(a) Calculate the process transconductance k'_p in units of mA/V2.

(b) Find the device transconductance β_p and the resistance R_p.

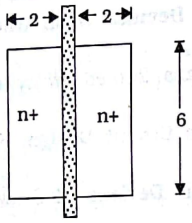
[6.5] An nFET has a gate oxide with a thickness of $t_{ox} = 120$ Å. The p-type bulk region is doped with boron at a density of $N_a = 8 \times 10^{14}$ cm^{-3}. Given that $V_{T0n} = 0.55$ V and $(W/L) = 10$.

(a) Calculate the body bias coefficient γ.

(b) What is the device threshold voltage if a body bias voltage of V_{SBn} = 2 V is applied?

(c) The electron mobility is $\mu_n = 540$ cm^2/V-sec. Calculate the drain current with bias voltages of $V_{GSn} = 3$ V, $V_{DSn} = 3$ V, and $V_{SBn} = 3$ V applied to the device.

[6.6] Construct the RC switch model for the FET layout in Figure P6.1. Assume a power supply voltage of 3 V and that the dimensions are in units of microns.



$L' = 0.5$ µm
$L_o = 0.05$ µm
$V_{T0n} = 0.6$ V

$k'_n = 150$ µA/V^2
$C_{ox} = 2.70$ fF/µm^2
$C_j = 0.86$ fF/µm^2
$C_{jsw} = 0.24$ fF/µm

Figure P6.1 Transistor layout geometry for Problem 6.6

[6.7] Write a SPICE description of the nFET in Figure P6.1. Use your listing to obtain the family of I_D versus V_{DS} curves.

[6.8] Consider the FET geometry shown in Figure P6.1 where the sheet resistance of the n+ regions is 30 Ω, and the poly gate has a sheet resistance of 26 Ω. Compute the parasitic resistances R_{n+} and R_{poly} associated with these parameters by determining the appropriate geometry that applies for each. How would these parasitics affect the device operation?

[6.9] An nFET with $W = 20$ µm and $L = 0.5$ µm is built in a process where $k'_n = 120$ µA/V^2 and $V_{Tn} = 0.65$ V. The voltages are set to a value of $V_{GSn} = V_{DSn} = V_{DD} = 5$ V.

(a) Is the transistor saturated or non-saturated?

(b) Calculate the drain-source resistance using the proper equation for the transistor.

(c) Compare your value in (b) with that found using equation (6.71) with a value of $\eta = 1$.

[6.10] An nFET with $L = 0.5$ µm is built in a process where $k'_n = 100$ µA/V^2 and $V_{Tn} = 0.70$ V. The gate-source voltage is set to a value of $V_{GSn} = V_{DD} = 3.3$ V. Calculate the required channel width to obtain a resistance of $R_n = 950$ Ω using equation (6.71) with for a value of $\eta = 1$.

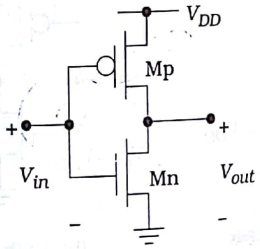
Electronic Analysis of CMOS Logic Gates

7

In the previous chapter we examined the electrical characteristics of MOSFETs. This sets the foundation for analyzing the behavior of transistors in CMOS logic circuits in this chapter. The treatment centers on the important areas of switching speed and layout design, and provides the foundation for much of modern chip design.

7.1 DC Characteristics of the CMOS Inverter

The CMOS inverter gives the basis for calculating the electrical characteristics of logic gates. Consider the circuit shown in Figure 7.1. The input voltage V_{in} determines the conduction states of the two FETs Mn and Mp. This produces the output voltage V_{out} of the gate. Two types of calculations are needed to characterize a digital logic circuit. A **DC analysis** determines V_{out} for a given value of V_{in}. In this type of calculation, it is assumed that V_{in} is changed very slowly, and that V_{out} is allowed to stabilize before a measurement is made. A DC analysis provides a direct mapping of the input to the output, which in turn tells us the voltage ranges



pFET: $V_{Tp} < 0$
$\beta_p = k'_p \left(\frac{W}{L}\right)_p$

nFET: $V_{Tn} > 0$
$\beta_n = k'_n \left(\frac{W}{L}\right)_n$

Figure 7.1 The CMOS inverter circuit

that define Boolean logic 0 and logic 1 values. The second type of characterization is called a **transient analysis**. In this case, the input voltage is an explicit function of time $V_{in}(t)$ corresponding to a changing logic value. The response of the circuit is contained in $V_{out}(t)$. The delay between a change in the input and the corresponding change at the output is the fundamental limiting factor for high-speed design. In this section we will concentrate on the DC analysis. The transient response is analyzed in the next section.

The DC characteristics of the inverter are portrayed in the **voltage transfer characteristic (VTC)**, which is a plot of V_{out} as a function of V_{in}. This is obtained by varying the input voltage V_{in} in the range from 0 V to V_{DD} and finding the output voltage V_{out}. The end point values are easily found with the aid of the circuits in Figure 7.2. If V_{in} is equal to 0 V as in Figure 7.2(a), Mn is off while Mp is on. Since the pFET is on, it connects the output to the power supply and gives $V_{out} = V_{DD}$. This defines the **output high voltage** of the circuit as

$$V_{OH} = V_{DD} \tag{7.1}$$

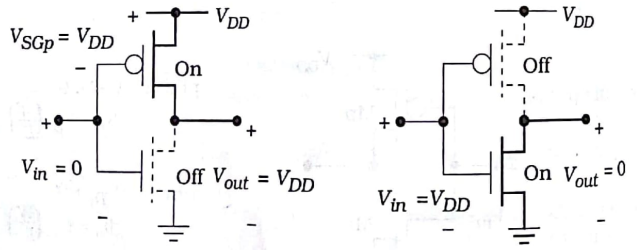
i.e., the highest output voltage is the value of the power supply V_{DD}. The opposite case with $V_{in} = V_{DD}$ is illustrated in Figure 7.2(b). This turns on Mn while Mp is in cutoff. The output node is then connected to 0 V (ground) through the nFET, defining the **output low voltage**

$$V_{OL} = 0 \text{ V} \tag{7.2}$$

The **logic swing** at the output is

$$V_L = V_{OH} - V_{OL} = V_{DD} \tag{7.3}$$

Since this is equal to the full value of the power supply, this is called a **full-rail output**.



(a) Low input voltage (b) High input voltage

Figure 7.2 V_{OH} and V_{OL} for the inverter circuit

The VTC for the circuit is obtained by starting with an input voltage of $V_{in} = 0$ V and then increasing it up to a value of $V_{in} = V_{DD}$. This results in the plot shown in Figure 7.3. The details can be understood by writing the device voltages in terms of the input and output voltages:

$$V_{GSn} = V_{in}$$
$$V_{SGp} = V_{DD} - V_{in} \tag{7.4}$$

Mn is in cutoff so long as $V_{in} \le V_{Tn}$. Since the output voltage is high with a value $V_{out} = V_{DD}$, any input voltage in the range labeled as "0" can be interpreted as a logic 0 input. Increasing V_{in} causes a downward transition in the VTC. This is because the input voltage turns the nFET on while the pFET is still conducting. Note, however, that increasing V_{in} decreases V_{SGp}, so the pFET becomes a less efficient conductor and the output voltage falls. Mp goes into cutoff when

$$V_{in} = V_{DD} - |V_{Tp}| \tag{7.5}$$

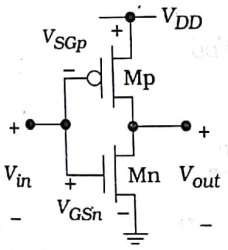
For V_{in} greater than this value, $V_{out} = 0$ V since only the nFET is active. This shows that there is a range of input voltages that act as logic 1 input values as indicated by the "1" on the VTC.

The logic 0 and 1 voltage ranges are defined by the changing slope of the VTC. Point 'a' in the drawing is where the slope has a value of -1, and defines the **input low voltage** V_{IL}. By definition, a logic 0 input voltage is defined by

$$0 \le V_{in} \le V_{IL} \tag{7.6}$$

83448

The second -1 slope point is labeled as 'b' and defines the **input high voltage** V_{IH}. This is used to define a logic 1 input voltage as



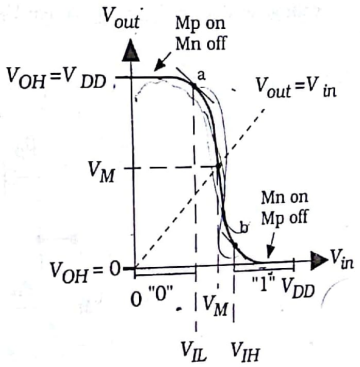


Figure 7.3 Voltage transfer curve for the NOT gate

$$V_{IH} \le V_{in} \le V_{DD}$$

The **voltage noise margins** are

$$VNM_H = V_{OH} - V_{IH} \quad (7.)$$

$$VNM_L = V_{IL} - V_{OL} \quad (7.)$$

for high and low states, respectively. The noise margins give a quantitative measure of how stable the inputs are with respect to coupled electromagnetic signal interference.

While it is possible to calculate the exact voltages that define logic 0 and 1 input voltages, it is simpler to introduce the midpoint voltage V_M shown in the VTC. This is defined as the point where the VTC intersects the unity gain line that is defined by $V_{out} = V_{in} = V_M$. A value of $V_{in} = V_M$ is in the transition region and does not represent a Boolean quantity. However, for V_{in} less than V_M the input voltage is toward the logic 0 value while $V_{in} > V_M$ indicates that the input is on the logic 1 side. Knowing the value of V_M thus tells us the center point for input transitions.

To calculate the midpoint voltage we set $V_{in} = V_{out} = V_M$ as shown in Figure 7.4. Equating the drain currents of the FETs gives

$$I_{Dn} = I_{Dp} \quad (7.9)$$

but we need to find the operating region (saturation or non-saturation) of each FET before we can use the expression. Consider first the nFET and recall that the saturation voltage is given by

$$V_{sat} = V_{GSn} - V_{Tn} = V_M - V_{Tn} \quad (7.10)$$

where we have used $V_{in} = V_{GSn} = V_M$ in the second line. The drain-source voltage is $V_{DSn} = V_{out} = V_M$. Since V_{Tn} is a positive number,

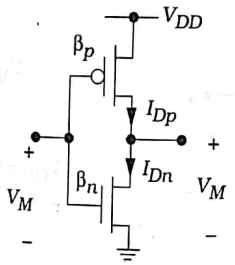


Figure 7.4 Inverter voltages for V_M calculation

$$V_{DSn} > V_{sat} = V_M - V_{Tn} \quad (7.11)$$

which says that Mn must be saturated. The same arguments can be applied to the pFET Mp since $V_{SGp} = V_{SDp}$. Using the saturation current equations from Chapter 6 gives

$$\frac{\beta_n}{2}(V_M - V_{Tn})^2 = \frac{\beta_p}{2}(V_{DD} - V_M - |V_{Tp}|)^2 \quad (7.12)$$

Dividing by β_p and taking the square root gives

$$\sqrt{\frac{\beta_n}{\beta_p}}(V_M - V_{Tn}) = V_{DD} - V_M - |V_{Tp}| \quad (7.13)$$

Simple algebra then gives the midpoint voltage as

$$V_M = \frac{V_{DD} - |V_{Tp}| + \sqrt{\frac{\beta_n}{\beta_p}} V_{Tn}}{1 + \sqrt{\frac{\beta_n}{\beta_p}}} \quad (7.14)$$

This equation shows that V_M is set by the nFET-to-pFET ratio

$$\frac{\beta_n}{\beta_p} = \frac{k'_n \left(\frac{W}{L}\right)_n}{k'_p \left(\frac{W}{L}\right)_p} \quad (7.15)$$

Since k'_n and k'_p are set in the processing, the ratio of the FET sizes establishes the switching point. It is important to remember that nFETs and pFETs have different mobility factors with a typical ratio of

$$\frac{k'_n}{k'_p} \approx 2 \text{ to } 3 \quad (7.16)$$

depending upon the details of the processing. This fact has a significant effect on the choices we make in both the sizing of individual transistors, and the types of circuits that are used in advanced VLSI designs. Note that, since C_{ox} is approximately the same for both FET types,

$$\frac{k'_n}{k'_p} = \frac{\mu_n}{\mu_p} = r \quad (7.17)$$

where r is the mobility ratio introduced in Chapter 5.

A **symmetrical inverter** VTC is one that has equal "0" and "1" input voltage ranges. This can be achieved by choosing

$$V_M = \frac{1}{2}V_{DD}$$

in equation (7.12). Rearranging gives us the design equation

$$\frac{\beta_n}{\beta_p} = \left(\frac{\frac{1}{2}V_{DD} - |V_{Tp}|}{\frac{1}{2}V_{DD} - V_{Tn}}\right)^2$$

This allows us to compute the transistor sizes for this particular choice of V_M. Note that if $V_{Tn} = |V_{Tp}|$, then a symmetric design requires that

$$\beta_n = \beta_p$$

i.e., the device transconductance values of the two FETs are equal. It is important to remember that β is proportional to the aspect ratio (W/L) of a MOSFET, and that (W/L) is the actual design variable.

