudo nMOS inverter is shown below.



Notice that since the pMOS is not driven by signals, it is always 'on'. The effective gate voltage seen by the pMOS transistor is V_{dd}. Thus the overvoltage on the p channel gate is always V_{dd}- V_{Tp}. When the nMOS is turned 'on', a direct path between supply and ground exists and static power will be drawn.

3.1.1 Static Characteristics

As we sweep the input voltage from ground to V_{dd}, we encounter the following regimes of operation:

nMOS 'off'

This is the case when the input voltage is less than V_{Tn}. The output is 'high' and no current is drawn from the supply.

nMOS saturated, pMOS linear

As the input voltage is raised above V_{Tn}, we enter this region. The input voltage is assumed to be sufficiently low that the output voltage exceeds the saturation voltage $V_i - V_{Tn}$. Normally, this voltage will be higher than V_{Tp}, so the p channel transistor is in linear mode of operation. Equating currents through the n and p channel transistors, we get

$$K_p \left[(V_{dd} - V_{Tp})(V_{dd} - V_o) - \frac{1}{2}(V_{dd} - V_o)^2 \right] = \frac{K_n}{2}(V_i - V_{Tn})^2 \tag{3.1}$$

defining $V_1 \equiv V_{dd} - V_o$ and $V_2 \equiv V_{dd} - V_{Tp}$, we get

$$\frac{1}{2}V_1^2 - V_2 V_1 + \frac{\beta}{2}(V_i - V_{Tn})^2 = 0 \tag{3.2}$$

with solutions

$$V_1 = V_2 \pm \sqrt{V_2^2 - \beta(V_i - V_{Tn})^2}$$

substituting the values of V_1 and V_2 and choosing the sign which puts V_o in the correct range, we get

$$V_o = V_{Tp} + \sqrt{(V_{dd} - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{3.3}$$

nMOS linear, pMOS linear

As the input voltage is increased, the output voltage will decrease in accordance with equation(3.3). At some point, the output voltage will fall below $V_i - V_{Tn}$. It can be shown that this will happen when

$$V_i > V_{Tn} + \frac{V_{Tp} + \sqrt{V_{Tp}^2 + (\beta + 1)V_{dd}(V_{dd} - 2V_{Tp})}}{\beta + 1}.$$

The nMOS is now in its linear mode of operation. We shall not derive the expression for the output voltage in this mode of operation in the discussion here. The solution is straightforward, though algebraically tedious.

nMOS linear, pMOS saturated

As the input voltage is raised still further, the output voltage will fall below V_{Tp}. The pMOS transistor is now in saturation regime. Equating currents, we get

$$K_n \left[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2 \right] = \frac{K_p}{2}(V_{dd} - V_{Tp})^2$$

which gives

$$\frac{1}{2}V_o^2 - (V_o - V_{Tn})V_o + \frac{(V_{dd} - V_{Tp})^2}{2\beta}$$

This can be solved to get

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_i - V_{Tn})^2 - (V_{dd} - V_{Tp})^2/\beta} \tag{3.4}$$

3.1.2 Noise margins

As in the case of CMOS inverter, we find points on the transfer curve where the slope is -1.

When the input is low and output high, we should use eq(3.3). Differentiating this equation with respect to V_i and setting the slope to -1, we get

$$V_{iL} = V_{Tn} + \frac{V_{dd} - V_{Tp}}{\sqrt{\beta(\beta + 1)}} \tag{3.5}$$

and

$$V_{oH} = V_{Tp} + \sqrt{\frac{\beta}{\beta + 1}}\ (V_{dd} - V_{Tp}) \tag{3.6}$$

When the input is high and the output low, we use eq(3.4). Again, differentiating with respect to V_i and setting the slope to -1, we get

$$V_{iH} = V_{Tn} + \frac{2}{\sqrt{3\beta}}\ (V_{dd} - V_{Tp}) \tag{3.7}$$

and

$$V_{oL} = \frac{(V_{dd} - V_{Tp})}{\sqrt{3\beta}} \tag{3.8}$$

To make the output 'low' value lower than V_{Tn}, we get the condition

$$\beta > \frac{1}{3}\left(\frac{V_{dd} - V_{Tp}}{V_{Tn}}\right)^2$$

This condition on values of β places a requirement on the ratios of widths of n and p channel transistors. The logic gates work properly only when this equation is satisfied. Therefore this

kind of logic is also called 'ratioed logic'. In contrast, CMOS logic is called ratioless logic because it does not place any restriction on the ratios of widths of n and p channel transistors for static operation. The noise margin for pseudo nMOS can be determined easily from the expressions for V_{iL}, V_{oL}, V_{iH}, V_{oH}.

3.1.3 Dynamic characteristics

In the sections above, we have derived the behaviour of a pseudo nMOS inverter in static conditions. In the sections below, we discuss the dynamic behaviour of this inverter.

Rise Time

When the input is low and the output rises from 'low' to 'high', the nMOS is off. The situation is identical to the charge up condition of a CMOS gate with the pMOS being biased with its gate at 0V. This gives

$$\tau_{rise} = \frac{C}{K_p(V_{dd} - V_{Tp})}\left[\frac{2V_{Tp}}{V_{dd} - V_{Tp}} + \ln\frac{V_{dd} + V_{oH} - 2V_{Tp}}{V_{dd} - V_{oH}}\right] \tag{3.9}$$

Fall Time

Analytical calculation of fall time is complicated by the fact that the pMOS load continues to dump current in the output node, even as the nMOS tries to discharge the output capacitor.

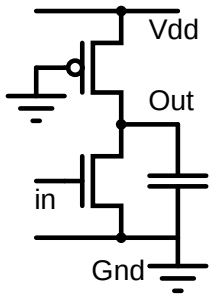


Figure 3.1: 'high' to 'low' transition on the output

Thus the nMOS should sink the discharge current as well as the drain current of the pMOS transistor. We make the simplifying assumption that the pMOS current remains constant at its saturation value through the entire discharge process. (This will result in a slightly pessimistic value of discharge time). Then,

$$I_p = \frac{K_p}{2}(V_{dd} - V_{Tp})^2$$

. We can write the KCL equation at the output node as:

$$I_n - I_p + C\frac{dV_o}{dt} = 0$$

which gives

$$\frac{\tau_{fall}}{C} = -\int_{V_{dd}}^{V_{oL}} \frac{dV_o}{I_n - I_p}$$

We define $V_1 \equiv V_i - V_{Tn}$ and $V_2 \equiv V_{dd} - V_{Tp}$. The integration range can be divided into two regimes. nMOS is saturated when $V_1 \leq V_o < V_{dd}$ and is in linear regime when $V_{oL} < V_o < V_1$. Therefore,

$$\frac{\tau_{fall}}{C} = -\int_{V_{dd}}^{V_1} \frac{dV_o}{\frac{1}{2}K_n V_1^2 - I_p} - \int_{V_1}^{V_{oL}} \frac{dV_o}{K_n(V_1 V_o - \frac{1}{2}V_o^2) - I_p}$$

so,

$$\frac{\tau_{fall}}{C} = \frac{V_{dd} - V_1}{\frac{1}{2}K_n V_1^2 - I_p} + \int_{V_{oL}}^{V_1} \frac{dV_o}{K_n(V_1 V_o - \frac{1}{2}V_o^2) - I_p}$$

3.1.4 Pseudo nMOS design Flow

We design the basic inverter first and then map the inverter design to other logic circuits. The load device size is calculated from the rise time. From eq. 3.9 we have

$$\tau_{rise} = \frac{C}{K_p(V_{dd} - V_{Tp})}\left[\frac{2V_{Tp}}{V_{dd} - V_{Tp}} + \ln\frac{V_{dd} + V_{oH} - 2V_{Tp}}{V_{dd} - V_{oH}}\right]$$

Given a value of τ_{rise}, operating voltages and technological constants, K_p and hence, the geometry of the p channel transistor can be determined.

Geometry of the n channel transistor in the reference inverter design can be determined from static considerations. Using eq. 3.4, the output 'low' level is given by:

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_i - V_{Tn})^2 - (V_{dd} - V_{Tp})^2/\beta}$$

If the desired value of the output 'low' level is given, we can calculate β. But $\beta \equiv K_n/K_p$ and K_p is already known. This evaluates K_n and hence, the geometry of the n channel transistor.

3.1.5 Conversion of pseudo nMOS Inverter to other logic

Once the basic pseudo nMOS inverter is designed, other logic gates can be derived from it. The procedure is the same as that for CMOS, except that it is applied only to nMOS transistors. The p channel transistor is kept at the same size as that for an inverter.

The logic is expressed as a sum of products with a bar (inversion) on top. For every '.' in the expression, we put the corresponding n channel transistors in series and for every '+', we

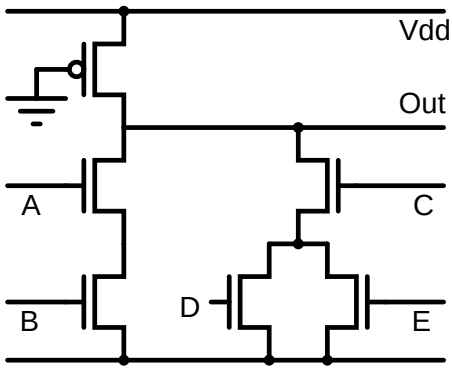


Figure 3.2: Pseudo NMOS implementation of $\overline{A.B + C.(D + E)}$

put the n channel transistors in parallel. We scale the transistor widths up by the number of devices put in series. The geometries are left untouched for devices put in parallel. Fig.3.2 shows the implementation of $\overline{A.B + C.(D + E)}$ in pseudo NMOS logic design style.

3.2 Complementary Pass gate Logic

This logic family is based on multiplexer logic.

Given a boolean function F(x1, x2, ..., xn), we can express it as:

$$F(x1, x2, \ldots, xn) = xi \cdot f1 + \overline{xi} \cdot f2$$

where f1 and f2 are reduced expressions for F with xi forced to '1' and '0' respectively. Thus, F can be implemented with a multiplexer controlled by xi which selects f1 or f2 depending on xi. f1 and f2 can themselves be decomposed into simpler expressions by the same technique.