Example 7.1

Consider a CMOS process with the following parameters

$$k'_n = 140\ \mu A/V^2 \qquad V_{Tn} = +0.70\ V$$
$$k'_p = 60\ \mu A/V^2 \qquad V_{Tp} = -0.70\ V$$

with $V_{DD} = 3.0$ V.

Consider the case where $\beta_n = \beta_p$. We can verify that this is a symmetrical design by calculating

$$V_M = \frac{3 - 0.7 + \sqrt{1}(0.7)}{1 + \sqrt{1}} = 1.5\ V$$

so that V_M is one-half the value of the power supply voltage. To achieve this design, we must choose the device aspect ratios such that

$$\frac{\beta_n}{\beta_p} = \frac{k'_n\left(\frac{W}{L}\right)_n}{k'_p\left(\frac{W}{L}\right)_p} = 1$$

where we recall that the process transconductance parameters k' are given by $k' = \mu_n C_{ox}$, and are set by the processing. For the present case we rearrange the expression to read

$$\left(\frac{W}{L}\right)_p = \frac{k'_n}{k'_p}\left(\frac{W}{L}\right)_n$$

so that

$$\left(\frac{W}{L}\right)_p = \left(\frac{140}{60}\right)\left(\frac{W}{L}\right)_n = 2.33\left(\frac{W}{L}\right)_n \qquad (7.25)$$

This shows that the pFET must be about 2.33 times larger than the nFET.

Let us now examine the case where the nFET and the pFET have the same aspect ratio: $(W/L)_n = (W/L)_p$. With the values provided in the problem statement,

$$\frac{\beta_n}{\beta_p} = \frac{k'_n}{k'_p} = 2.33 \qquad (7.26)$$

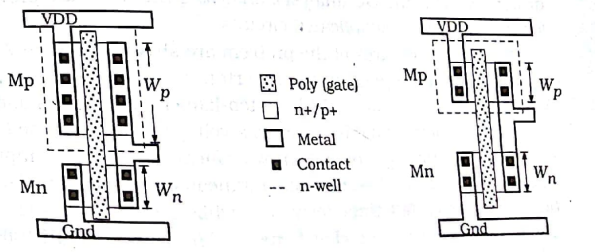
so that the midpoint voltage is given by

$$V_M = \frac{3 - 0.7 + \sqrt{2.33}\ (0.7)}{1 + \sqrt{2.33}} = 1.33\ V \qquad (7.27)$$

This choice shifts V_M to a value that is smaller than $(V_{DD}/2)$.

Figure 7.5 illustrates the difference in the layout between an inverter that uses the two design styles. The channel length is the same for both transistors in the inverter, leaving the channel widths W_p and W_n as the design variables. In Figure 7.5(a), the pFET has a width of about $W_p \approx 2$ W_n which gives V_M of about $(V_{DD}/2)$. Equal size transistors are used in the layout of Figure 7.5(b), so that the circuit has $V_M < (V_{DD}/2)$. It is important to remember that we are only dealing with the DC characteristics at the moment. As we will see in the next section, the switching properties of the two designs are also affected by the aspect ratios.

The derivation and examples above illustrate the importance of the FET aspect ratios in the DC behavior of the logic gate. At the physical level, the relative device sizes contained in the ratio (β_n/β_p) determine the switching points. In general, increasing (β_n/β_p) decreases the value of the midpoint voltage V_M. This dependence is illustrated in the plot of Figure



(a) Larger pFET design (b) Equal aspect ratios

Figure 7.5 Comparison of the layouts for Example 7.1

7.6. With the parameters shown, a symmetrical design with $\beta_n = \beta_p$ gives $V_M = (V_{DD}/2) = 1.5$ V. Increasing the ratio to $(\beta_n/\beta_p) = 1.5$ gives $V_M \approx 1.$ V, while $(\beta_n/\beta_p) = 2.5$ decreases the midpoint voltage to $V_M \approx 1.31$ V. It is also possible to use a ratio of $(\beta_n/\beta_p) < 1$, which shifts the VTC toward the right, i.e., $V_M > (V_{DD}/2)$. However, this is rarely used since the pFET aspect ratios get quite large.

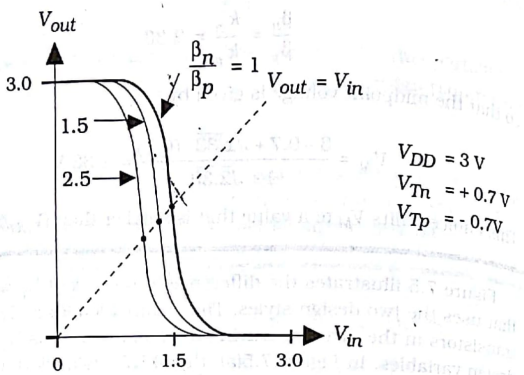


Figure 7.6 Dependence of V_M on the device ratio

7.2 Inverter Switching Characteristics

High-speed digital system design is based on the ability to perform calculations very quickly. This requires that logic gates introduce a minimum amount of time delay when the inputs change. Designing fast logic circuits is one of the more challenging (but critical) aspects of VLSI physical design. As with the DC analysis, analyzing the NOT gate provides a basis for studying more complicated circuits.

The general features of the problem are shown in Figure 7.7. An input voltage $V_{in}(t)$ is applied to the inverter, resulting in an output voltage $V_{out}(t)$. We assume that $V_{in}(t)$ has step-like characteristics and makes an abrupt transition from 0 to 1 (i.e., to a voltage of V_{DD}) at time t_1, and back down to 0 at time t_2. The output waveform reacts to the input, but the output voltage cannot change instantaneously. The output 1-to-0 transition introduces a **fall time** delay of t_f, while the 0-to-1 change at the output is described by the **rise time** t_r. The rise and fall times can be calculated by analyzing the electronic transitions of the circuits.

The rise and fall time delays are due to the parasitic resistance and

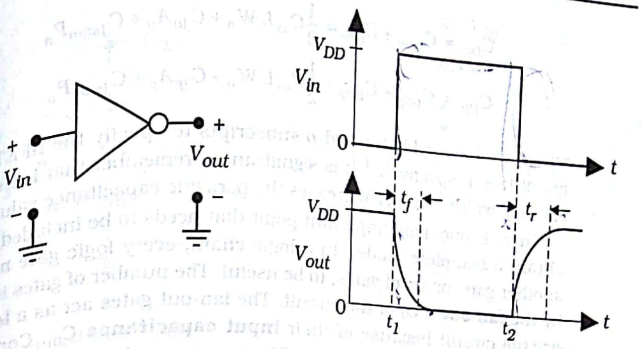
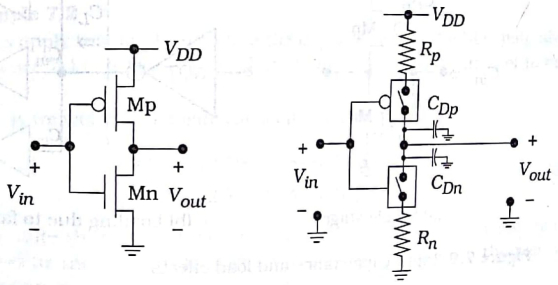


Figure 7.7 General switching waveforms

capacitances of the transistors. Consider the NOT circuit shown in Figure 7.8(a). Both FETs can be replaced by their switch equivalents, which results in the simplified RC model in Figure 7.8(b). It is worth recalling that the actual values of the components depend upon the device dimensions. Once we specify the aspect ratios $(W/L)_n$ and $(W/L)_p$, we can calculate R_n and R_p using

$$R_n = \frac{1}{\beta_n(V_{DD} - V_{Tn})}$$
$$R_p = \frac{1}{\beta_p(V_{DD} - |V_{Tp}|)} \qquad (7.28)$$

Knowing the layout dimensions of each FET allows us to find the capacitances C_{Dn} and C_{Dp} at the output node. The formulas are given by



(a) FET circuit (b) RC switch model equivalent

Figure 7.8 RC switch model equivalent for the CMOS inverter

$$C_{Dn} = C_{GSn} + C_{DBn} = \frac{1}{2}C_{ox}L'W_n + C_{Jn}A_n + C_{Jswn}P_n$$

$$C_{Dp} = C_{GSp} + C_{DBp} = \frac{1}{2}C_{ox}L'W_p + C_{Jp}A_p + C_{Jswp}P_p \quad (7.29)$$

where we have added n and p subscripts to specify the nFET or pFET quantities, respectively.[1] It is significant to remember that increasing the channel width of a FET increases the parasitic capacitance values.

There is one more important point that needs to be included before we obtain a complete model. In a logic chain, every logic gate must drive another gate, or set of gates, to be useful. The number of gates is specified by the **fan-out** (FO) of the circuit. The fan-out gates act as a **load** to the driving circuit because of their **input capacitance** C_{in}. Consider the inverter shown in Figure 7.9(a). The input capacitance of the inverter is just the sum of the FET capacitances

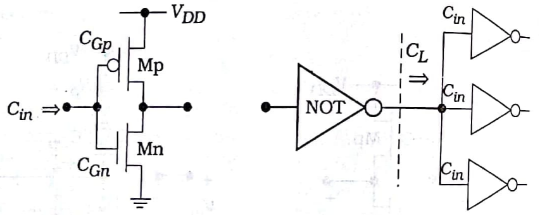
$$C_{in} = C_{Gp} + C_{Gn} \quad (7.30)$$

Figure 7.8(b) shows the effect of input capacitance for a fan-out of FO = 3. The input capacitance to each gate acts as an **external load capacitance** C_L to the driving gate. In this example, it is easily seen that

$$C_L = 3C_{in} \quad (7.31)$$

is the value of the load presented to the NOT gate.

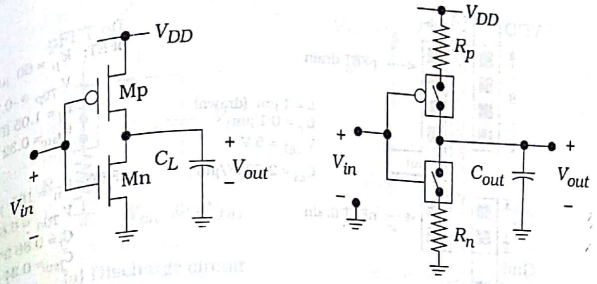
We may now calculate the switching times of the inverter. Figure 7.10 illustrates the general problem. A CMOS NOT gate is used to drive an external load capacitance C_L as in Figure 7.10(a). This gives the complete



(a) Single stage (b) Loading due to fan-out

Figure 7.9 Input capacitance and load effects

[1] Note that the source capacitances C_{Sp} and C_{Sn} do not enter the problem as they are at the power supply and ground, respectively, and have constant voltages.



(a) External load (b) Complete switching model

Figure 7.10 Evolution of the inverter switching model

switching model shown in Figure 7.10(b) where the total output capacitance is defined as

$$C_{out} = C_{FET} + C_L \quad (7.32)$$

The FET capacitances shown earlier in Figure 7.8 have been merged into the single term

$$C_{FET} = C_{Dn} + C_{Dp} \quad (7.33)$$

and are the parasitic internal contributions that cannot be eliminated. These add with C_L since all elements are in parallel. The total output capacitance C_{out} is the load that the gate must drive; the numerical value varies with the load.

Example 7.2

Let us apply this analysis to find the capacitances in the NOT gate shown in Figure 7.11. It is assumed that all dimensions have units of microns (μm).

First we will find the gate capacitances using

$$C_{Gp} = (2.70)(1)(8) = 21.6 \text{ fF}$$
$$C_{Gn} = (2.70)(1)(4) = 10.8 \text{ fF} \quad (7.34)$$

Next, note that the overlap distance L_o is specified as 0.1 μm, which should be included in the area and perimeter factors in the junction capacitances. For the pFET, the p+ capacitance is

$$C_p = C_j A_{bot} + C_{jsw} P_{sw} \quad (7.35)$$

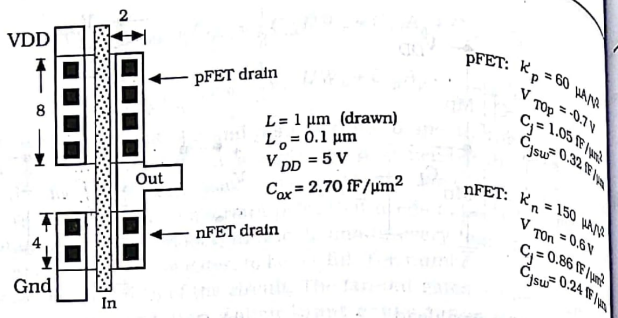


Figure 7.11 Example of capacitance calculations

so

$$C_p = (1.05)(8)(2.1) + (0.32)2(8 + 2.1) = 24.10 \text{ fF} \quad (7.3$$

The total capacitance at the pFET drain is therefore given by

$$C_{Dp} = \frac{21.6}{2} + 24.10 = 34.9 \text{ fF} \quad (7.3$$

The nFET drain is analyzed using the same approach. The n+ junctio capacitance is

$$C_n = (0.86)(4)(2.1) + (0.24)(2)(4 + 2.1) = 10.15 \text{ fF} \quad (7.3$$

so that

$$C_{Dn} = \frac{10.8}{2} + 10.15 = 15.55 \text{ fF} \quad (7.3$$

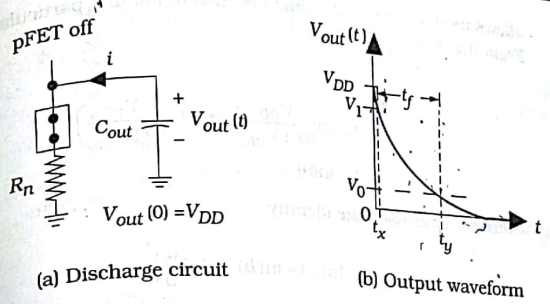
is the total capacitance at the drain of the nFET. Adding gives

$$\begin{aligned} C_{FET} &= C_{Dp} + C_{Dn} \\ &= 34.9 + 15.55 \\ &= 50.45 \text{ fF} \end{aligned} \quad (7.40$$

as the total internal FET capacitance. The total capacitance at the output is

$$C_{out} = 50.45 + C_L \quad (7.41$$

in fF, where C_L is the external load (also in fF).



(a) Discharge circuit (b) Output waveform

Figure 7.12 Discharge circuit for the fall time calculation

7.2.1 Fall Time Calculation

Let us start by calculating the output fall time t_f. We will shift the time origin such that V_{in} changes from 0 to V_{DD} at time $t = 0$. The initial condition at the output is $V_{out}(0) = V_{DD}$. When the input is switched, the nFET goes active while the pFET is driven into cutoff. In terms of the switch models, the nFET switch is closed and the pFET switch is open. This gives us the simplified discharge circuit shown in Figure 7.12(a). The capacitor C_{out} is initially charged to a voltage V_{DD}, and is allowed to discharge to 0 V through the nFET resistance R_n. The current leaving the capacitor is

$$i = -C_{out}\frac{dV_{out}}{dt} = \frac{V_{out}}{R_n} \quad (7.42)$$

which gives the differential equation for the discharge event. Solving with the initial condition $V_{out}(0) = V_{DD}$ results in the well-known form

$$V_{out}(t) = V_{DD}e^{-t/\tau_n} \quad (7.43)$$

where

$$\tau_n = R_n C_{out} \quad (7.44)$$

is the nFET **time constant** with units of seconds. The function is plotted in Figure 7.12(b).

The fall time is traditionally defined to be the time interval from $V_1 = 0.9\ V_{DD}$ to $V_0 = 0.1\ V_{DD}$, which are respectively known as the 90% and the 10% voltages as referenced to the full rail swing of V_{DD}. Rearranging the solution to the form

$$t = \tau_n \ln\left(\frac{V_{DD}}{V_{out}}\right) \quad (7.45)$$

allows us to calculate the time t needed to fall to a particular voltage V_y. From the drawing we see that

$$\begin{aligned} t_f &= t_y - t_x \\ &= \tau_n \ln\left(\frac{V_{DD}}{0.1V_{DD}}\right) - \tau_n \ln\left(\frac{V_{DD}}{0.9V_{DD}}\right) \\ &= \tau_n \ln(9) \end{aligned} \tag{7.4}$$

where we have used the identity

$$\ln(a) - \ln(b) = \ln\left(\frac{a}{b}\right) \tag{7.5}$$

in the last step. Approximating $\ln(9) \approx 2.2$ gives the final result

$$t_f \approx 2.2\tau_n \tag{7.6}$$

as the fall time for the circuit. The output fall time in a generic digital logic gate is usually called the output **high-to-low time** t_{HL} and is identical to the value computed here:

$$t_{HL} = t_f \tag{7.7}$$

The two symbols will be used interchangeably in the discussion.

7.2.2 The Rise Time

The rise time calculation follows in the same manner. Initially, the input voltage is at $V_{in} = V_{DD}$ and is switched to $V_{in} = 0$ V; we time shift this event to occur at $t = 0$ for simplicity. This turns on the pFET while simultaneously driving the nFET into cutoff, so that the simplified charging circuit of Figure 7.13(a) is valid. The output voltage at $t = 0$ is given by $V_{out}(0) = 0$ V.

The charging current is given by

$$i = C_{out}\frac{dV_{out}}{dt} = \frac{V_{DD} - V_{out}}{R_p} \tag{7.50}$$

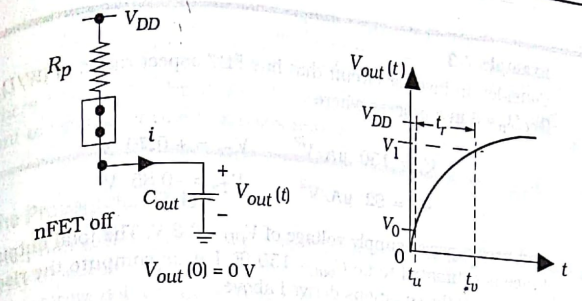
Solving and applying the initial condition gives the exponential form

$$V_{out}(t) = V_{DD}[1 - e^{-t/\tau_p}] \tag{7.51}$$

where the pFET time constant is defined by

$$\tau_p = R_p C_{out} \tag{7.52}$$

Figure 7.13(b) shows the output voltage as a function of time. The rise time is taken between 10% and 90% points such that



(a) Charge circuit (b) Output waveform

Figure 7.13 Rise time calculation

$$t_r = t_v - t_u \tag{7.53}$$

A little algebra yields the expression

$$t_r = \ln(9)\tau_p \approx 2.2\tau_p \tag{7.54}$$

for the rise time t_r. This has the same form as the fall time t_f because of the symmetry of the charge and discharge circuits. The rise time is identical to the output **low-to-high time** t_{LH}; the symbols will be used interchangeably.

The low-to-high time t_{LH} and the high-to-low time t_{HL} represent the shortest amount of time needed for the output to change from a logic 0 to logic 1 voltage, or from a logic 1 to a logic 0 voltage, respectively. Let us assume that the input is a square wave with a period of T sec such that the voltage is 0 for $(T/2)$ and V_{DD} for a $(T/2)$ time interval.[2] We then define the **maximum signal frequency** as

$$f_{max} = \frac{1}{t_{HL} + t_{LH}} = \frac{1}{t_r + t_f} \tag{7.55}$$

since this is the largest frequency that can be applied to the gate and still allow the output to settle to a definable state.[3] If the signal frequency exceeds f_{max}, the output voltage of the gate will not have sufficient time to stabilize to the correct value.

[2] This defines what is known as a 50% duty cycle.

[3] This definition assumes that t_{HL} and t_{LH} have the same order of magnitude to be useful.

Example 7.3

Consider an inverter circuit that has FET aspect ratios of $(W/L)_n = 6$ and $(W/L)_p = 8$ in a process where

$$k'_n = 150\ \mu A/V^2 \qquad V_{Tn} = +0.70\ V$$
$$k'_p = 62\ \mu A/V^2 \qquad V_{Tp} = -0.85\ V \tag{7.56}$$

and uses a power supply voltage of $V_{DD} = 3.3$ V. The total output capacitance is estimated to be $C_{out} = 150$ fF. Let us compute the rise and fall times using the equations derived above.

Consider first the fall time. The pFET resistance is given by

$$R_p = \frac{1}{\beta_p(V_{DD} - |V_{Tp}|)} = \frac{1}{(62\times10^{-6})(8)(3.3-0.85)} = 822.9\ \Omega \tag{7.57}$$

The time constant for the charging event is computed using the RC product R_pC_{out} to find

$$\tau_p = (822.9)(150\times10^{-15}) = 123.43 \quad ps \tag{7.58}$$

where 1 ps (picosecond) is 10^{-12} sec. The rise time is

$$t_r = 2.2\tau_p = 271.55 \quad ps \tag{7.59}$$

The fall time is calculated in a similar manner. First, we find the nFET resistance

$$R_n = \frac{1}{\beta_n(V_{DD} - V_{Tn})} = \frac{1}{(150\times10^{-6})(6)(3.3-0.70)} = 427.35\ \Omega \tag{7.60}$$

so that the discharge time constant is

$$\tau_p = (427.35)(150\times10^{-15}) = 64.1 \quad ps \tag{7.61}$$

The fall time is

$$t_f = 2.2\tau_n = 141.0 \quad ps \tag{7.62}$$

Combining these results, the maximum signal frequency is

$$f_{max} = \frac{1}{t_r + t_f} = \frac{1}{(271.55 + 141.0)\times10^{-12}} = 2.42 \quad GHz \tag{7.63}$$

where 1 GHz = 10^9 Hz. Although this is a very high frequency, it is important to remember that this refers only to a single inverter.

7.2.3 The Propagation Delay

The propagation delay time t_p is often used to estimate the "reaction" delay time from input to output. When we use step-like input voltages, the propagation delay is defined by the simple average of the two time intervals shown in Figure 7.14 by

$$t_p = \frac{(t_{pf} + t_{pr})}{2} \tag{7.64}$$

In this expression, t_{pf} is the output fall time from the maximum level to the "50%" voltage line, i.e., from V_{DD} to $(V_{DD}/2)$; t_{pr} is the propagation rise time from 0 V to $(V_{DD}/2)$. Using the exponential equations for V_{out} we obtain

$$t_{pf} = \ln(2)\tau_n$$
$$t_{pr} = \ln(2)\tau_p \tag{7.65}$$

Approximating $\ln(2) \approx 0.693$ then gives

$$t_p \approx 0.35(\tau_n + \tau_p) \tag{7.66}$$

The propagation delay time is a useful estimate of the basic delay, but does not provide detailed information on the rise and fall times as individual quantities. Propagation delays are commonly used in basic logic simulation programs.

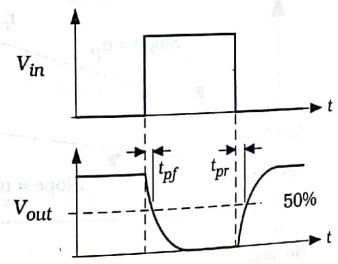


Figure 7.14 Propagation time definitions

7.2.4 General Analysis

The rise and fall time equations provide the basis for high-speed CMOS design. We can manipulate them to show us how to design single logic gates and then characterize the behavior of the gates when used in logic cascades.

To see the important factors, recall that the total output capacitance consists of two terms such that

$$C_{out} = C_{FET} + C_L \tag{7.67}$$

C_{FET} represents the parasitic capacitances of the transistors, while C_L is the external load. The layout geometry establishes the value of C_{FET}, but the load capacitance C_L varies with the application. Substituting this expression into the rise and fall time equations gives

$$t_r \approx 2.2R_p(C_{FET} + C_L)$$
$$t_f \approx 2.2R_n(C_{FET} + C_L) \tag{7.68}$$

which can be cast into the forms

$$t_r = t_{r0} + \alpha_p C_L$$
$$t_f = t_{f0} + \alpha_n C_L \tag{7.69}$$

These show that the rise and fall times are linear functions of the load capacitance C_L. The general behavior of both quantities is shown in Figure 7.15. Under zero-load conditions ($C_L = 0$), the inverter drives its own capacitances such that

$$t_r = t_{r0} \approx 2.2R_pC_{FET}$$
$$t_f = t_{f0} \approx 2.2R_nC_{FET} \tag{7.70}$$

are determined solely from the inverter parameters. When an external load C_L is added, the switching times increase in a linear fashion. Large capacitive loads may cause problems because of longer delays. The dependence is described by the slope values

$$\alpha_p = 2.2R_p = \frac{2.2}{\beta_p(V_{DD} - |V_{Tp}|)} \tag{7.71}$$

and

$$\alpha_n = 2.2R_n = \frac{2.2}{\beta_n(V_{DD} - V_{Tn})} \tag{7.72}$$

Note that these are inversely proportional to the aspect ratios since

$$\beta_p = k_p'\left(\frac{W}{L}\right)_p, \qquad \beta_n = k_n'\left(\frac{W}{L}\right)_n \tag{7.73}$$

For a given load capacitance C_L, t_r and t_f can be reduced by using large FETs. However, increasing the aspect ratio of a transistor implies that it will consume more area on the chip, which in turn decreases the number of devices that can be placed on the die area allocated for the circuit. Designing for speed thus decreases the integration density of the circuit. This is called the **speed versus area trade-off** which says that

Fast circuits consume more area than slow circuits

Chip designers regularly face the problem of minimizing the switching delays without requiring excessive amounts of silicon "real estate," which is slang for chip area.

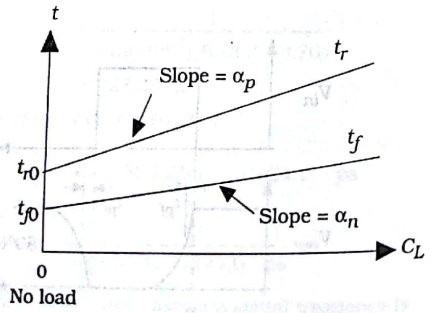


Figure 7.15 General behavior of the rise and fall times

Example 7.4

Let us use the results of Example 7.3 to find the general delay equations for the case where the internal FET capacitance is $C_{FET} = 80$ fF.