To implement a multiplexer, we need both xi and $\overline{xi}$. Therefore, this logic family needs all inputs in true as well as in complement form. In order to drive other gates of the same type, it must produce the outputs also in true and complement forms. Thus each signal is carried by two wires. This logic style is called "Complementary Passgate Logic" or CPL for short.

3.2.1 Basic Multiplexer Structure

Pure passgate logic contains no 'amplifying' elements. Therefore, it has zero or negative noise margin. (Each logic stage degrades the logic level). Therefore, multiple logic stages cannot be cascaded. We shall assume that each stage includes conventional CMOS inverters to restore the logic level. Ideally, the multiplexer should be composed of complementary pass

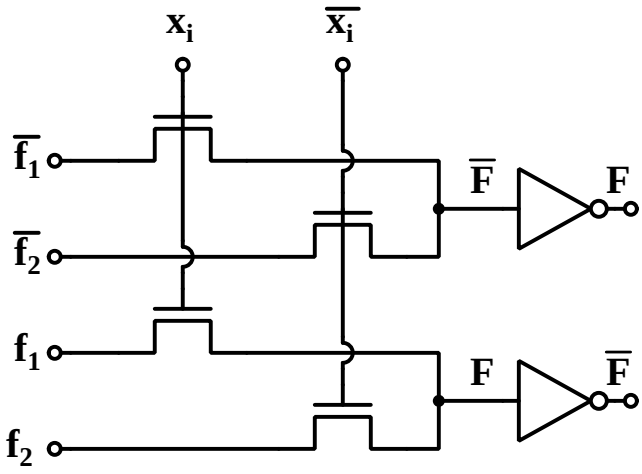


Figure 3.3: Basic Multiplexer with logic restoring inverters

gate transistors. However, we shall use just n channel transistors as switches for simplicity. This gives us the multiplexer structure shown in fig.3.3.

3.2.2 Logic Design using CPL

Since both true and complement outputs are generated by CPL, we do not need separate gates for AND and NAND functions. The same applies to OR-NOR, and XOR-XNOR functions.

To take an example, let us consider the XOR-XNOR functions. Because of the inverter, the multiplexer for the XOR output first calculates the XNOR function given by $A.B + \overline{A}.\overline{B}$. If we put A ='1', this reduces to B and for A = '0', it reduces to $\overline{B}$. Similarly, for the XNOR output, we generate the XOR expression = $A.\overline{B} + \overline{A}.B$ which will be inverted by the logic level restoring inverter. The expression reduces to $\overline{B}$ for A = '1' and to B for A = '0'. This leads to an implementation of XOR-XNOR as shown in fig.3.4

Implementation of AND and OR functions is similar. In case of AND, the multiplexer should output $\overline{A.B}$ to be inverted by the buffer. This reduces to $\overline{B}$ when A ='1'. When A = '0', it evaluates to 1 = $\overline{A}$. For NAND output, the multiplexer should output A.B, which evaluates to B for A = '1' and to '0' (or A) when A = '0'.

3.2.3 Buffer Leakage Current

The circuit configuration described above uses nMOS multiplexers. This limits the 'high' output of the multiplexer (node y - which is the input for the inverter) to V_{dd} - V_{Tn}. Conse-

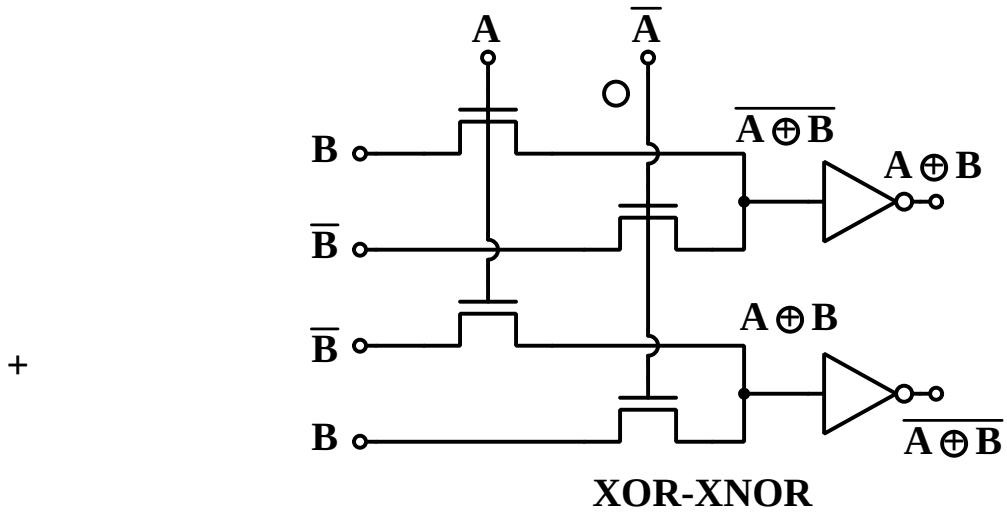


Figure 3.4: Implementation of XOR and XNOR by CPL logic.

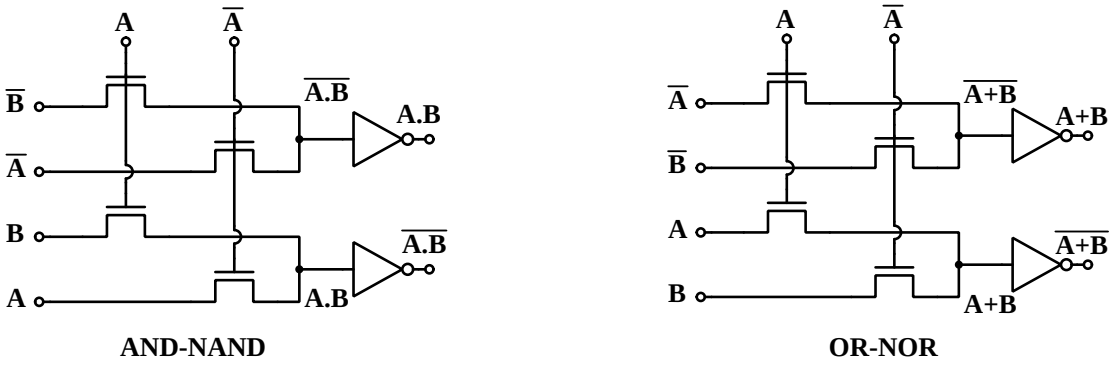


Figure 3.5: Implementation of (a) AND-NAND and (b) OR-NOR functions using complementary passgate logic.

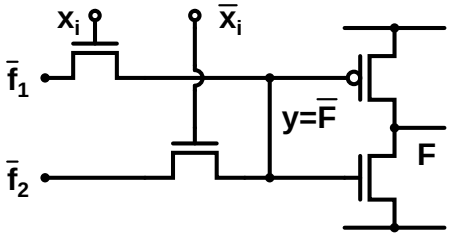


Figure 3.6: High leakage current in inverter

quently, the pMOS transistor in the buffer inverter never quite turns off. This results in static

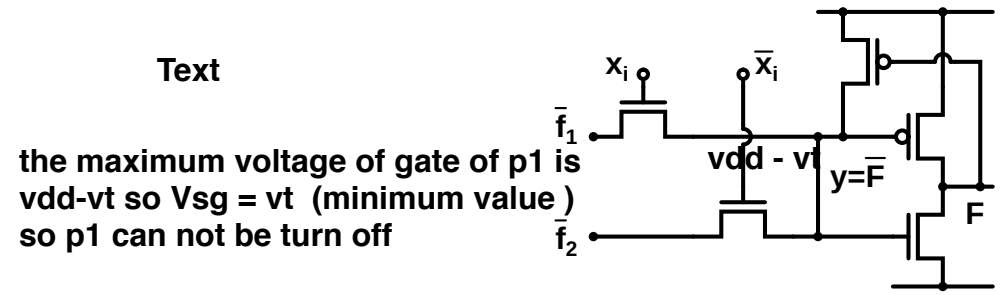


Figure 3.7: Pull up pMOS to avoid leakage in the inverter

power consumption in the inverter. This can be avoided by adding a pull up pMOS as shown in fig. 3.7. When the multiplexer output (y) is 'low', the inverter output is high. The pMOS is therefore off and has no effect. When the multiplexer output goes 'high', the inverter input charges up, the output starts falling and turns the pMOS on. Now, as the multiplexer output (y) approaches V_{dd} - V_{Tn}, the nMOS switch in the multiplexer turn off. However, the pMOS pull up remains 'on' and takes the inverter input all the way to V_{dd}. This avoids leakage in the inverter. However, this solution brings up another problem. Consider the equivalent

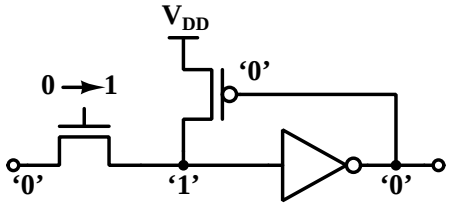


Figure 3.8: Problem with a low to high transition on the output

Text

circuit when the inverter output is 'low' and the pMOS is 'on'. Now if the multiplexer output wants to go 'low', it has to fight the pMOS pullup - which is trying to keep this node 'high'.

In fact, the multiplexer n transistor and the pull up p transistor constitute a pseudo nMOS inverter. Therefore, the multiplexer output cannot be pulled low unless the transistor geometries are appropriately ratioed.

3.3 Cascade Voltage Switch Logic

We can understand this logic configuration as an attempt to improve pseudo-nMOS logic circuits. Consider the NOR gate shown below: Static power is consumed by this NOR circuit

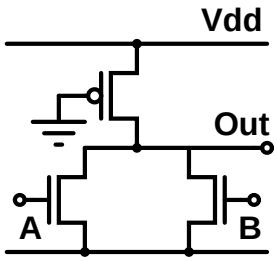


Figure 3.9: Pseudo-nMOS NOR

whenever the output is 'LOW'. This happens when A OR B is TRUE. We wish that the pMOS could be turned off for just this combination of inputs.