The rise time t_r is controlled by the pFET that has a resistance of $R_p = 822.9\ \Omega$. The slope is given by

$$\alpha_p = 2.2R_p = 1,810.4\Omega \tag{7.74}$$

while

$$t_{r0} \approx 2.2R_pC_{FET} = 2.2(822.9)(80\times10^{-15}) = 144.9 \text{ ps} \tag{7.75}$$

The rise time can thus be written in the form

$$t_r = t_{r0} + \alpha_p C_L = 144.9 + 1.810C_L \text{ ps} \tag{7.76}$$

which requires that C_L be in units of fF.

For the fall time equation, we calculate

$$\alpha_n = 2.2(427.35) = 940.2\Omega \tag{7.77}$$

and

$$t_{f0} \approx 2.2(940.2)(80\times10^{-15}) = 165.5 \text{ ps} \tag{7.78}$$

yielding

$$t_f = 165.5 + 0.940C_L \text{ ps} \tag{7.79}$$

as the general expression.

As an example of using these equations, suppose that the load is specified as C_L = 150 fF. We compute

$$t_r = 144.9 + 1.810(150) = 416.4 \text{ ps}$$
$$t_f = 165.5 + 0.940(150) = 306.5 \text{ ps} \tag{7.80}$$

for the rise and fall times at the output. This corresponds to a maximum switching frequency for the gate of $f_{max} \approx 1.38$ GHz.

The relative values of $(W/L)_n$ and $(W/L)_p$ determine the shape of the output waveform. For example, if we design the circuit such that

$$R_p = R_n \tag{7.81}$$

then the output waveform is symmetrical with

$$t_r = t_f \tag{7.82}$$

To equalize the resistances we must design the circuit such that

$$\beta_p(V_{DD} - |V_{Tp}|) = \beta_n(V_{DD} - V_{Tn}) \tag{7.83}$$

is satisfied. If $V_{Tn} = |V_{Tp}|$, then the requirement reduces to

$$\beta_p = \beta_n \tag{7.84}$$

which gives the DC midpoint voltage at $V_M = (V_{DD}/2)$. This illustrates the fact that the nFET/pFET ratio (β_n/β_p) determines the DC midpoint voltage, while the individual values of β_n and β_p establish the switching times t_f and t_r, respectively.

7.2.5 Summary of the Inverter Circuit

It is worth taking the time to summarize the results of our study to this point. The electrical characteristics of an isolated CMOS inverter are established by two sets of parameters:

- The processing variables, such as k' and V_T values, and parasitic capacitances,

and,

- The transistor aspect ratios $(W/L)_n$ and $(W/L)_p$.

VLSI designers do not have any control over the processing parameters, as they are set by the details of the manufacturing sequence. Device sizing thus becomes the critical issue in high-speed circuit design.

System design is accomplished by using cascades of logic gates to perform the necessary binary operations. In electrical terms, the logic flow path establishes the load capacitance C_L seen by each gate. The choice of aspect ratios is the key to achieving the desired transient response of a chain of gates.

7.3 Power Dissipation

An important characteristic of CMOS integrated circuits is the power dissipated by a particular design technique. The general problem is shown in Figure 7.16. The current I_{DD} flowing from the power supply to ground gives a dissipated power of

$$P = V_{DD}I_{DD} \tag{7.85}$$

Since the value of the voltage supply V_{DD} is assumed to be a constant, we can find the value of P by studying the nature of the current flow. We usually divide the currents into DC and dynamic (or switching) contributions, so let us write

$$P = P_{DC} + P_{dyn} \tag{7.86}$$

where P_{DC} is the DC term and P_{dyn} is due to dynamic switching events.

The DC contribution can be calculated by examining the voltage transfer curve reproduced in Figure 7.17(a). When the input voltage V_{in} is stable at a low logic 0 value, the nFET Mn is off; as seen earlier in Figure 7.2, there is no direct current flow path between V_{DD} and ground. Ideally, the

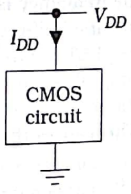


Figure 7.16 Origin of power dissipation calculation

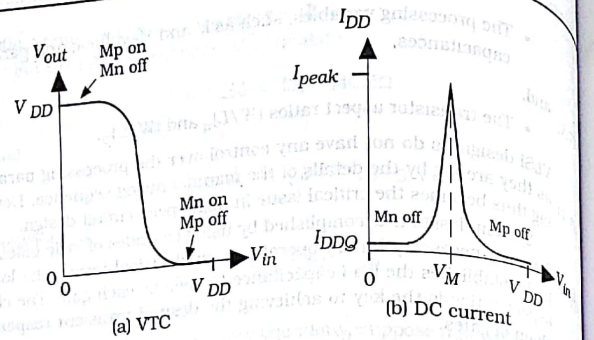


Figure 7.17 DC current flow

DC current flow for this case would be $I_{DD} = 0$, but in a realistic circuit small **leakage currents** exist.[4] The value is denoted as I_{DDQ} and is called the **quiescent** leakage current. When V_{in} is switched, the current flow reaches a peak value I_{peak} at V_M as shown in Figure 7.17(b). However, when the input reaches a logic 1 voltage, then the pFET Mp turns off once again preventing a direct current flow path. If we assume that the inputs are in stable 0 or 1 states as in an idle system, the DC power dissipation is given by

$$P_{DC} = V_{DD} I_{DDQ} \tag{7.87}$$

The leakage current I_{DDQ} is usually quite small, with a typical value on the order of a picoampere per gate. The value of P_{DC} is thus quite small. This consideration was a major factor in the move to CMOS in the mid-1990's.

To find the dynamic power dissipation P_{dyn}, we use a square-wave input voltage $V_{in}(t)$ as shown in Figure 7.18(a). The waveform has a period T corresponding to a switching frequency of

$$f = \frac{1}{T} \tag{7.88}$$

with units of Hertz; the frequency is the number of cycles completed in one second. During the first half-cycle, the input voltage is at a value $V_{in} = 0$. This turns on the pFET Mp as shown in Figure 7.18(b). Since the nFET is off, the current i_{DD} flows through Mp and charges C_{out} to a voltage of $V_{out} = V_{DD}$. During the second half-cycle, the input voltage is high, turning on the nFET Mn. This causes the discharge event illustrated in Figure

[4] These are discussed in more detail in Chapter 9.

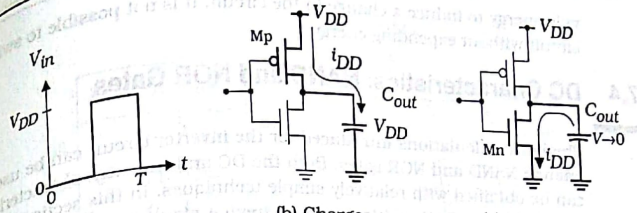


Figure 7.18 Circuit for finding the transient power dissipation

7.18(c) where V_{out} decays to 0 V. The dynamic power P_{dyn} arises from the observation that a complete cycle effectively creates a path for current to flow from the power supply to ground: during the charge event, current flows to the capacitor C_{out} while the discharge path to ground completes the circuit.

To calculate P_{dyn}, we note that the charging event leaves C_{out} with a voltage of $V_{out} = V_{DD}$. This corresponds to a stored electric charge on the capacitor of

$$Q_e = C_{out} V_{DD} \tag{7.89}$$

which has units of coulombs. When the capacitor is discharged through the nFET, the same amount of charge is lost. The average power dissipated over a single cycle with a period T is

$$P_{av} = V_{DD} I_{DD} = V_{DD}\left(\frac{Q_e}{T}\right) \tag{7.90}$$

Substituting for Q_e gives

$$P_{sw} = C_{out} V_{DD}^2 f \tag{7.91}$$

as the switching power. Combining the DC and dynamic power terms gives the total power as

$$P = V_{DD} I_{DDQ} + C_{out} V_{DD}^2 f \tag{7.92}$$

which will usually be dominated by the dynamic term. This illustrates an extremely important point:

- The dynamic power dissipation is proportional to the signal frequency

In other words, a fast circuit dissipates more power than a slow circuit. If we double the switching speed, then the dynamic power dissipation dou-

bles. These are simply statements of the physical law that we must pro-

vide energy to induce a change in the circuit. It is not possible to switch a circuit without expending energy.

7.4 DC Characteristics: NAND and NOR Gates

The basic calculations introduced for the inverter circuit can be used to analyze NAND and NOR gates. Both the DC and transient characteristics can be obtained with relatively simple techniques. In this section we will examine the relationship between device sizes and the transitions described by the VTC.

7.4.1 NAND Analysis

Let us start with the NAND2 gate illustrated in Figure 7.19. We will analyze the case where like-polarity FETs have the same aspect ratio. This means that both pFETs are described by β_p and both nFETs have the same β_n. Since the pFETs are in parallel while the nFETs are in series, the circuit behaves quite differently from the simple inverter.

The presence of two independent inputs implies that more than one VTC curve is needed to describe the circuit. Suppose that we look for transitions where V_{out} is initially high at V_{DD} and then falls to 0 V when inputs are changed. Figure 7.20(a) summarizes the possible starting points that can lead to this situation. In case (i), both V_A and V_B are at 0 V and then switched to the bottom line condition where $V_A = V_B = V_{DD}$ such that $V_{out} = 0$ V. Since both inputs are increased at the same time, this describes the case for simultaneous input switching. The other two possibilities (ii) and (iii) describe cases where only a single input is changed. For example, in (ii) V_A is changed from 0 V to V_{DD} while V_B is held constant at V_{DD}. These three possibilities lead to the three distinct transitions shown in the plot of Figure 7.20(b). This shows that the simultaneous switching case is "pushed to the right" compared to the single-switched input cases.

It is instructive to calculate the value of the midpoint voltage V_M for the

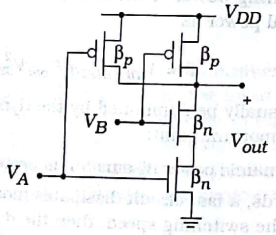
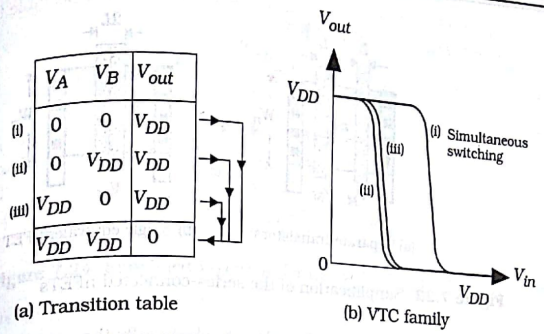


Figure 7.19 NAND2 logic circuit



(a) Transition table (b) VTC family

Figure 7.20 NAND2 VTC analysis

case of simultaneous switching using layout drawings. The circuit problem is illustrated in Figure 7.21, where W_n and W_p are the nFET and pFET channel widths, respectively. All transistors are assumed to have the same channel length L. Now then, for this case both input voltages V_A and V_B are equal to V_M. On the layout plot, both gates are thus at the same potential and can be connected to simplify the calculations.

Consider the nFETs first. In Figure 7.22(a), the layout is shown in its original form with two separate series-connected transistors. Let us "merge" the two gates together into one to obtain the patterning shown in Figure 7.22(b). If we ignore the n+ region that separates the two gates, then the structure can be approximated as a single nFET with an aspect ratio of $(W_n/2L)$ as shown. Since the original nFETs each had a device transconductance of β_n, the single equivalent transistor is described by the value $(\beta_n/2)$.

The pFETs can be combined in a similar manner. The original parallel-connected transistors are illustrated in Figure 7.23(a). Owing to the paral-

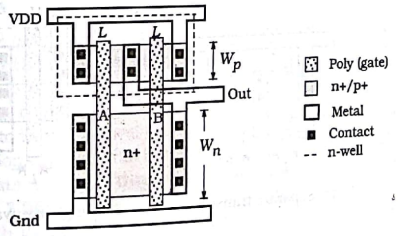
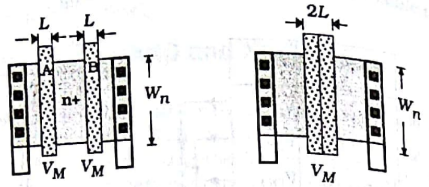


Figure 7.21 Layout of NAND2 for V_M calculation



(a) Separate transistors (b) Single equivalent FET

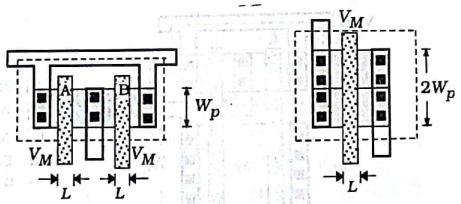
Figure 7.22 Simplification of the series-connected nFETs

lel wiring, the left and right sides are electrically the same point, so that the two may be simplified into the single gate structure shown in Figure 7.23(b). In this case, the two combine to act as a single pFET with an aspect ratio of $(2W_p/L)$. If the original devices each have β_p, then the equivalent structure acts as a pFET with $2\beta_p$.