To turn the pMOS transistor off, we need to apply a 'HIGH' voltage level to its gate whenever A OR B is true. This obviously requires an OR gate. Non-inverting gates cannot be made in a single stage. However, We can create the OR function by using a NAND of $\overline{A}$ and $\overline{B}$ as shown in figure 3.10. But then what about the pMOS drive of this circuit?

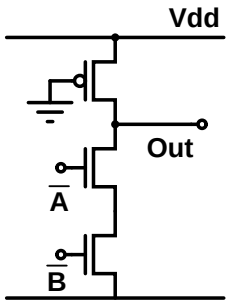


Figure 3.10: Pseudo-nMOS OR from complemented inputs

We want to turn the pMOS of this OR circuit off when both $\overline{A}$ and $\overline{B}$ are 'HIGH'; i.e. when A = B = 0. This means we would like to turn the pMOS of this circuit off when the NOR of A and B is 'TRUE'.

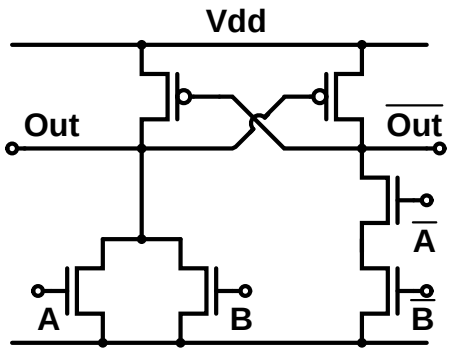


Figure 3.11: OR-NOR implementation in Cascade Voltage Switch Logic

But we already have this signal as the output of the first (NOR) circuit! So the two circuits can drive each other's pMOS transistors and avoid static power consumption. This kind of logic is called Cascade Voltage Switch Logic (CVSL). It can use any network f and its complementary network $\overline{f}$ in the two cross-coupled branches. The complementary network is constructed by changing all series connections in f to parallel and all parallel connections to series, and complementing all input signals.

CVSL shares many characteristics with static CMOS, CPL and pseudo-nMOS.

- Like CMOS static logic, there is no static power consumption.
- Like CPL, this logic requires both True and Complement signals. It also provides both True and complement outputs. (Dual Rail Logic).
- Like pseudo nMOS, the inputs present a single transistor load to the driving stage.
- The circuit is self latching. This reduces ratioing requirements.

3.4 Dynamic Logic

In this style of logic, some nodes are required to hold their logic value as a charge stored on a capacitor. These nodes are not connected to their 'drivers' permanently. The 'driver' places the logic value on them, and is then disconnected from the node. Due to leakage etc., the logic value cannot be held indefinitely. Dynamic circuits therefore require a *minimum* clock frequency to operate correctly. Use of dynamic circuits can reduce circuit complexity and power consumption substantially, compared to pseudo NMOS. When the clock is low, pMOS is on and the bottom nMOS is off. The output is 'pre-charged' to '1' unconditionally. When the clock goes high, the pMOS turns off and the bottom nMOS comes on. The circuit then

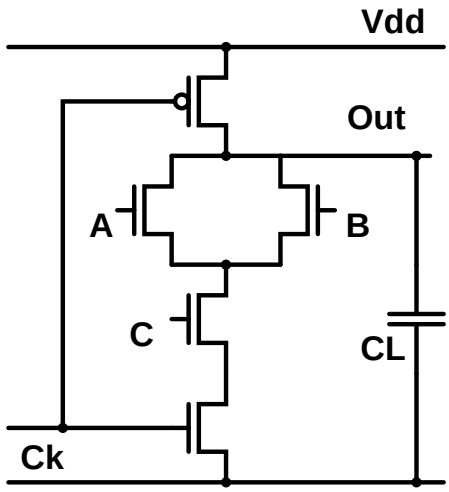


Figure 3.12: CMOS dynamic gate to implement $\overline{(A+B).C}$.

conditionally discharges the output node, if (A+B).C is TRUE. This implements the function $\overline{(A+B).C}$.

3.4.1 Problem with Cascading CMOS dynamic logic

The dynamic circuit described above can run into problems when several dynamic gates are cascaded. Consider the case when the circuit described in Fig. 3.12 is followed by an inverter. Let us take two cases – when (A+B).C is FALSE and when it is TRUE.

When (A+B).C is FALSE, There is no problem X pre-charges to '1' and remains at '1'. Therefore the inverter sees the correct logic value at its input all the time and discharges the output to '0', which is the expected output.

However, When (A+B).C is TRUE, there is a problem.
The correct value for X is now '0'. After the pre-charge cycle, X takes some time to discharge to '0'. So for some time after pre-charge, its output is held at the wrong value of '1' During this time, charge placed on the output leaks away as the input to nMOS of the inverter is not '0'. This can lead to a wrong evaluation of the logic function!

3.4.2 Four Phase Dynamic Logic

The problem can be solved by introducing additional clock phases. We use different phases for pre-charge, evaluation and for holding a valid output. Now we can sample the previous

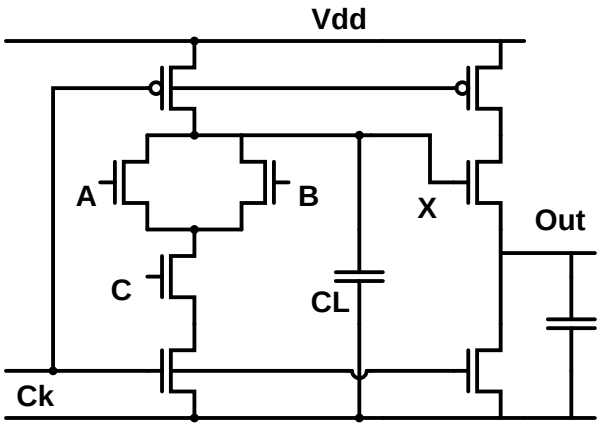
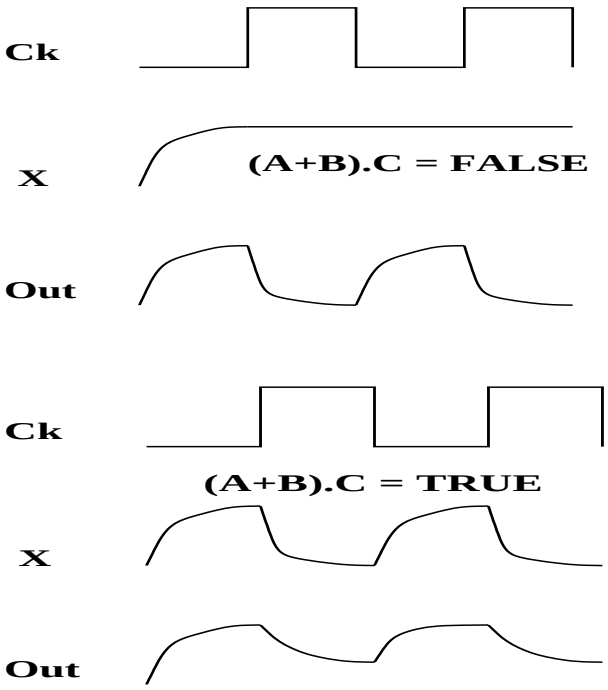


Figure 3.13: Dynamic gate for $\overline{(A+B).C}$ cascaded with an inverter.



stage only in the clock phase when evaluation is complete and it is valid.

For the circuit shown in Fig. 3.14, we have a 4 phase clock. Combined clock signals of the type Ck_{mn} are generated as required. Ck_{mn} is high during the m and n phases of the clock.

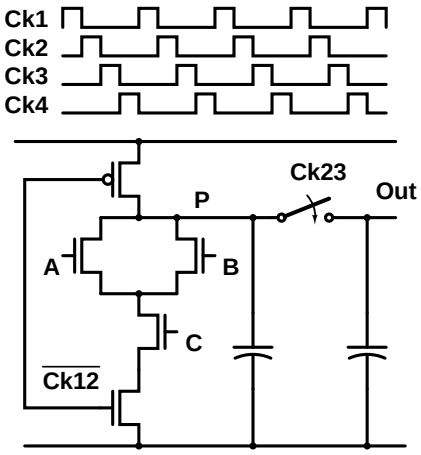


Figure 3.14: CMOS 4 phase dynamic logic

Now, for the gate shown in the figure,

1. In phase 1, node P is pre-charged.
2. In phase 2, P as well as the output are pre-charged.
3. In phase 3, The gate evaluates.
4. In phases 4 and 1, the output is isolated from the driver and remains valid.

This is called a type 3 gate. It evaluates in phase 3 and is valid in phases 4 and 1. Similarly, we can have type 4, type 1 and type 2 gates. A type 3 gate can drive a type 4 or a type 1 gate. Similarly, type 4 will drive types 1 and 2; type 1 will drive types 2 and 3; and type 2 will drive types 3 and 4. We can use a 2 phase clock if we stick to type 1 and type 3 gates (or type 2 and type 4 gates) as these can drive each other.

3.4.3 Domino Logic

Another way to eliminate the problem with cascading logic stages is to use a static inverter after the CMOS dynamic gate. Recall that cascading of dynamic CMOS stage causes problems because the output is pre-charged to V_{dd}. If the final value of a stage is meant to be zero, the next stage nMOS to which this output is connected erroneously sees a one till the pre-charged output is brought down to zero. During this time, it ends up discharging its own pre-charged output, which it was not supposed to do.

If an inverter is added, the output is held 'low' before logic evaluation. Now, if the final output of this gate is '0', there is no problem anyway. If the final output was supposed be

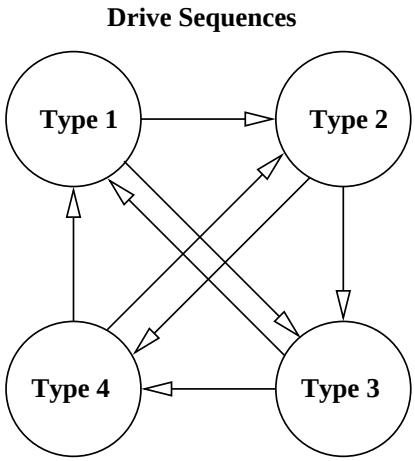


Figure 3.15: CMOS 4 phase dynamic logic drive constraints

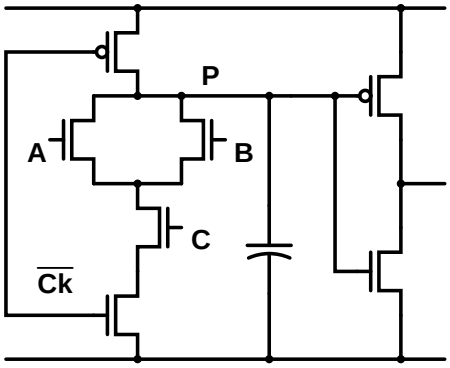


Figure 3.16: CMOS domino logic

'1', the next stage is erroneously held at zero for some time. However, this does not result in a false evaluation by the next stage. The only effect it can have is that the next stage starts its evaluation a little later.

Domino logic is fast, because the pre-charge cycle is common to all stages. Once pre-charge is done, each stage evaluates one after the other (hence the name - domino logic). Thus for n stage logic, there is only one cycle of pre-charge and n cycles of evaluation, rather than n cycles of pre-charge and n of evaluation.

However, the addition of an inverter means that the logic is non-inverting. Therefore, it cannot be used to implement any arbitrary logic function. In synchronous digital circuits,

combinational logic alternates with clocked latches. If inversion is required, we insert a latch at that place and take the $\overline{Q}$ output of the latch to the next group of domino logic.

3.4.4 Zipper logic

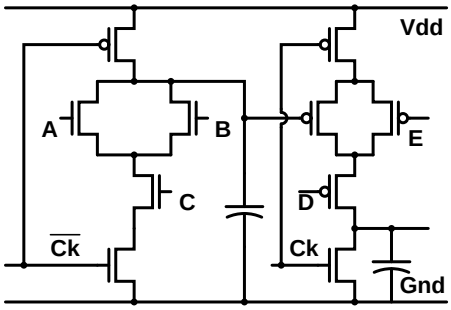


Figure 3.17: Zipper logic: A, B, C must be from a p stage, while D and E should be from an n stage.

Instead of using an inverter, we can alternate n and p evaluation stages. The n stage is pre-charged high, but it drives a p stage. A high pre-charged stage will keep the p evaluation stage off, which will not cause any malfunction. The p stage will be pre-discharged to 'low', which is safe for driving n stages. This kind of logic is called *zipper* logic.

Chapter 4

Multi-Stage Logic Design

We have seen how to design single stage logic in different design styles. However, typical designs need multi-stage combinational logic. In this chapter, we shall discuss techniques used for the design of multi-stage logic.

4.1 Stage Delay and Sizing

In chapter 2, we had derived expressions of the type

$$\frac{K\tau_L}{C_L} = \int_{V_1}^{V_2} \frac{dV}{f(V)}$$

for the delay of a single logic stage. Here τ_L is the time taken to charge/discharge the load capacitor C_L from V_1 to V_2 and K is the conductance factor given by $\mu C_{ox}\frac{W}{L}$.

The right hand side of this equation is a definite integral. It will evaluate to some constant depending on the voltages defining the 'High' and 'Low' logic levels, the supply voltage, turn on voltages, drain saturation voltage etc. In digital design, we shall keep the channel length at its minimum value, so L is a constant. Let us initially ignore the parasitic capacitances. We can see that

$$\frac{W\tau_L}{C_L} = \text{Constant} \qquad \text{so } \tau_L \propto \frac{C_L}{W}$$

This tells us that the delay associated with a gate charging a load capacitor scales directly with C_L and inversely with W, the width of the charging/discharging transistor. This linear dependence permits us to design logic stages easily.

4.2 Tapered Buffer

As an example of multi-stage logic, let us take the case when a large capacitor is to be driven by a CMOS circuit. A minimum sized inverter will take too long to charge this capacitor. Therefore, we would like to scale up an inverter (multiply all transistor width by a scale factor) in order to drive this large capacitor. However, the input capacitance of this scaled up inverter may be too large for a minimum sized inverter to drive. Therefore, we need a medium sized inverter to drive the large final inverter. We keep adding inverters, till the first inverter in the chain is small enough to be driven by standard CMOS logic. This kind of buffering is referred to as a *tapered buffer*.

How do we decide the number of inverters to include in this chain? And what should be the

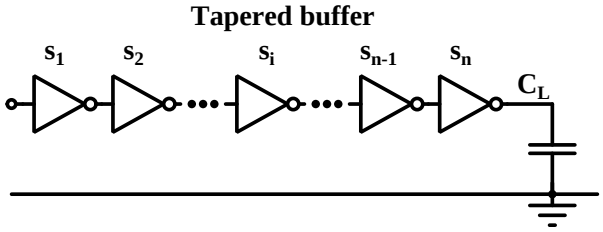


Figure 4.1: A tapered buffer

scale factors for each successive stage to minimize the total delay?

Let the i'th inverter in the chain be scaled up by a factor s_i relative to a minimum sized inverter. Let the delay of a minimum sized inverter driving another minimum sized inverter be τ. Then the i'th inverter provides charging currents which are s_i times the minimum sized inverter. However, the load it sees is s_{i+1} times the input capacitance of a minimum inverter. Therefore the delay associated with the i'th stage is $\frac{s_{i+1}}{s_i}\tau$. The total delay of the inverter chain is given by

$$d_{total} = \sum_{1}^{n} \frac{s_{i+1}}{s_i}\tau = \tau \sum_{1}^{n} \frac{s_{i+1}}{s_i} \tag{4.1}$$

In order to minimize the total delay, we should put the partial derivative with respect to each of the s_i equal to zero. Therefore,

$$\tau \frac{d}{ds_i}\left(\frac{s_2}{s_1} + \cdots + \frac{s_i}{s_{i-1}} + \frac{s_{i+1}}{s_i} + \cdots\right) = 0 \tag{4.2}$$

Only two terms in the sum contain s_i. Since all scale factors s_i are independent, the derivative of all the rest of the terms is 0. Therefore,

$$\frac{1}{s_{i-1}} - \frac{s_{i+1}}{s_i^2} = 0 \qquad \text{Which gives:} \qquad \frac{s_i}{s_{i-1}} = \frac{s_{i+1}}{s_i} \tag{4.3}$$

This means that the *stage ratio*, which is the factor by which an inverter is larger than the previous one, should be the same for all stages.

Let this constant stage ratio for the tapered buffer be ρ. The delay contributed by the i'th stage is $\frac{s_{i+1}}{s_i}\tau = \rho\tau$. The first stage is a unit inverter. Each subsequent stage has a drive capability which is ρ times the drive capability of the previous stage. Since the drive capability is being stepped up by ρ in n stages, we should have

$$\rho^n = \frac{C_L}{C_{in}} \qquad \text{so } \rho = \left(\frac{C_L}{C_{in}}\right)^{1/n} \tag{4.4}$$

We define the ratio $H \equiv C_L/C_{in}$. Then, $\rho^n = H$ and so, $n = \ln H/\ln\rho$.
The total delay is given by

$$d_{total} = \sum_1^n \frac{s_{i+1}}{s_i}\tau = \sum_1^n \rho\tau = n\rho\tau \tag{4.5}$$

We want to find the value of ρ which will minimize the total delay. Note that n and ρ are not independent since $\rho^n = H$. Taking logarithms on both sides, we get

$$n\ln\rho = \ln H, \qquad \text{so} \qquad n = \frac{\ln H}{\ln\rho} \tag{4.6}$$

The total delay can then be expressed as

$$d_{total} = \frac{\ln H}{\ln\rho}\rho\tau = \tau\ln H\frac{\rho}{\ln\rho} \tag{4.7}$$

Total delay will be minimized with respect to ρ when its derivative with respect to ρ is 0. This gives

$$\tau\ln H\left(\frac{1}{\ln\rho} - \frac{\rho}{(\ln\rho)^2}\frac{1}{\rho}\right) = 0 \tag{4.8}$$

This leads to

$$\frac{1}{\ln\rho} = \frac{1}{(\ln\rho)^2}, \qquad \text{so} \qquad \ln\rho = 1, \quad \text{or} \quad \rho = e \tag{4.9}$$

$$\text{Since} \quad \rho = e, \qquad n = \frac{\ln H}{\ln\rho} = \ln H \tag{4.10}$$

Thus we obtain the result that the optimum stage ratio for a tapered buffer is e, while the optimum number of stages in the buffer is given by $\ln(C_{out}/C_{in})$.
(The optimum stage ratio comes out higher (≈ 3 to 4), if we take self loading into account.)

These results were computed for a situation where capacitors were driven only by inverters and self loading due to transistors in the gate was ignored. The general situation will differ from the tapered inverter chain in several respects:

1. Multi-stage logic can use gates other than inverters.

2. Each gate will have a parasitic delay associated with it due to the capacitance of the driver transistors themselves.

3. In the tapered inverter chain, each stage drives another inverter. In general, we can have branching, where a stage drives multiple gates. (Fanout is greater than 1).

The generalized optimization which removes these restrictions was developed by Ivan E. Sutherland and Robert F Sproull in a paper "Logical Effort: Designing for Speed on the Back of an Envelope.", published in the Proceedings of the Conference on Advanced Research in VLSI, held in March 1991. The method was later described in much greater detail in a book by Sutherland, Sproull and Davis titled "Logical Effort: Designing Fast CMOS Circuits", published by Morgan Kaufmann in 1998. The material in this chapter is largely based on that book.

First, Let us see what happens if we replace inverters in the tapered buffer with static CMOS logic gates.

4.3 Using Logic Gates Other Than Inverters

Transistor sizes in a gate are adjusted based on several considerations.

1. Difference in mobility of PMOS and NMOS. A PMOS transistor has to be wider than an NMOS transistor to provide the same drive current because the hole mobility is lower than electrons mobility. For example, we may scale the width of PMOS transistors to be double that of NMOS transistors connected in the same configuration.

2. If there are n series connected transistors, their widths should be scaled up by n. in order to make the output drive of the logic gate equivalent to an unscaled inverter.

3. After accounting for mobility differences and series connections, we may scale up *all* transistors by some factor in order to drive larger loads.