Let us now use these results to find V_M for the case of simultaneous switching. Replacing the transistor pairs by their single-FET equivalents gives the inverter circuit in Figure 7.24, where the nFET and pFET transconductances are $(\beta_n/2)$ and $2\beta_p$, respectively. The calculation then proceeds in the same manner as for the "normal" NOT gate. Both transistors are saturated, so equating currents gives

$$\frac{(\beta_n/2)}{2}(V_M - V_{Tn})^2 = \frac{(2\beta_p)}{2}(V_{DD} - V_M - |V_{Tp}|)^2 \qquad (7.93)$$

Taking square roots of both sides and solving for the midpoint voltage results in the expression



(a) Separate transistors (b) Single equivalent FET

Figure 7.23 Simplification of parallel-connected pFETs

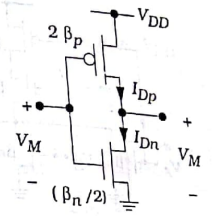


Figure 7.24 Simplified V_M circuit for the NAND2 gate

$$V_M = \frac{V_{DD} - |V_{Tp}| + \frac{1}{2}\sqrt{\frac{\beta_n}{\beta_p}}V_{Tn}}{1 + \frac{1}{2}\sqrt{\frac{\beta_n}{\beta_p}}} \qquad (7.94)$$

This has the same form as the NOT gate in equation (7.14), except that the square root term is multiplied by a factor of (1/2). This reduces the denominator, which is why the VTC curve is shifted toward the right. If we apply the same reasoning to an *N*-input NAND gate, the simultaneous switching point is found to be

$$V_M = \frac{V_{DD} - |V_{Tp}| + \frac{1}{N}\sqrt{\frac{\beta_n}{\beta_p}}V_{Tn}}{1 + \frac{1}{N}\sqrt{\frac{\beta_n}{\beta_p}}} \qquad (7.95)$$

The right shift is due to the series-connected nFETs, since their resistances add.

.4.2 NOR Gate

The NOR2 gate can be analyzed using the same techniques. We assume that the nFETs have the same β_n and that both pFETs are described by β_p as shown in the basic circuit of Figure 7.25. To construct VTC, note that $V_{out} = V_{DD}$ requires that $V_A = V_B = 0$ V. If either input (or both) are switched to logic 1 values, then the output will fall to $V_{out} = 0$ V. The three combinations are listed in the function table of Figure 7.26(a). As with the NAND2 gate, there are three distinct transitions shown in the VTC family of Figure 7.26(b). Case (i) describes the simultaneous switching event where both V_A and V_B are increased from 0 V toward V_{DD}. This case is the leftmost plot in the VTC family, exactly opposite to that found for the NAND2. Single-input switching cases (ii) and (iii) are distinct, but are close to each other.

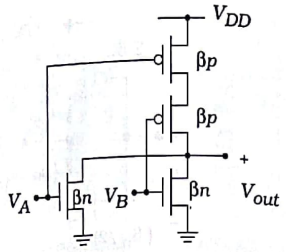


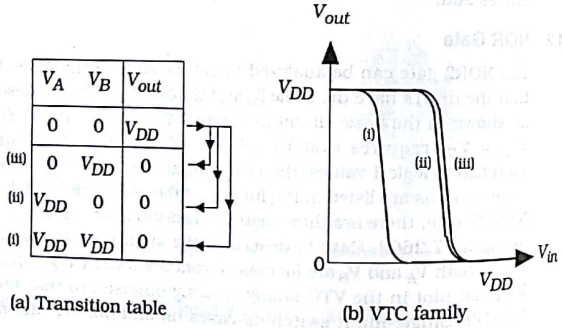
Figure 7.25 NOR2 circuit

The techniques of combining series and parallel transistors may be used to compute V_M for the simultaneous switching case. Since the nFETs are in parallel, they may be combined to a single equivalent nFET with a transconductance of $2\beta_n$. The series-connected pFETs act as a single pFET with $(\beta_p/2)$ which gives rise to the simplified equivalent circuit in Figure 7.27. Equating the saturation currents using the effective transconductance values gives us

$$\frac{(2\beta_n)}{2}(V_M - V_{Tn})^2 = \frac{(\beta_p/2)}{2}(V_{DD} - V_M - |V_{Tp}|)^2 \tag{7.96}$$

This may be solved to give

$$V_M = \frac{V_{DD} - |V_{Tp}| + 2\sqrt{\frac{\beta_n}{\beta_p}}V_{Tn}}{1 + 2\sqrt{\frac{\beta_n}{\beta_p}}} \tag{7.97}$$



(a) Transition table (b) VTC family

Figure 7.26 NOR2 VTC construction

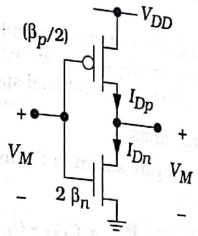


Figure 7.27 NOR2 V_M calculation for simultaneous switching

Comparing this with the NOT and NAND expressions shows that the only difference is the factor of 2 multiplying the square root term. This increases the denominator, which decreases the value of V_M from that of an inverter with a device ratio of (β_n/β_p). The midpoint voltage for an N-input NOR gate is

$$V_M = \frac{V_{DD} - |V_{Tp}| + N\sqrt{\frac{\beta_n}{\beta_p}}V_{Tn}}{1 + N\sqrt{\frac{\beta_n}{\beta_p}}} \tag{7.98}$$

It is worthwhile noting that the NAND and NOR gates tend have opposite behaviors with respect to the reference NOT gate VTC.

As a final comment, we note that both the NAND and NOR gates exhibit low DC power dissipation values of

$$P_{DC} = V_{DD}I_{DDQ} \tag{7.99}$$

since there is no direct current flow path from the power supply to ground when the inputs are stable logic 0 or logic 1 values. The low power characteristic of the gates is due to the use of complementary pairs and series-parallel structuring of the transistor arrays. Dynamic power is still present in the general form

$$P_{sw} = C_{out}V_{DD}^2 f_{gate} \tag{7.100}$$

which shows the dependence on gate switching frequency f_{gate}. Since it takes more than a single input to switch the gate, f_{gate} is different from the basic switching frequency used for the inverter. This is discussed in more detail later.

7.5 NAND and NOR Transient Response

Transient switching times often represent the limiting factor in designing a digital logic chain. In this section we will examine how the FET topology and device sizing affect the operational speed of the gate.

7.5.1 NAND2 Switching Times

Consider the NAND2 gate shown in Figure 7.28. The total output capacitance is denoted as

$$C_{out} = C_{FET} + C_L \tag{7.101}$$

where C_L is the external load and

$$C_{FET} = C_{Dn} + 2C_{Dp} \tag{7.102}$$

represents the parasitic internal FET capacitances. Note that there are two contributions of C_{Dp} since two pFETs are connected to the output node. The drawing identifies the transistors by their resistance values

$$R_p = \frac{1}{\beta_p(V_{DD} - |V_{Tp}|)}, \qquad R_n = \frac{1}{\beta_n(V_{DD} - V_{Tn})} \tag{7.103}$$

The transient calculations are based on finding RC time constants for the charge time (t_r or t_{LH}) and fall time (t_f or t_{HL}) for the transitions. The procedure is complicated by the presence of two inputs. We will concentrate on estimating the worst-case values of the switching times.

Let us consider the rise time t_r first. The output voltage is initially at a value $V_{out}(0) = 0$ V and is then charged to V_{DD}. If only one pFET is conducting, we obtain the simplified charging circuit shown in Figure 7.29(a) where C_{out} charges through a pFET resistance R_p. Since this looks like the charging circuit for a simple inverter, we can write

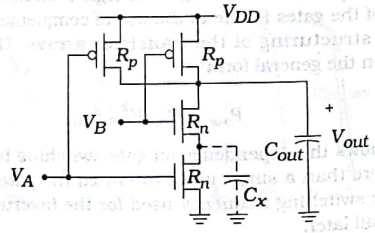
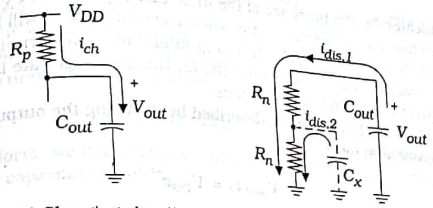


Figure 7.28 NAND2 circuit for transient calculations



(a) Charging circuit (b) Discharging circuit

Figure 7.29 NAND2 subcircuits for estimating rise and fall times

$$V_{out}(t) = V_{DD}[1 - e^{-t/\tau_p}] \tag{7.104}$$

where

$$\tau_p = R_p C_{out} \tag{7.105}$$

is the time constant. The rise time is thus given by

$$t_r \approx 2.2\tau_p \tag{7.106}$$

This is considered to be a "worst-case" situation since only one pFET is charging C_{out}. Note that this can be cast into the linear form

$$t_r = t_0 + \alpha_0 C_L \tag{7.107}$$

where

$$t_0 = 2.2R_p C_{FET} \tag{7.108}$$

is the zero-load value, and

$$\alpha_0 = 2.2R_p \tag{7.109}$$

is the slope of t_r as a function of the load capacitance C_L. If both pFETs are conducting, then the equivalent resistance is lowered to $(R_p/2)$ since the two are in parallel; this would be the "best-case" event, i.e., the one with the shortest charging time. Design is usually based on worst-case analysis since we want to insure that the circuit operates under all conditions.

The situation is more complicated when we analyze the fall time t_f where C_{out} discharges through the series-connected nFET chain. RC modeling of each device leads to the "ladder" network shown in Figure 7.29(b). While the main item of interest is discharging C_{out}, the situation is com-

plicated by the presence of the inter-FET capacitance C_X between the two n-channel transistors. In the worst-case analysis, C_X will have charge that will flow through nFET MnA to ground. Since the current through a FET is limited by its aspect ratio (W/L), the discharge rate is limited by the current that MnA can maintain.

The discharge can be described by modeling the output voltage in the exponential form

$$V_{out}(t) = V_{DD}e^{-t/\tau_n} \tag{7.110}$$

such that the time constant is given by the **Elmore formula** as

$$\tau_n = C_{out}(R_n + R_n) + C_X R_n \tag{7.111}$$

This estimates the time constant as the superposition of time constants

$$\tau_n = \tau_{n1} + \tau_{n2} \tag{7.112}$$

where

$$\tau_{n1} = C_{out}(R_n + R_n) \tag{7.113}$$

is the time constant for C_{out} discharging through two nFETs, each with a resistance R_n; this is shown by the current $i_{dis,1}$ in the drawing. The other term

$$\tau_{n2} = C_X R_n \tag{7.114}$$

is the time constant for C_X discharging through one nFET with a resistance R_n. This is corresponds to the discharge current $i_{dis,2}$. The fall time t_f is then given by

$$t_f \approx 2.2\tau_n \tag{7.115}$$

Substituting the time constant expression transforms this into

$$t_f \approx 2.2[(C_{FET} + C_L)(2R_n) + C_X R_n] \tag{7.116}$$

Grouping terms results in the linear expression

$$t_f = t_1 + \alpha_1 C_L \tag{7.117}$$

with a zero-load delay of

$$t_1 = 2.2R_n(2C_{FET} + C_X) \tag{7.118}$$

and a slope of

$$\alpha_1 = 4.4R_n \tag{7.119}$$

where the multiplier is from (2×2.2). Although we are able to write t_f as a

linear function of C_L, both the zero-load delay and the slope are affected by the series-connected nFETs in the discharge circuitry.

The Elmore formulation of time constants for RC ladder-type networks illustrates that series-connected FETs lead to longer delays in CMOS circuits. To understand this comment, let us rewrite equation (7.111) as

$$\tau_n = R_n(2C_{out} + C_X) \tag{7.120}$$

In this form, we can interpret the time constant as R_n multiplying an effective capacitance with a value

$$C_{eff} = 2C_{out} + C_X \tag{7.121}$$

which is larger than twice the output capacitance. Alternately, we may write

$$\tau_n = C_{out}(2R_n) + C_X R_n \tag{7.122}$$

which clearly shows the effect of the series-connected FETs in the term $2R_n$ and the increase due to the parasitic capacitance C_X. Regardless of the interpretation one chooses, it is important to remember that series-connected FET chains can lead to excessive logic delays.

7.5.2 NOR2 Switching Times

The analysis of the NOR2 transients proceeds in the same manner. Figure 7.30 shows the circuit with FET resistances and the capacitances. The output capacitance for any gate is given by the general form

$$C_{out} = C_{FET} + C_L \tag{7.123}$$

For the NOR2 circuit, the internal capacitance can be broken down into components as

$$C_{FET} = 2C_{Dn} + C_{Dp} \tag{7.124}$$

since there are two nFETs connected to the output node but only one

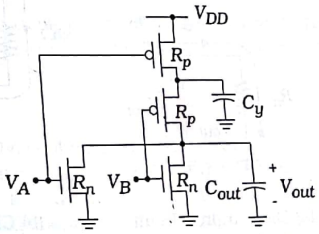


Figure 7.30 NOR2 circuit for switching time calculations

pFET. The inter-FET capacitance C_y represents the parasitic contribu-
tions between the two pFETs.