However, this has an an impact on the capacitive loading placed on the *previous* stage. Let the ratio of widths of p and n channel transistors in a minimal inverter to get equal rise and fall times be γ. We express capacitive loading in units of capacitance of a minimum sized n channel MOS transistor. As we can see from Fig.4.2, an inverter places a load of $(1+\gamma)$ units on the previous gate, a 2 input NAND gate with the same output drive loads the previous stage with a capacitance of $2+\gamma$ units and a 2 input NOR gate loads the previous stage with a capacitance of $1+2\gamma$ units.For example if $\gamma = 2$, the input capacitance of an inverter is 3 units, that of a 2 input NAND is 4 units, while a 2 input NOR presents a capacitance of 5

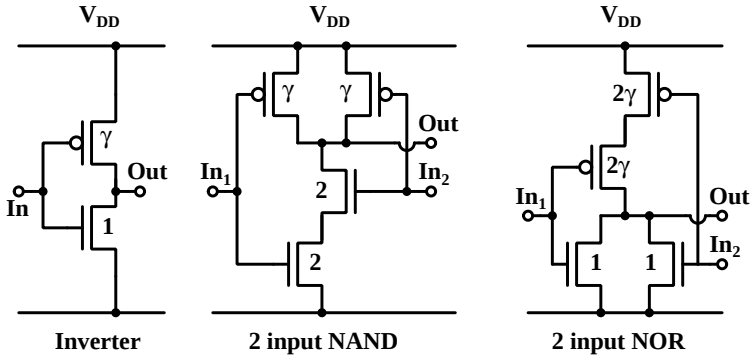


Figure 4.2: Load presented by 2 input NAND and NOR gates to their drivers.

units to its driver.

Thus we must account for a loading factor for different gate types when optimizing multi-stage logic. The ratio of $(2+\gamma)/(1+\gamma)$ for a 2 input NAND gate and of $(1+2\gamma)/(1+\gamma)$ for a 2 input NOR gate is independent of eventual scaling for drive capability. A NAND gate scaled to provide the same drive as a scaled up inverter will also present a load which is $(2+\gamma)/(1+\gamma)$ times higher on the previous stage as compared to the corresponding scaled up inverter. The same can be said for the NOR gate or in fact, for any other CMOS gate.

This correction factor of $(2+\gamma)/(1+\gamma)$ for a 2 input NAND or of $(1+2\gamma)/(1+\gamma)$ for a 2 input NOR gate is called the *logical effort* of this gate. This factor should be multiplied with the scale factor of this gate.

4.3.1 Delay model for a single gate

4.3.2 Considering the Effects of Self-Loading

If we consider the effects of self-loading, the logic has to drive some additional capacitance coming from the drain capacitance and Miller capacitance associated with the driver transistor. This additional capacitance is proportional to W. Thus,

$$C_L = C_{ext} + WC_p$$

Where C_p is the parasitic capacitance per unit width of driver transistors. Therefore the delay of the stage is

$$\tau_L \propto \frac{C_L}{W} = \frac{C_{ext} + WC_p}{W} \quad \text{Which gives} \quad \tau_L \propto \frac{C_{ext}}{W} + C_p \tag{4.11}$$

Thus the parasitic delay associated with self-loading is scale independent.
These refinements for computing the delay of a logic gate are encapsulated in the idea of "Logical Effort".

4.3.3 Gate Delays with Logical Effort

In the method of Logical Effort, we deal with normalized delays. The unit of time is taken to be the delay of a minimum sized inverter driving another minimum sized inverter *without any parasitic elements*.

As discussed in the section above, the delay of a gate can be expressed as

$$d = f + p \tag{4.12}$$

where f is the *effort delay*, which depends on transistor currents and load capacitance, while p is the parasitic delay, which is size independent. We further express f as a product of two quantities, g and h. Here h is the *electrical effort* of this stage. This is given by the ratio of output capacitance to input capacitance. This is equivalent to the *stage ratio* we had used while discussing the tapered inverter. g is the *logical effort* which accounts for the extra loading caused by a gate as compared to an inverter, as discussed in the previous section. So,

$$f = gh \tag{4.13}$$

Combining Equations 4.12 and 4.13, we can express the delay introduced by a logic gate as

$$d = gh + p \tag{4.14}$$

where the delay is measured in units of τ, which is the delay of a minimum inverter driving another minimum inverter *not including the parasitic delay*.

This way of expressing delays separates the effects of different components which cause delay, so these can be handled independently.

1. Dependence on technology is encapsulated in τ. All delays are expressed as a multiple of this quantity. Thus delays in this formulation are dimensionless numbers and must be multiplied with τ to get the absolute value of delay.

2. Dependence on sizing is encapsulated in h. This is the only component of delay which is size dependent. h is also a unit less quantity, because it is defined as the ratio of output capacitance to input capacitance.

3. Dependence on logic type is expressed through g. This accounts for the series/parallel configuration of transistors in a particular gate.

4. The effect of parasitic delays due to the load presented by the driver transistors themselves is expressed through p. p is size independent. It is also a dimensionless quantity as this delay is also expressed in units of τ.

4.3.4 Logical Effort for Common CMOS Gates

The logical effort of an inverter is, by definition, 1. The logical effort of other logic functions depends on their circuit topology. As we had seen in Fig.4.2 the logical effort of a two input NAND gate is $(2+\gamma)/(1+\gamma)$ while that of a 2 input NOR gate is $(1+2\gamma)/(1+\gamma)$.

n input NAND and NOR gates are shown in the figure below. The n input NAND gate

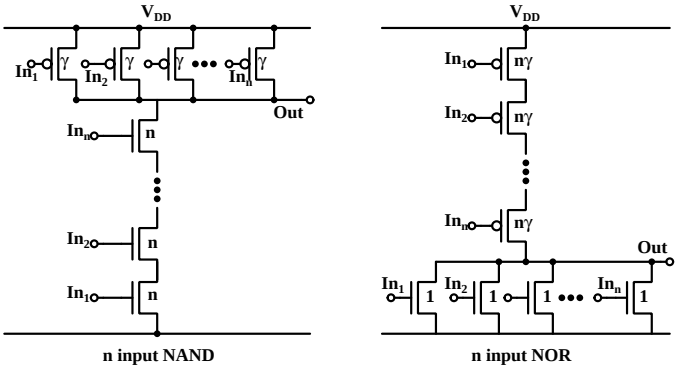


Figure 4.3: n input NAND and NOR gates

has n NMOS transistors in series and n PMOS transistors in parallel. Therefore each NMOS should have a width which is n times that of the minimum inverter. Each PMOS will have the same width as that of the minimum inverter – which is γ times the minimum size (to account for lower hole mobility). Taking the capacitance of a minimum sized transistor as the unit of capacitance, each input of the n input NAND gate loads its driver with $(n+\gamma)$ units of capacitance. The minimum inverter loads its driver by $(1+\gamma)$ units. So the logical effort of an N input NAND gate is $(n+\gamma)/(1+\gamma)$. This reduces to $(2+\gamma)/(1+\gamma)$ for a 2 input NAND, as expected.

We can approximate the parasitic delay by considering only the self capacitance which is directly connected to the output. In case of an inverter, this is proportional to $(1+\gamma)$. For the n input NAND gate, the capacitance contributed by the nMOS transistor at the output node is n, while the capacitance from the n pMOS transistors is $n\gamma$. Therefore, the parasitic delay of the n input NAND is related to the parasitic delay of the inverter by the factor $(n+n\gamma)/(1+\gamma) = n$. Thus we have

$$p_{NAND} = np_{inv} \qquad \text{for an n input NAND gate} \tag{4.15}$$

The n input NOR gate has n PMOS transistors in series, each of which should have $n\gamma$ times the minimum width. It has n NMOS transistors in parallel, each of which can have the minimum geometry. Thus the loading on the driver of each input is $(1+n\gamma)$ units. Thus the

logical effort of the N input NOR gate is $(1 + n\gamma)/(1 + \gamma)$. This reduces to $(1 + 2\gamma)/(1 + \gamma)$ for a 2 input NOR as expected.

For the parasitic delay of the n input NOR gate, the capacitance contributed by the n nMOS transistors at the output node is n, while the capacitance from the pMOS transistor is $n\gamma$. Therefore, the parasitic delay of the n input NOR gate is related to the parasitic delay of the inverter by the factor $(n + n\gamma)/(1 + \gamma) = n$. Thus we have

$$p_{NOR} = np_{inv} \qquad \text{for an n input NOR gate} \tag{4.16}$$

A 2 way multiplexer is shown in Fig. 4.4 below. It is clear that each signal input is loaded

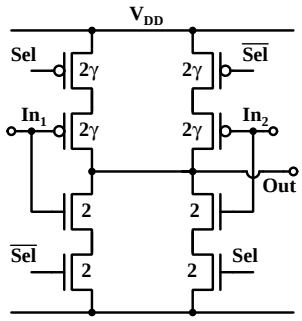


Figure 4.4: A 2 way multiplxer

with a capacitance of $(2 + 2\gamma)$ units. Therefore the logical effort for either data input in this mux is $(2 + 2\gamma)/(1 + \gamma) = 2$. The logical effort for the control inputs Sel and $\overline{Sel}$ is also 2. The parasitic delay for the 2 way mux is $2 + 2\gamma + 2 + 2\gamma$ which is 4 times the parasitic delay of an inverter.

It is interesting to see that the logical effort will remain 2 for every data input even when we parallel n tri-stateable inverters to form an n way mux. However the parasitic delay will increase to $2np_{inv}$ if we add n units.

The table below lists the logical effort and the parasitic delays of common logic gates. Notice that the unit of time is the delay of a minimum inverter driving another minimum inverter (excluding parasitic delay). Therefore in these units, the parasitic delay of an inverter, which is the ratio the parasitic delay (in absolute units) to the inverter delay without taking parasitic delay into account, need not be 1. It is convenient to specify the parasitic delay of other gates as multiple of the parasitic delay of an inverter p_{inv}.

Gate	Logical Effort	Parasitic Delay
Inverter	1	p_{inv}
2 input NAND	$(2+\gamma)/(1+\gamma)$	$2p_{inv}$
n input NAND	$(n+\gamma)/(1+\gamma)$	np_{inv}
2 input NOR	$(1+2\gamma)/(1+\gamma)$	$2p_{inv}$
n input NOR	$(1+n\gamma)/(1+\gamma)$	np_{inv}
2 way mux	2	$4p_{inv}$
n way mux	2	$2np_{inv}$

Table 4.1: Logical effort and parasitic delays of common gates

4.4 Design of multi-stage logic

In multi-stage logic, we consider the logical effort g_i and electrical effort h_i of each stage. We define the *path logical effort* as the product of logical effort of all stages on a given path.

$$G = \prod_{i=1}^{N} g_i \tag{4.17}$$

Similarly, we define the *path electrical effort* as the product of electrical effort of all stages on a given path.

$$H = \prod_{i=1}^{N} h_i \qquad \text{where} \qquad h_i = \frac{C_{out_i}}{C_{in_i}} \quad \text{for stage i.} \tag{4.18}$$

If there is no branching, the load on a particular stage is just the input capacitance of the next stage, that is: $C_{out_i} = C_{in_{i+1}}$. When we multiply all electrical efforts along a path, all except the first and last capacitances cancel. Thus we get

$$H = \frac{C_L}{C_{in_1}} \tag{4.19}$$

Where C_L is the final load capacitance and C_{in_1} is the input capacitance of the first stage. If, however, there is branching at some stage i, $C_{out_i} \neq C_{in_{i+1}}$. This is because C_{out_i} includes not only $C_{in_{i+1}}$ but also the input capacitance of other logic gates which are not on the path under consideration.

4.4.1 Branching Effort

We introduce a new kind of effort, named *branching effort*, which corrects for the fact that $C_{out_i} \neq C_{in_{i+1}}$ at certain points in the logic chain. Suppose there is branching at the i'th stage of the logic chain. The actual capacitive loading on the i'th stage is $C_{onpath} + C_{offpath}$, where $C_{onpath} = C_{in_{i+1}}$ and $C_{offpath}$ is the sum of all the other input capacitances connected

to the output of the i'th stage. It is convenient to continue to define H by Eqn. 4.19 even in the branching case. We now define a correction factor which is called the branching effort. Branching effort b_i at the output of a logic gate is defined to be:

$$b = \frac{C_{onpath} + C_{offpath}}{C_{onpath}} \tag{4.20}$$

where $C_{onpath} = C_{in_{i+1}}$, is the load capacitance of the gate being driven along the path being analyzed, while $C_{offpath}$ is the capacitance presented by the gates which are not on the path.

The output capacitance included in the definition of H is now multiplied by this correction factor at every stage.

$$b_i h_i = \frac{C_{onpath_i} + C_{offpath_i}}{C_{onpath_i}} \times \frac{C_{onpath_i}}{C_{in_i}} \tag{4.21}$$

If there is no branching at a stage, the corresponding b_i value is 1 since $C_{offpath} = 0$ and so, multiplication by b_i does not change any thing. If, however, there is branching at the i'th stage, multiplying h_i by b_i corrects the output capacitance, because

$$b_i h_i = \frac{C_{onpath_i} + C_{offpath_i}}{C_{onpath_i}} \times \frac{C_{onpath_i}}{C_{in_i}} = \frac{C_{onpath_i} + C_{offpath_i}}{C_{in_i}} \tag{4.22}$$

We define the branching effort of the whole path, denoted by B, as the product of the branching effort at each stage along the path.

$$B = \prod_{i=1}^{N} b_i \tag{4.23}$$

We can now define the *path effort*, F as the product of all logical efforts and branch corrected electrical efforts.

$$F = \prod_{i=1}^{N} g_i \prod_{i=1}^{N} b_i h_i = \prod_{i=1}^{N} g_i \prod_{i=1}^{N} b_i \prod_{i=1}^{N} h_i \tag{4.24}$$

So,

$$F = GBH \qquad \text{where } G = \prod_{i=1}^{N} g_i, B = \prod_{i=1}^{N} b_i, \text{ and } H = \prod_{i=1}^{N} h_i \tag{4.25}$$

The advantage of using branching correction separately from electrical effort is that H retains its cancellation property for capacitance of intermediate stages and is defined only by the final load and the input capacitance.

$$H = \frac{C_L}{C_{in_1}} \tag{4.26}$$

The equation that defines the path effort looks quite similar to the definition of the stage effort in Equation 4.13, which defined the effort for a single logic gate. Notice, however, that

unlike the stage effort f, the path effort F does not define the delay of the path. The total delay is the *sum* of individual delays and not their product.

$$D = \sum d_i = \sum g_i b_i h_i + \sum p_i \tag{4.27}$$

Still, F is a useful quantity for optimisation of path delays as we shall see.

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 4.14), we separate the contribution to path delay from effort and the delay due to parasitic capacitances.

$$D = \sum d_i = D_F + P \tag{4.28}$$

where D_F is the path effort delay, while P is the *path parasitic delay.* The path effort delay is simply $D_F = \sum g_i b_i h_i$ and the path parasitic delay is $P = \sum p_i$. In order to choose the optimum sizes of gates to minimize the total delay through the path, we need to minimize the above expression with respect to the transistor widths (and hence capacitances) of every stage. We have

$$D = \sum d_i = D_F + P = \sum g_i b_i h_i + P = \sum g_i b_i \frac{C_{i+1}}{C_i} + P \tag{4.29}$$

Notice that p_i (and hence P) are size independent and we can only minimize D_F by choosing optimum sizes of gates. Setting the derivative of D with respect to C_i to zero, and noticing that only two terms in the series expansion of D_F involve C_i, we can write

$$0 = \frac{\partial}{\partial C_i}\left(g_{i-1} b_{i-1} \frac{C_i}{C_{i-1}} + g_i b_i \frac{C_{i-1}}{C_i}\right) \tag{4.30}$$

This leads to

$$g_{i-1} b_{i-1} \frac{1}{C_{i-1}} - g_i b_i \frac{C_{i+1}}{C_i^2} = 0 \tag{4.31}$$

Which gives

$$g_{i-1} b_{i-1} \frac{C_i}{C_{i-1}} = g_i b_i \frac{C_{i+1}}{C_i} \qquad \text{hence} \qquad f_{i-1} = f_i \tag{4.32}$$

Thus, *the path delay is minimized when each stage in the path has the same stage effort, f.* The value of f which minimizes the path delay is denoted as $\hat{f}$. Since we had defined F as $F = \prod_{i=1}^{N} f_i$, in the optimum case, $F = \hat{f}^N$. In other words, the minimum delay is achieved when each stage effort is

$$\hat{f} = b_i g_i h_i = F^{1/N} \tag{4.33}$$

For this optimum effort, we obtain

$$\hat{D} = N F^{1/N} + P = N(GBH)^{1/N} + P \tag{4.34}$$

$\hat{D}$ is the minimum delay possible for this path. We achieve this minimum delay by appropriate sizing of transistors in each stage of the logic path. Transistor sizes must be so chosen that the stage delay f is the same for all stages in the logic path. Equations 4.33 and 4.13 combine to require that each logic stage be designed so that

$$\hat{h}_i \equiv \frac{C_{out_i}}{C_{in_i}} = \frac{F^{1/N}}{b_i g_i} \tag{4.35}$$

We start with the last stage, where the output capacitance is known (=C_L). Since the output capacitance is known, the input capacitance can be calculated from Equation 4.35. This gives the scale factor for this stage from which, geometries of transistors can be computed.

Since $C_{out_i} = b_i C_{in_{i+1}}$, we can write the recursive relation

$$C_{in_i} = g_i b_i \frac{C_{in_{i+1}}}{F^{1/N}} \tag{4.36}$$

C_{in} for every stage can now be determined using the above recursive relation (Equation 4.36). From C_{in}, we can calculate the geometry of transistors for this stage.

We can equivalently start from the input side, computinge the output capacitance and hence the input capacitance of the next stage. This can be continued till we reach the final output.

4.4.2 An example: 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? Let us take $\gamma = 2$ and $p_{inv} = 0.6$ for this example. The parasitic delay of

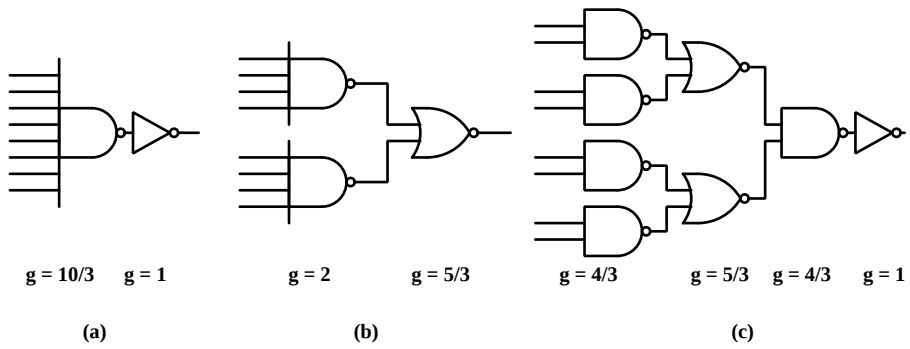


Figure 4.5: Three circuits that compute the AND function of eight inputs.

n input NANDs and NORs will then be $0.6n$. 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 configuration a, $6/3 \times 5/3 = 3.33$ for configuration b, and $4/3 \times 5/3 \times 4/3 \times 1 = 2.96$ for configuration c. These figures can be used in the delay equation (4.34) to find the minimum delay that can be obtained from each circuit. The following equations also include an estimate of parasitic delays, obtained by summing the parasitic delays of each of the logic gates along the path:

$$\text{Configuration a} \quad D = 2(3.33H)^{1/2} + 5.4 \tag{4.37}$$

$$\text{Configuration b} \quad D = 2(3.33H)^{1/2} + 3.6 \tag{4.38}$$

$$\text{Configuration c} \quad D = 4(2.96H)^{1/4} + 4.2 \tag{4.39}$$

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

Choosing between configurations b and c depends on the electrical effort H, that must be borne by the network. When $H = 1$, delay in configuration b is 7.25, and in configuration c is 9.5. Therefore in this case, configuration b will be best. However, if $H = 12$, the delay in configuration b is 16.24, while for configuration c it is 13.97. Thus, configuration c will be best in this case. The equations show that for high electrical effort, configuration c yields the 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 $C_{out} = 48$ (so that $H = 12$). The preceding computation showed that configuration c should be selected for $H = 12$. We have already computed G for this configuration, which is the product of logical efforts of all stages and its value is $4/3 \times 5/3 \times 4/3 \times 1 = 80/27 = 2.963$. There is no branching in this circuit, so all $b_i = 1$ and hence $B = 1$. Thus $F = GBH = 2.963 \times 1 \times 12 = 35.556$ and correspondingly, the optimum stage effort for this four stage circuit is $35.556^{1/4} = 2.442$.