Figure 7.31 shows the subcircuits for the output transients. The fall
time t_f may be computed using the worst-case circuit in Figure 7.31a
where only one nFET acts to discharge the output capacitance. We thus
write the output voltage as

$$V_{out}(t) = V_{DD}e^{-t/\tau_n} \tag{7.125}$$

with

$$\tau_n = R_n C_{out} \tag{7.126}$$

as the time constant. The fall time is then given by

$$t_f \approx 2.2\tau_n \tag{7.127}$$

which is identical to that for a simple inverter. Expanding C_{out} gives the
linear dependence

$$t_f = t_1 + \alpha_1 C_L \tag{7.128}$$

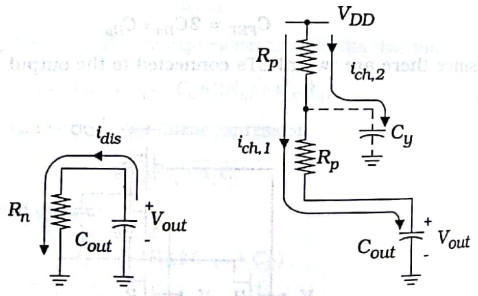
where the zero-load delay is

$$t_1 = 2.2R_n C_{FET} \tag{7.129}$$

and the slope is

$$\alpha_1 = 2.2R_n \tag{7.130}$$

These results are similar to the NOT gate, but it is important to remember
that C_{FET} is larger for the NOR2 gate.



(a) Discharging circuit (b) Charging circuit

Figure 7.31 Subcircuits for the NOR2 transient calculations

The charging circuit for finding the rise time t_r is shown in Figure
7.31(b). We will write the output voltage in the exponential form

$$V_{out}(t) = V_{DD}[1 - e^{-t/\tau_p}] \tag{7.131}$$

However, since C_y will be charged during this event, we must use the
Elmore formula to find the time constant. The two paths are shown as
$i_{ch,1}$ and $i_{ch,2}$ in the drawing. The primary charge path due to $i_{ch,1}$ is
described by a time constant

$$\tau_1 = C_{out}(R_p + R_p) \tag{7.132}$$

while that associated with $i_{ch,2}$ is

$$\tau_2 = C_y R_p \tag{7.133}$$

Superposing gives the total effective time constant in the form

$$\begin{aligned}\tau_p &= \tau_1 + \tau_2 \\ &= C_{out}(2R_p) + C_y R_p\end{aligned} \tag{7.134}$$

such that the rise time is

$$t_r = 2.2\tau_p \tag{7.135}$$

Since the series-connected pFETs introduce a large time constant, the
rise time may be quite large compared to the fall time. Substituting for
C_{out} gives the linear equation

$$t_r = t_0 + \alpha_0 C_L \tag{7.136}$$

where

$$t_0 = 2.2R_p(2C_{FET} + C_y) \tag{7.137}$$

and

$$\alpha_0 = 4.4R_p \tag{7.138}$$

characterize the dependence of t_r on C_L. As with the NAND2 gate, the
presence of series-connected FETs slows down the associated switching
time.

7.5.3 Summary

The analyses above illustrate that the NAND and NOR gates exhibit com-
plementary characteristics at both the DC and transient levels. This
arises because they are constructed using complementary series-parallel
transistor arrangements.

While the DC characteristics are important, most design effort is

directed toward minimizing delays through logic chains. The study above allows us to make some general statements about NAND and NOR gates as compared to the simpler NOT circuit. First, we have seen that the rise time can be written in the form

$$t_r = t_0 + \alpha_0 C_L \tag{7.139}$$

while the fall time has the same structure with

$$t_f = t_1 + \alpha_1 C_L \tag{7.140}$$

The constants (t_0 and α_0 for the rise time, and t_1 and α_1 for the fall time) depend upon the parasitic transistor resistances and capacitances. The constants are the smallest for a NOT gate, so we often use it as a reference. This, of course, is because the inverter consists of only two FETs. In general, adding complementary transistor pairs increases the delay times because C_{FET} is increased. The number of inputs to a logic gate is called the **fan-in** (FI). Since every input is connected to a complementary pair, we can state that

- Switching delays increase with the fan-in.

This says, for example, a NAND3 gate will be slower than a NAND2 gate if the two use the same size transistors. Of course, the actual delay depends upon the value of the load capacitance C_L such that

- Switching delays increase with the external load.

Since logic functions are implemented using cascades of gates, the effect of this dependence varies with the circuit.

Let us summarize the results of the NAND and NOR analysis. As with the inverter, the electrical characteristics of these gates are set by

- The processing variables and
- The aspect ratios $(W/L)_n$ and $(W/L)_p$ of every FET

Furthermore, series transistors introduced us to the problem of parasitic capacitance between the two devices. This factor leads us to make one additional statement

- The details of the layout geometries affect the transient response of the logic gate.

We thus conclude that the physical layout and structure of the circuitry is a critical factor in designing high-speed logic networks.

7.6 Analysis of Complex Logic Gates

The analysis techniques developed for the NAND and NOT circuits may be extended to analyze complex CMOS logic gates with AOI and OAI structuring. The most important problem is the transient delay associated with series-connected FETs.

Consider the complex logic gate shown in Figure 7.32. This implements the logic function

$$f = \overline{x \cdot (y + z)} \tag{7.141}$$

with series-parallel FET arrays. The aspect ratio values shown in the drawing are the critical parameters that affect the rise and fall times. The fall time is governed by the nFETs. If we assume that they are all the same size with

$$\left(\frac{W}{L}\right)_{nx} = \left(\frac{W}{L}\right)_{ny} = \left(\frac{W}{L}\right)_{nz} \tag{7.142}$$

then the nFET resistance R_n can be used to describe each one. The worst-case fall time will occur when $x = 1$, but only one of the ORed inputs y or z is 1. This results in a 2-FET series pair that must handle the discharge of the output capacitor

$$C_{out} = C_{FET} + C_L \tag{7.143}$$

With the capacitance C_n in the chain, the time constant is

$$\tau_n = R_n C_n + 2R_n C_{out} \tag{7.144}$$

which gives a fall time of

$$\begin{aligned} t_f &= 2.2\tau_n \\ &= 2.2R_n[C_n + 2(C_{FET} + C_L)] \\ &= t_1 + \alpha_1 C_L \end{aligned} \tag{7.145}$$

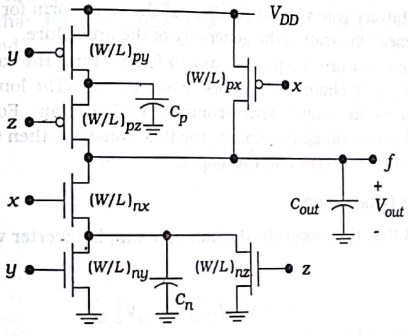


Figure 7.32 Complex logic gate circuit

where

$$t_1 = 2.2R_n(C_n + 2C_{FET}) \tag{7.146}$$

is the zero-load time, and

$$\alpha_1 = 2.2R_n \tag{7.147}$$

is the slope.

The rise time t_r is determined by the pFETs. If these are chosen with equal aspect ratios

$$\left(\frac{W}{L}\right)_{px} = \left(\frac{W}{L}\right)_{py} = \left(\frac{W}{L}\right)_{pz} \tag{7.148}$$

then we can use the same R_p for each device. The limiting series chain is with the y and z input p-channel transistors; the x-input pFET provides the fast switching, and could be decreased to half-size without affecting the results. The series chain gives a time constant of

$$\tau_p = R_pC_p + 2R_pC_{out} \tag{7.149}$$

where C_p is the parasitic capacitance between the pFETs. The worst-case rise time is thus of the form

$$t_r = t_0 + \alpha_0 C_L \tag{7.150}$$

where the zero-load delay is

$$t_0 = 2.2R_p(C_p + 2C_{FET}) \tag{7.151}$$

and the slope is

$$\alpha_0 = 2.2R_p \tag{7.152}$$

An arbitrary gate yields equations of the same form for both the rise and fall times, illustrating the generality of the procedure.

The important steps are easy to follow. Find the longest series-connected nFET chain for the worst-case fall time. The longest rise time will be due to the longest series-connected pFET chain. For both cases, use the Elmore formula to compute the time constant, then separate terms for the zero bias delays and the slopes.

7.6.1 Power Dissipation

Recall that the power dissipation in a simple inverter was written in the form

$$P = V_{DD}I_{DDQ} + C_{out}V_{DD}^2 f \tag{7.153}$$

When we analyze a general static CMOS logic gate, the DC term is still small, but the dynamic switching power P_{dyn} becomes important in high-speed, high-density designs.

To model the dynamic power dissipation of an arbitrary gate we recall that P_{dyn} originates from an output switching event. First, the output capacitor C_{out} is charged from 0 V to V_{DD}, corresponding to an output logic 0→1 transition. Then, C_{out} discharges to give a 1→0 transition, completing the cycle. To model the number of transitions that take place over a switching period T we introduce the **activity coefficient** a that represents the probability that an output 0→1 transition takes place during one period. The dynamic power is then modified to read

$$P_{dyn} = aC_{out}V_{DD}^2 f \tag{7.154}$$

For a network that consists of N gates, the total dynamic power is more generally written in the form

$$P_{dyn} = \sum_{i=1}^{N} a_i C_i V_i V_{DD} f \tag{7.155}$$

where, for the i-th gate, a_i is the activity coefficient and C_i is the node capacitance that charges to a maximum value of V_i.

Activity coefficients can be determined from truth tables. Figure 7.33 provides the truth tables for the NOR2 and NAND2 functions. We will assume that each input combination has equal probability of occurring. Let us analyze the NOR2 transitions first. Since the activity factor a_{NOR2} is the probability that the gate makes a 0 → 1 transition, it can be calculated by

$$a = p_0 p_1 \tag{7.156}$$

where p_0 is the probability that the output is initially at 0, and p_1 the probability that it makes a transition to 1. The truth table shows us that $p_0 = (3/4)$ and $p_1 = (1/4)$, so

A	B	$\overline{A+B}$	$\overline{A \cdot B}$
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

Figure 7.33 Truth tables for determining activity coefficients

$$a_{NOR2} = \left(\frac{3}{4}\right)\left(\frac{1}{4}\right) = \frac{3}{16} \tag{7.157}$$

The NAND2 gate can be analyzed in the same manner. For this gate, the truth table shows that $p_0 = (1/4)$ and $p_1 = (3/4)$ so

$$a_{NAND2} = \left(\frac{3}{4}\right)\left(\frac{1}{4}\right) = \frac{3}{16} \tag{7.158}$$

has the same value as the NOR2 gate. If we look at 3-input gates, the truth tables give

$$a_{NOR3} = \frac{7}{64} = a_{NAND3} \tag{7.159}$$

Similarly, we can calculate

$$a_{XNOR2} = \frac{1}{4} = a_{XOR2} \tag{7.160}$$

since $p_0 = (1/4) = p_1$. The technique can be applied to an arbitrary gate.

The limit on this simple treatment is that, in practice, we rarely have input combinations that occur with equal probability. More advanced techniques have been developed to handle these situations. The interested reader is directed to Reference [2] for an excellent discussion of the details. Reference [8] is a very thorough analysis of power dissipation and low-power design.

7.7 Gate Design for Transient Performance

High-speed circuits are limited by the switching time of individual gates. Logic formation determines the series and parallel connections of the transistors. The aspect ratios are the critical design parameters for both the DC and transient switching times. Once these are specified and the transistors are created in the layout, all of the parasitics are set.

The DC switching characteristics are often considered less important than the switching speed. It is common to design a gate to have the desired transient times, and then check the DC VTC to insure that it is acceptable. This approach is based on the fact that the individual nFET and pFET aspect ratios determine the switching response, while the DC transition point is a result of the ratio of the nFET to pFET values. For example, the value of β_n/β_p gives V_M for an inverter, while t_r depends primarily on β_p and t_f is established by β_n.

The design philosophy used to select aspect ratios varies with the situation. A straightforward approach is to use the inverter as a reference and then attempt to design other gates that have approximately the same switching times. Since the NOT gate is the simplest, it can be built using

relatively small transistors. We will use the device transconductance

$$\beta = k'\left(\frac{W}{L}\right) \tag{7.161}$$

as being equivalent to the aspect ratio.

Figure 7.34(a) shows an inverter with device sizes specified by β_p and β_n, which we will assume are known. These set the rise and fall times t_r and t_f for the circuit, which serve as the reference switching times. Since both transistors drive the same capacitance, the difference is in the resistance values

$$R_p = \frac{1}{\beta_p(V_{DD} - |V_{Tp}|)}, \qquad R_n = \frac{1}{\beta_n(V_{DD} - V_{Tn})} \tag{7.162}$$

Recall that a symmetrical inverter has

$$\beta_n = \beta_p \tag{7.163}$$

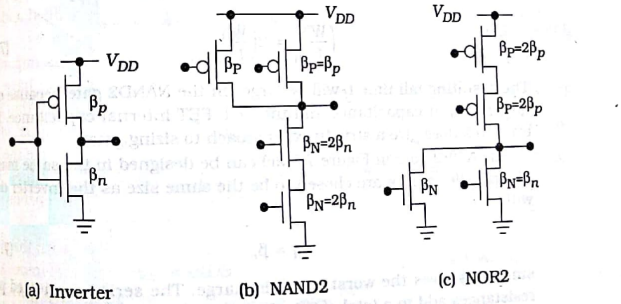
and requires the device sizes to be related by

$$\left(\frac{W}{L}\right)_p = r\left(\frac{W}{L}\right)_n \tag{7.164}$$

where

$$r = \frac{k'_n}{k'_p} \tag{7.165}$$

is the process transconductance ratio. A nonsymmetrical design that uses equal size transistors such that $\beta_n > \beta_p$ is also commonly used as a reference.



(a) Inverter (b) NAND2 (c) NOR2

Figure 7.34 Relative FET sizing

Let us use these values to find the device sizes β_P and β_N for the NAND2 gate in Figure 7.34(b) with the philosophy that we want to achieve simil... rise and fall times. Consider first the parallel pFETs. Since the worst-ca... situation is where only one transistor contributes to the rise time, we ... select the same size as the inverter:

$$\beta_P = \beta_p \tag{7.16}$$

The actual rise time t_r will be longer than that of the inverter because C_o... is larger. The series-connected nFET chain has to be modeled as tw... series-connected resistors between the output and ground, with a tot... value of

$$R = R_N + R_N \tag{7.16}$$

where

$$R_N = \frac{1}{\beta_N(V_{DD} - V_{Tn})} \tag{7.16}$$

Using the inverter as a reference, we set

$$R = R_n = 2R_N \tag{7.16}$$

Substituting,

$$\frac{1}{\beta_n(V_{DD} - V_{Tn})} = \frac{2}{\beta_N(V_{DD} - V_{Tn})} \tag{7.170}$$

which has the solution

$$\beta_N = 2\beta_n \tag{7.171}$$

i.e., the series-connected nFETs are twice as large as the inverter transis... tor:

$$\left(\frac{W}{L}\right)_N = 2\left(\frac{W}{L}\right)_n \tag{7.172}$$

The resulting fall time t_f will be larger in the NAND2 gate because of the larger output capacitance and the FET-FET internal capacitance. How... ever, this does give a structured approach to sizing gates.

The NOR2 gate in Figure 7.34(c) can be designed in the same manner. The parallel nFETs are chosen to be the same size as the inverter device with

$$\beta_N = \beta_n \tag{7.173}$$

since this gives the worst-case discharge. The series-connected pFET resistances add to a total of $2R_p$. Equating this to the inverter resistance

R_p gives

$$\frac{1}{\beta_p(V_{DD} - |V_{Tp}|)} = \frac{2}{\beta_P(V_{DD} - |V_{Tp}|)} \tag{7.174}$$

so that

$$\beta_P = 2\beta_p \tag{7.175}$$

indicating that the pFETs are twice as large as the inverter transistors:

$$\left(\frac{W}{L}\right)_P = 2\left(\frac{W}{L}\right)_p \tag{7.176}$$

The main problem is that pFETs are intrinsically slow, so that the value of $(W/L)_p$ may be large to begin with.

This technique can be extended to larger chains. For n series-connected FETs, the size must be n times larger than the inverter value. The NAND3 gate in Figure 7.35(a) would thus be designed with

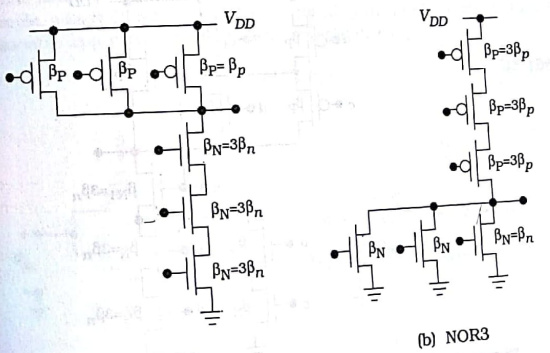
$$\beta_N = 3\beta_n, \qquad \beta_P = \beta_p \tag{7.177}$$

such that

$$\left(\frac{W}{L}\right)_N = 3\left(\frac{W}{L}\right)_n, \qquad \left(\frac{W}{L}\right)_P = \left(\frac{W}{L}\right)_p \tag{7.178}$$

while the NOR3 gate in Figure 7.35(b) would have

$$\beta_N = \beta_n, \qquad \beta_P = 3\beta_p \tag{7.179}$$



(a) NAND3 (b) NOR3

Figure 7.35 Sizing for 3-input gates

with

$$\left(\frac{W}{L}\right)_N = \left(\frac{W}{L}\right)_n \quad , \quad \left(\frac{W}{L}\right)_P = 3\left(\frac{W}{L}\right)_p \tag{7.18}$$

Since the reference values β_n and β_p are arbitrary, the sizes can be adjusted as needed to accommodate reasonable values. Also note that if we select a symmetric inverter design with $\beta_n = \beta_p$, then the resulting gates will also be approximately symmetric.

Complex logic gates can be designed in the same manner. Consider the gate in Figure 7.36 that has an output of

$$f = \overline{(a \cdot b + c \cdot d) \cdot x} \tag{7.18}$$

using series-parallel structuring. Consider the nFET array first. Any discharge event will have current flow through a minimum of three series-connected nFETs. The device sizes would all be the same with the value

$$\beta_N = 3\beta_n = \beta_{N1} \tag{7.18}$$

The pFET array is a little different. The worst-case charge path is through two series-connected transistors on the left side of the circuit. The sizes would be

$$\beta_P = 2\beta_p \tag{7.18}$$

for the pFETs in the inputs *a*, *b*, *c*, and *d*. The *x*-input pFET is alone, so

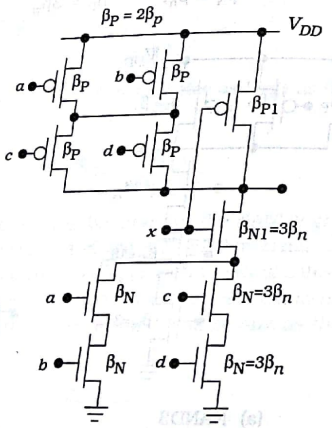


Figure 7.36 Sizing of a complex logic gate

that we can select its size as being the same as for an inverter:

$$\beta_{P1} = \beta_p \tag{7.184}$$

Alternately, the choice

$$\beta_{P1} = \beta_P = 2\beta_p \tag{7.185}$$

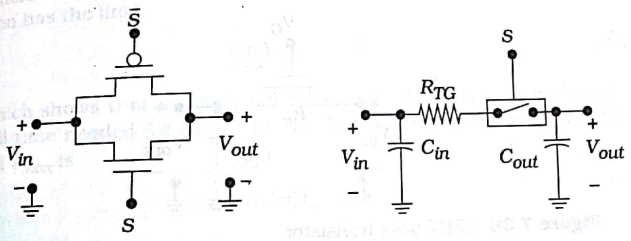
may lead to simpler layout since only a single size pFET would be used. Note that the two options for β_{P1} result in different input capacitances for the *x*-input.

Although this approach provides a nice structured methodology, it leads to large transistors. The designer must decide whether the real estate consumption is worth the added speed. This becomes more complicated as the number of FETs increases since the FET-to-FET parasitic capacitance terms in the Elmore time constant formula will also increase. In practice, we may just select a standard cell that meets the area allocation and then find the overall speed of the logic cascade. If the design is not fast enough, we can apply some of the techniques in the next chapter to find a better design.

7.8 Transmission Gates and Pass Transistors

Transmission gates consist of an nFET/pFET pair wired in parallel as shown in Figure 7.37(a). The RC switching model shown in Figure 7.37(b) consists of a TG resistance R_{TG} and capacitances that account for the parasitic contributions of both FETs. Even though the FETs are in parallel, one usually dominates the conduction process at any given time. For example, a logic 0 transmission is controlled by the nFET. Owing to this, a reasonable approximation for the linear resistance is

$$R_{TG} = \max(R_n, R_p) \tag{7.186}$$



(a) Circuit (b) RC model

Figure 7.37 Transmission gate modeling

i.e., we use the larger of the two values. The capacitances are obtained by adding the contributions. For example, assuming that the left side is at a lower voltage than the right side,

$$C_{in} = C_{S,n} + C_{D,p} \tag{7.187}$$

since the left side of the nFET is the source, while the same node is the drain of the pFET.[5] We note the trade-off in selecting the aspect ratios for the two transistors: large values of (W/L) decrease the resistance, but a large W implies large capacitances. This has made TGs less and less attractive during the evolution of high-density VLSI.

An important electrical feature of the transmission gate (and the pass FETs discussed below) is that there are no direct signal connections to the power supply V_{DD} or ground. Static logic circuits are able to provide full rail outputs $V_{OH} = V_{DD}$ and $V_{OL} = 0$ V by using a power supply rail. Since the TG is not used in this manner, the driving circuit (the one preceding the transmission gate) is responsible for providing the input signal voltages. However, the TG appears to be an RC parasitic to the driving gate, so the response is slower than if the TG were absent. Additional buffer circuits are thus needed to maintain the speed.

Pass transistors are single FETs that pass the **signal** between the drain and source terminals instead of a fixed power supply value. "Pass FETs" can be used in place of transmission gates in most circuits. They require less area and wiring, but cannot pass the entire voltage range. When choosing between the two polarities, nFETs are preferred for this application since the larger electron mobility implies faster switching than could be obtained with pFETs of the same size.

The basic nFET pass circuit is shown in Figure 7.38. The switch is controlled by the gate voltage V_G. If $V_G = 0$, then the transistor is off and there is no connection between the input and output. Placing a high voltage of $V_G = V_{DD}$ drives the nFET active, and current can flow. For the case of a

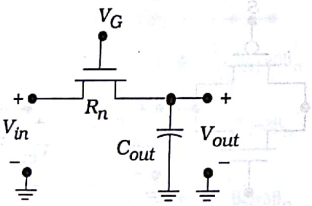


Figure 7.38 nFET pass transistor

[5] Remember that the drain and source are determined by the relative voltages.

logic 1 transfer, we use an input voltage of $V_{in} = V_{DD}$. Assuming an initial condition of $V_{out}(t = 0) = 0$, the analysis gives[6]

$$V_{out}(t) = V_{max}\left(\frac{t/2\tau_n}{1 + t/2\tau_n}\right) \tag{7.188}$$

where

$$V_{max} = V_{DD} - V_{Tn} \tag{7.189}$$

is the maximum voltage transferred through an nFET as seen by taking the limit

$$\lim_{t \to \infty} V_{out}(t) = V_{max} \tag{7.190}$$

This clearly exhibits the threshold drop problem. The time constant is defined by

$$\tau_n = R_n C_{out} \tag{7.191}$$

but does not have the same interpretation as when it appears in an exponential. The rise time needed for the output voltage to rise from 0 V to a value of 0.9 V_{max} is calculated as

$$t_r = 18\tau_n \tag{7.192}$$

These results show that the logic 1 transfer event is slow and suffers from the threshold loss problem.

A logic 0 transfer is analyzed by placing $V_{in} = 0$ V. With the initial condition $V_{out}(0) = V_{max}$, the analysis gives

$$V_{out}(t) = V_{max}\left(\frac{2e^{-(t/\tau_n)}}{1 + e^{-(t/\tau_n)}}\right) \tag{7.193}$$

where the time constant has the same definition. This exponential function has the limit

$$\lim_{t \to \infty} V_{out}(t) = 0 \tag{7.194}$$

which shows that an nFET can pass a logic 0 without any problems. The fall time needed for the output to change from V_{max} to the 10% voltage $0.1V_{max}$ is

$$t_f = \ln(19)\tau_n \approx 2.94\tau_n \tag{7.195}$$

[6] See Reference [10] for the details of the derivation.

Comparing the rise and fall times shows that

$$t_r \approx 6t_f \tag{7.196}$$

so the rise time is the limiting factor. The plot in Figure 7.39 is an example of the shapes of the input versus the output waveforms for an nFET pass transistor.

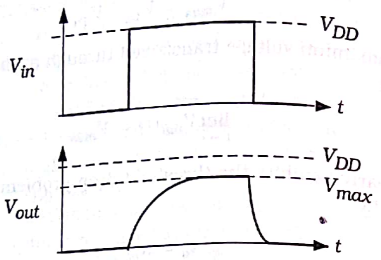


Figure 7.39 Voltage waveforms for a nFET pass transistor

If we use a pFET as a pass transistor, we find complementary results. The maximum voltage through the FET is V_{DD}, and the output charges quite rapidly with a rise time of

$$t_r = 2.94\tau_p \tag{7.197}$$

where

$$\tau_p = R_p C_{out} \tag{7.198}$$

The pFET is thus able to pass a strong logic 1 voltage. When a logic 0 is applied at the input, however, the output discharges to a level

$$V_{min} = |V_{Tp}| \tag{7.199}$$

with a fall time of

$$t_f = 18\tau_p \tag{7.200}$$

The discharge is thus the limiting factor. These results are expected due to the complementary behavior of nFETs and pFETs.