Let us work forward along the path, starting with the 2-input NAND gate at the left with $C_{in} = 4$ and $g = 4/3$. We are going to express all capacitances in units of the input capacitance of the minimal inverter. All trasistor widths will be expressed as multiples of the width of the n channel transistor in a minimal inverter.

In a minimal NAND, widths of n and p channel transistors are 2 each, while the input capacitance is 4/3. To scale this up to $C_{in} = 4$, all n and p channel transistors should have widths which are 3 times the width of minimal NAND. Thus n as well as p channel transistors of these gates should have widths of 6.

For the 2 input NAND gates, $C_{in} = 4$ and $g = 4/3$. Since $\hat{f} = gh = 2.442$, $h = 2.442 \times 3/4 = 1.831$.

Since $h = C_{out}/C_{in}$, we get $C_{out} = 1.83 \times 4 = 7.326$.
The input capacitance of the NOR gate in the second stage should be the same as the output capacitance of the first stage = 7.326.

We now move to the second stage which uses 2 input NOR gates. A 2 input NOR gate with a unit sized n channel transistor will have input capacitance of 5/3 units. To get an input capacitance of 7.326, all transistors should be scaled up by a factor of $7.326 \times 3/5 = 4.4$. Thus the 2 input NOR should be sized such that the two parallel connected n channel transistors have a width of 4.4, while the two series connected p channel transistors should have widths of $4 \times 4.4 = 17.6$.
For the NOR stage, $\hat{f} = gh = 2.44$, $g = 5/3$ so $h = 2.44 \times 3/5 = 1.464$. Since $C_{in} = 7.326$. $C_{out} = 1.464 \times 7.326 = 10.73$.
This should be the input capacitance of the third stage, a 2 input NAND.

The input capacitance of a minimum NAND with 2 series connected n channel transistors of width 2 and two parallel connected pMOS transistors with width of 2 is 4/3. To make the input capacitance = 10.73, all transistors should be scaled up by a factor of $10.73 \times 3/4 = 8.05$. Thus the n as well as p channel transistors should have widths of 16.1.
For the third stage, $\hat{f} = gh = 2.442$, so $h = 2.442 \times 3/4 = 1.83$. Since $C_{in} = 10.73$, $C_{out} = 1.83 \times 10.73 = 19.65$. This should be the input capacitance of the last stage, an inverter with $g = 1$.

For the last stage, the input capacitance for a minimal inverter with n size of 1 and p size of 2 is 1. To bring it up to 19.65, all transistors should be scaled up by this factor. Thus the n width should be 19.65 and the p width should be 39.3

For the last stage, $g = 1$ and $\hat{f} = gh = h = 2.442$. Since $C_{in} = 19.65$, $C_{out} = 2.442 * 19.65 = 48$, which agrees with the specification and confirms our solution.

4.5 Optimizing the path length

Equation 4.33 requires that the number of stages in the logic path be known for optimizing the total delay. However, this may not be the optimum path length and we can some times get a faster circuit by buffering intermediate outputs in this logic path.

We assume that there is a logic path containing n_1 stages and we are free to add n_2 inverters to this path, if that results in a lower overall delay. We now consider the optimization of this logic path containing $N = n_1 + n_2$ stages. We shall assume that there is no requirement for n_2 to be even. (This implies that an inverted output is equally acceptable or else, the logic path and inputs can be suitably altered to produce the desired output). The optimization

problem is to find the scale factors for each of the $N = n_1 + n_2$ stages, such that the delay is minimum.

The path effort $F = GBH$ for the n_1 stages of logic is known. This is because the logical effort of each of the logic gates is known to us, so that G may be evaluated. Branching, if any, in the logic chain is also known, so B can be evaluated. finally, H depends only on the load capacitance and input capacitance, which is the starting specification for the optimization.

Addition of n_2 inverters does not change the value of G, since $g = 1$ for each of the n_2 inverters. Similarly, the inverters do not introduce any branching, so B remains the same. Finally, H is defined by the final load and the input capacitance of the first logic element, which is not changed by the inserted inverters. Therefore $F = GBH$ for the $N = n_1 + n_2$ stages (including n_2 inverters) is the same as the F for n_1 logic stages.

Additional inverters in the logic chain permit sharing the effort over a larger number of stages, which can reduce the total delay. We first find the optimum value of N. If $N > n_1$, we shall have the opportunity of reducing the delay by adding inverters.

The total delay in the N stages is the sum of delays of n_1 logic stages and n_2 inverters. For optimum delay, the stage effort should be equal for all the N stages. Therefore, the stage effort of logic stages as well as the inverters is the same and is $= F^{1/N}$. So the total delay is:

$$\hat{D} = NF^{1/N} + \sum_{i=1}^{n_1} p_i \ + (N - n_1)p_{inv} \tag{4.40}$$

The first term in Equation 4.40 above is the effort delay of N stages. The sum of parasitic delays of n_1 logic gates gives us the second term. Finally, the parasitic delay of $n_2 = N - n_1$ inverters gives the third term.

We define the optimum stage effort $\rho \equiv F^{1/N}$. Then

$$\hat{D} = N(\rho + p_{inv}) + \sum_{i=1}^{n_1} p_i \ - n_1 p_{inv} \tag{4.41}$$

Since $\rho \equiv F^{1/N}$, $N = \ln F / \ln \rho$. Therefore,

$$\hat{D} = \frac{\ln F}{\ln \rho}(\rho + p_{inv}) + \sum_{i=1}^{n_1} p_i \ - n_1 p_{inv} \tag{4.42}$$

It is to be noticed that F, n_1, p_i and p_{inv} are given constants. We now optimize the total delay with respect to ρ by setting the derivative of $\hat{D}$ with respect to ρ to zero. Differentiating by parts, we get

$$\frac{\partial \hat{D}}{\partial \rho} = 0 = -\frac{\ln F}{(\ln \rho)^2} \cdot \frac{1}{\rho} \cdot (\rho + p_{inv}) + \frac{\ln F}{\ln \rho}(1) \tag{4.43}$$

Therefore,

$$\frac{\ln F}{\ln \rho} = \frac{\ln F}{(\ln \rho)^2} \cdot \frac{1}{\rho} \cdot (\rho + p_{inv}) \tag{4.44}$$

and so,

$$1 = \frac{1}{\rho \ln \rho}(\rho + p_{inv}) \tag{4.45}$$

Which gives

$$\rho + p_{inv} = \rho \ln \rho \tag{4.46}$$

Which can be written as

$$p_{inv} + \rho(1 - \ln \rho) = 0 \tag{4.47}$$

It is interesting to note that this condition is independent of F and the value of ρ is uniquely defined by p_{inv}.

Equation 4.47 cannot be solved in closed form and either iterative solutions or graphical solutions have to be used to determine ρ from p_{inv}. In the special case when $p_{inv} = 0$, we have

$$\rho(1 - \ln \rho) = 0 \quad \text{so } \ln \rho = 1 \quad \text{which gives } \rho = e$$

This corresponds to the case of tapered inverter that we had solved before.

For non-zero values of p_{inv}, Equation 4.47 can be solved iteratively using the Newton Raphson technique. We can write Equation 4.47 as

$$f(\rho) \equiv \rho - \rho \ln \rho + p_{inv} = 0 \tag{4.48}$$

$$\text{Then} \quad f'(\rho) = 1 - \ln \rho - \rho \frac{1}{\rho} = -\ln \rho \tag{4.49}$$

If we have a guess solution g for this equation, the next improved guess according to Newton Raphson technique is:

$$g_{next} = g - \frac{f(g)}{f'(g)} = g + \frac{g - g \ln g + p_{inv}}{\ln g} = \frac{g + p_{inv}}{\ln g} \tag{4.50}$$

We know that for $p_{inv} = 0$, the value of ρ is e. A good guess value to start iterations will be $\rho = 3$. Let us illustrate the iterative method by taking $p_{inv} = 1$. The successive values obtained from Equation 4.50, starting with $\rho = 3$ are: 3.0, 3.6410, 3.5914, 3.5911, 3.5911,... The value converges to 4 decimal places within 3 to 4 iterations.

Similarly, for $p_{inv} = 0.6$, the successive solutions starting from 3 are: 3.2769, 3.2664, 3.2664 ...

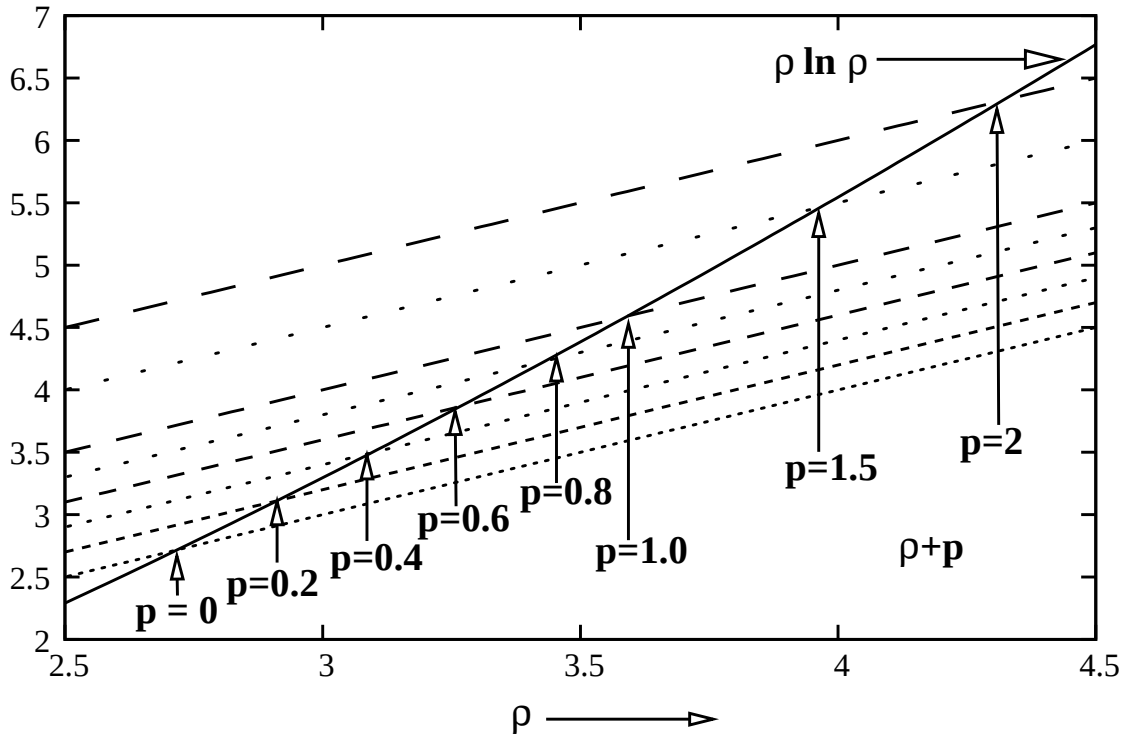


Figure 4.6: Graphical solution of the equation $\rho + p_{inv} = \rho \ln \rho$.

We can also solve Equation 4.46 graphically by plotting $\rho + p_{inv}$ and $\rho \ln \rho$ as functions of ρ. The value of ρ at which these two intersect is the solution of the equation.

Either way, one can evaluate ρ if the parasitic delay of an inverter is given. Table 4.2 below gives the values of ρ and the corresponding stage delay for several values of p_{inv}.

p_{inv}	ρ	$\ln \rho$	$d = \rho + p_{inv}$
0	2.718 (e)	1.000	2.718
0.2	2.912	1.069	3.11
0.4	3.093	1.129	3.49
0.6	3.266	1.184	3.87
0.8	3.432	1.233	4.23
1.0	3.591	1.278	4.59
1.5	3.967	1.378	5.47
2.0	4.319	1.463	6.32

Table 4.2: ρ as a function of p_{inv}. The table also gives the optimum delay.

Once ρ is known, we can evaluate the optimum number of logic stages for a given path effort by $N = \ln F / \ln \rho$. This value will be fractional in general and needs to be zapped to the nearest integer. After this rounding off, the stage effort has to be re-calculated as $f = F^(1/N)$. If $N > n_1$ (where n_1 is the number of logic stages which are necessary for implementing the desired logic function), we can insert $N - n_1$ inverter stages to optimize the overall delay.

Thus, $\hat{f}$ is the optimum stage effort when the number of logic stages is fixed and known. In this case the stage effort is F^{1/n_1} and the total delay is $n_1 F^{1/n_1} + \sum_{i=1}^{n_1} p_i$ where n_1 is the known number of logic stages.

If we have the freedom of inserting a number of inverters in the logic path to optimize delay, we follow the following procedure:

1. From p_{inv}, find the ideal stage effort ρ by solving $p_{inv} + \rho(1 - \ln \rho) = 0$. This can be done through iterative solutions or graphically as described earlier.

2. Once ρ is known, find the number of stages as $\ln F / \ln \rho$. The nearest integer to the value so found is the optimum number of stages N.

3. If $N > n_1$, insert $(N - n_1)$ inverters anywhere in the logic path. If $N \leq n_1$, take $N = n_1$.

4. The stage effort is now adjusted to $f = F^{1/N}$.

5. Given this value of f, we can start from the last stage and work backwards as earlier to calculate all transistor geometries.

6. For the last stage the output capacitance is known ($=C_L$). The input capacitance can be calculated from Equation 4.35.

$$C_{in_N} = b_N g_N \frac{C_L}{f}$$

   This gives the scale factor for this stage from which, geometries of transistors in the last stage can be computed.

7. For each preceding stage, we use the recursive relation 4.36

$$C_{in_i} = g_i \frac{b_i C_{in_{i+1}}}{f}$$

8. From C_{in_i}, we can calculate the scale factor, and hence the geometry of all transistors for this stage.

Path effort F	Optimum No. of stages $\hat{N}$	Min. Delay $\hat{D}$	range of values of Stage effort f
0-5.83	1	1.0-6.8	0-5.8
5.83-22.3	2	6.8-11.4	2.4-4.7
22.3-82.2	3	11.4-16.0	2.8-4.4
82.2-300	4	16.0-20.7	3.0-4.2
300-1090	5	20.7-25.3	3.1-4.1
1090-3920	6	25.3-29.8	3.2-4.0
3920-14200	7	29.8-34.4	3.3-4.0
14200-51000	8	34.4-39.0	3.3-3.9
51000-184000	9	39.0-43.6	3.3-3.9
184000-661000	10	43.6-48.2	3.4-3.8

Table 4.3: Optimum number of stages for different F values. This table assumes $p_{inv} = 1$, so $\rho = 3.59$.

The optimum number of stages will depend on $F = GBH$. Because N must be an integer, a range of values of F will require the same number of stages. Table 4.5 gives the optimum number of stages for ranges of F values. This table assumes the value of p_{inv} to be 1.0. 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 shown in Table 4.4, is that delay is quite insensitive to the number of stages, provided the deviation from optimum is not too large. As the table

$N/\hat{N}$	$D/\hat{D}$
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 4.4: 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}, $= 0.6$.

shows, doubling the number of stages from optimum increases the delay only 24%. Using half as many stages as the optimum increases the delay by 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.

4.6 Examples

(Examples in this section are taken from the book by Sutherland, Sproull and Harris).

4.6.1 A cache comparator

As an 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 diffcult 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 4.7. This figure is somewhat schematic; it

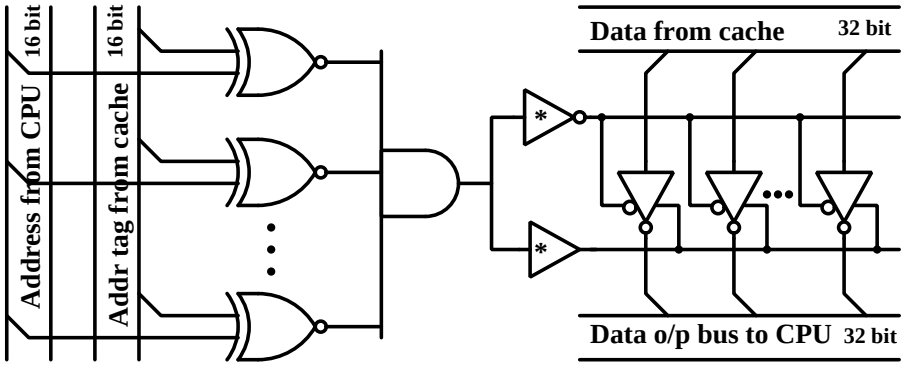


Figure 4.7: Block diagram of a cache comparator.

shows only a few of the 16 XNOR OR 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.

Figure 4.8 shows an XNOR circuit implemented with 3 NAND gates.

$$\overline{(A+B)\cdot(\overline{A}+\overline{B})} = \overline{A\cdot\overline{B}+\overline{A}\cdot B} = \overline{A\oplus B}$$

The logical effort of the XNOR gate so implemented by three NAND gates is $4/3\times4/3 = 16/9$.

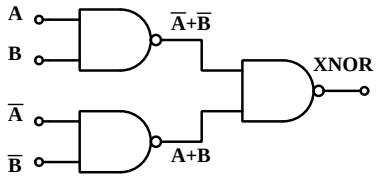


Figure 4.8: XNOR implemented with 3 NAND gates

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 figure 4.7 is $2\times32+4\times32 = 192$ units, then H = 24. Thus the path effort is F = GBH = 210.7. For p_{inv} = 0.6, the theoretical optimum ρ is 3.2664. Therefore the number of stages to be used is $\ln 210.7/\ln 3.2664 = 4,52$. Thus, 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.

4.6.2 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 4.9. One implementation of such an adder circuit uses single-stage parity and majority circuits,

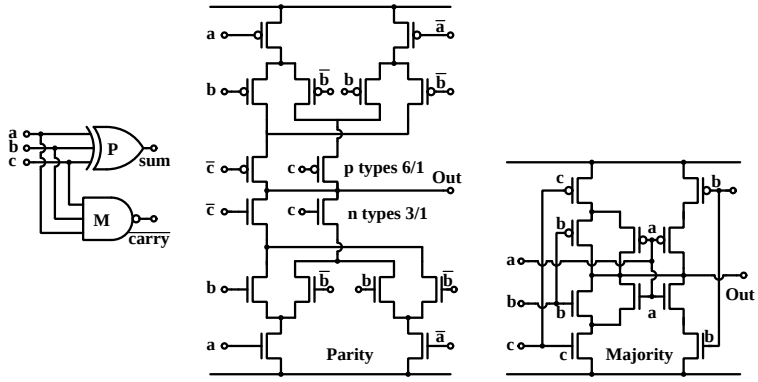


Figure 4.9: Parity and majority circuits for building an array multiplier.

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 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 (9 = 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.

4.7 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 data paths.

- 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 4.5.

- 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 T 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 4.1), 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 including pre-charged 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.

4.8 Related work

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 This model is less accurate than those that include terms to account for differences in rise times of input signals. 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.

4.9 Conclusions

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 are 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 what 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, 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 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 tri—state drivers. Logical effort unifies all of these design problems.

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-192159—161, 1984.

[7] J. Shyu, J. P. Fishburn, A. E. Dunlop, and A. L. Sangiovanni-Vincentelli. Optimization-Based Transistor Sizing. IEEE 1.987 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 Efiort: Designing Fast MOS Circuits. Monograph in preparation.