The analysis shows that pass transistors cannot be accurately modeled as simple RC circuits, since the threshold losses and the asymmetrical rise and fall times would be ignored. Regardless of this fact, however, it is common practice to model a pass FET using R_n or R_p in hand calculations during the initial design phase. This allows for quick modeling estimates and is a valuable approximation technique. More precise calculations can be obtained using a computer simulation.

7.9 Comments on SPICE Simulations

The analyses performed in this chapter provide the theoretical basis for designing CMOS logic gates. They allow one to estimate the behavior of a circuit and illustrate the dependence of the overall performance on individual device parameters.

Analytic treatments are intrinsically limited by the accuracy of the device models. In the case of MOSFETs, the square law model is only a low-order approximation to the true behavior. Another level of estimation was introduced with the assumption of step-like input voltage waveforms. We have also ignored the voltage dependence of the capacitances to simplify the analysis. In chip design, the operation of a circuit must be verified by computer simulations. These are not foolproof, as convergence problems and computational noise can affect the results. However, they do provide reasonable verification once the designer becomes familiar with the problem areas. In this section we will examine a few important features of SPICE simulations.

A SPICE netlist for a circuit is obtained from the extraction routine in the layout editor. Each element is represented by a separate line in the listing, and the elements are wired according to the layout. To run a simulation, we must add power supply values, input voltages, and modeling information. As an example, suppose that we extract the netlist from an inverter layout and obtain the following listing:

```
M1 15 17 20 20 NFET W=5U L=0.5U
M2 15 17 12 12 PFET W=10U L=0.5U
```

This identifies the two transistors using arbitrary device and node numbering. In the listing, M1 is an nFET while M2 is a pFET. Since the MOSFET node order is Drain-Gate-Source-Bulk, the input to the inverter is the common gate node 17, while the inverter output is taken from the drain node 15. Node 20 must be grounded, while node 12 is the power supply. Some of the more powerful extractors would also provide drain and source dimensions for the junction capacitance calculations in the form

```
M1 15 17 20 20 NFET W=5U L=0.5U AD=12.5P PD=15U AS=20P PS=18U
M2 15 17 12 12 PFET W=10U L=0.5U AD=25P PD=25U AS=40P PS=36U
```

If the extractor does not find the area and perimeter of the drain and source, it must be added by hand.

To run a full simulation, we will add elements to give the following listing:

```
NOT SIMULATION
VDD 12 0 5V
```

```
M1 15 17 20 20 NMOS W=5U L=0.5U AD=12.5P PD=15U AS=20P PS=18U
M2 15 17 12 12 PMOS W=10U L=0.5U AD=25P PD=25U AS=40P PS=36U
RGND 20 0 1U
CLOAD 15 0 100F
.MODEL NFET NMOS <parameter listing ... >
.MODEL PFET PMOS <parameter listing ... >
...
```

where the first line is the name of the circuit and CLOAD has been selected as a 100 fF external load capacitor. RGND is a 1 μΩ resistor to pull node 20 to ground; alternately, we could renumber the netlist or the layout editor may allow it to be defined in the layout before the extraction.[7]

The input voltage at node 17 allows us to model more realistic waveforms. One useful SPICE construct is the PULSE waveform shown in Figure 7.40. It is specified by a statement of the form

```
VIN 17 0 PULSE(V1 V2 TD TR TF PW PER)
```

where V1 and V2 are the start and final voltages, TD is the time delay before the transition starts, TR is the rise time, TF is the fall time, PER and is the period before the waveform repeats itself. This allows us to calculate low-to-high and high-to-low transition times that are more accurate than those found using step-like inputs. Another useful waveform is the exponential source EXP that is specified by a listing of the form

```
VIN_EXP 17 0 EXP (V1 V2 TD1 TAU1 TD2 TAU2)
```

where TD1 and TAU1 are the time delay and time constant for the V1-to-V2 transition, while TD2 and TAU2 are for the opposite case. In both cases, the time values need to be carefully chosen to represent a simulation that provides information on the transient response by displaying the changes

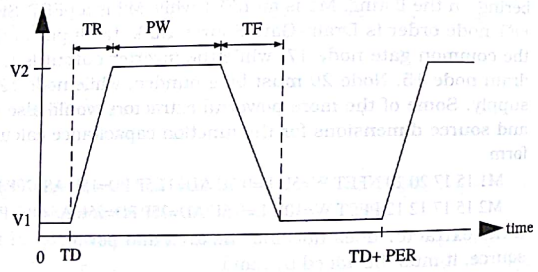


Figure 7.40 SPICE PULSE waveform

[7] Recall that the ground node in SPICE must be numbered as node 0.

in the output as smooth functions of time. These can be estimated from the RC model.

The voltage transfer curve is obtained by a DC sweep initiated by the dot command

```
.DC VIN 0 VDD VSTEP
```

which starts at VIN = 0 and increments by VSTEP to a final value of VDD. The transient response is calculated by

```
.TRAN TSTEP TSTOP
```

This starts at time 0 and increments by time units of TSTEP until the time TSTOP is reached. These two commands provide the most critical operating characteristics of the circuit discussed in this chapter.

The same techniques can be applied to modeling any CMOS circuit. One fine point that sometimes causes confusion is where a common active (n+ or p+) region is shared by adjacent gates. The designation of drain or source is arbitrary, and the total area and perimeter can be split between the two FETs as desired. Care must be taken to insure that the total area and the total perimeter length specified for the two transistors do not exceed the actual layout.

Example 7.5

Consider the two FETs in Figure 7.41. The shared region has a total area of (10)(8) = 80, and a total perimeter of 2(10+8) = 36. M1 uses this as a source region while M2 declares it to be a drain. The split could be listed by writing

```
M1 ... AS=40P PS=18U
M2 ... AD=40P PD=18U
```

which is an equal division. Another choice would be

```
M1 ... AS=10P PS=4.5U
M2 ... AD=70P PD=31.5U
```

which would work equally well.

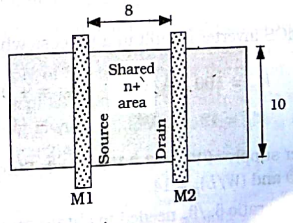


Figure 7.41 Shared active region

More tips and tricks of SPICE modeling of CMOS circuits can be found in the references. As with learning any code, experience is the best teacher.

7.10 References for Further Study

[1] R. Jacob Baker, Harry W. Li, and David E. Boyce, **CMOS Circuit Design, Layout, and Simulation**, IEEE Press, Piscataway, NJ, 1988.

[2] Abdellatif Bellaouar and Mohamed I. Elmasry, **Low-Power Digital VLSI Design**, Kluwer Academic Publishers, Norwell, MA, 1995.

[3] Yuhua Cheng and Chemning Hu, **MOSFET Modeling & BSIM3 User's Guide**, Kluwer Academic Publishers, Norwell, MA, 1999.

[4] Tor A. Fjeldly, Trond Ytterdal, and Michael Shur, **Introduction to Device Modeling and Circuit Simulation**, John Wiley & Sons, New York, 1998.

[5] Ken Martin, **Digital Integrated Circuit Design**, Oxford University Press, New York, 2000.

[6] Jan Rabaey, **Digital Integrated Circuits**, Prentice Hall, Upper Saddle River, NJ, 1996.

[7] Michael Reed and Ron Rohrer, **Applied Introductory Circuit Analysis**, Prentice Hall, Upper Saddle River, NJ, 1999.

[8] Kaushik Roy and Sharat C. Prasad, **Low-Power CMOS VLSI Circuit Design**, Wiley-Interscience, New York, 2000.

[9] Michael John Sebastian Smith, **Application-Specific Integrated Circuits**, Addison-Wesley, Reading, MA, 1997.

[10] John P. Uyemura, **CMOS Logic Circuit Design**, Kluwer Academic Publishers, Norwell, MA, 1999.

[11] Andrei Vladimirescu, **The SPICE Book**, John Wiley & Sons, New York, 1994.

[12] Gary K. Yeap, **Practical Low Power Digital VLSI Design**, Kluwer Academic Publishers, Norwell, MA, 1998.

7.11 Problems

[7.1] A CMOS inverter is built in a process where

$$k'_n = 100\ \mu A/V^2 \qquad V_{Tn} = +0.70\ V$$
$$k'_p = 42\ \mu A/V^2 \qquad V_{Tp} = -0.80\ V \tag{7.201}$$

and a power supply of $V_{DD} = 3.3$ V is used. Find the midpoint voltage V_M if $(W/L)_n = 10$ and $(W/L)_p = 14$.

[7.2] Find the ratio β_n/β_p needed to obtain an inverter midpoint voltage of $V_M = 1.3$ V with a power supply of 3 V. Assume that $V_{Tn} = 0.6$ V and $V_{Tp} = -0.82$ V. What would be the relative device sizes if $k'_n = 110\ \mu A/V^2$ and the mobility values are related by $\mu_n = 2.2\ \mu_p$?

[7.3] An inverter uses FETs with $\beta'_n = 2.1$ mA/V^2 and $\beta_p = 1.8$ mA/V^2. The threshold voltages are given as $V_{Tn} = 0.60$ V and $V_{Tp} = -0.70$ V and the power supply has a value of $V_{DD} = 5$ V. The parasitic FET capacitance at the output node is estimated to be $C_{FET} = 74$ fF.

(a) Find the midpoint voltage V_M.

(b) Find the values of R_n and R_n.

(c) Calculate the rise and fall times at the output when $C_L = 0$.

(d) Calculate the rise and fall times when an external load of value $C_L = 115$ fF is connected to the output.

(e) Plot t_r and t_f as functions of C_L.

[7.4] Find the midpoint voltage for the inverter layout shown in Figure 7.11.

[7.5] Consider the NOT gate shown in Figure 7.11 when an external load of $C_L = 80$ fF is connected to the output. Note that the electrical channel length is $L = 0.8$ μm.

(a) Find the input capacitance of the circuit.

(b) Find the values of R_n and R_p.

(c) Calculate rise and fall times for the inverter.

[7.6] Simulate the circuit in Figure 7.11 using SPICE. Perform both a DC and a transient simulation assuming an external load of $C_L = 100$ fF.

[7.7] A CMOS NAND2 is designed using identical nFETs with a value of $\beta_n = 2\beta_p$; the pFETs are the same size. The power supply is chosen to be $V_{DD} = 5$ V, and the device threshold voltages are given as $V_{Tn} = 0.60$ V and $V_{Tp} = -0.70$ V.

(a) Find the midpoint voltage V_M for the case of simultaneous switching.

(b) What would be the midpoint voltage for an inverter made with the same β-specification?

[7.8] A CMOS NOR2 gate is designed using nFETs with a value of β_n. The pFETs are both described by $\beta_p = 2.2\beta_n$. Find the value of V_M for the case of simultaneous switching if $V_{DD} = 3.3$ V, $V_{Tn} = 0.65$ V, and $V_{Tp} = -0.80$ V.

[7.9] A NAND3 gate uses identical nFETs with an aspect ratio of 4. The nFET process transconductance is 120 μA/V^2, and the threshold voltage is 0.55 V. A power supply of 5 V is chosen for the circuit.

Find the value of the pFET β_p needed to create a gate where the case of simultaneous switching gives a midpoint voltage of $V_M = 2.4$ V. Assume that $V_{Tp} = -0.90$ V and $r = 2.4$.

[7.10] Consider the nFET chain shown in Figure P7.1. This represents a portion of a NAND3 gate. The output capacitance has a value of $C_{out} = 130$ fF, while the internal values are $C_1 = 36$ fF and $C_2 = 36$ fF. The transistors are identical with $\beta_n = 2.0$ mA/V^2 in a process where $V_{DD} = 3.3$ V and $V_{Tn} = 0.70$ V.

(a) Find the discharge time constant for $C_{out} = 130$ fF using the Elmore

Designing High-Speed CMOS Logic Networks

8

Modern CMOS technology is capable of fabricating MOSFETs with channel lengths smaller than 0.1 μm. The channel width W of a FET establishes the aspect ratio (W/L) that is the critical parameter in determining the electrical characteristics of a logic circuit.

Systems designers must take a global view where the logic and architectural features are the first order of business, and the circuits are chosen to implement the necessary functions. In VLSI, however, the ability to meet system timing targets is intimately related to the switching speed of the logic circuits. If the timing specifications cannot be met by the circuitry, then we may be forced to modify the logic.

In this chapter we will initiate our study of high-speed system design and learn techniques to select transistor sizes. These methods are useful for designing both library collections and custom designs. The techniques presented in this chapter are an integral part of high-speed VLSI design, and are heavily oriented toward electronics. Owing to the specialized nature of the material, some readers may prefer to skip this and the following chapter in a first reading, and refer back to them as needed.

8.1 Gate Delays

In the previous chapter we found that the output switching times of the CMOS logic gate in Figure 8.1 are described by the linear expressions

$$t_r = t_{r0} + \alpha_p C_L \qquad (8.1)$$
$$t_f = t_{f0} + \alpha_n C_L$$

where C_L is the external load capacitance. Given the layout geometry and processing parameters, the equation set allows us to analyze the switching performance of an arbitrary gate. VLSI designers are faced with the

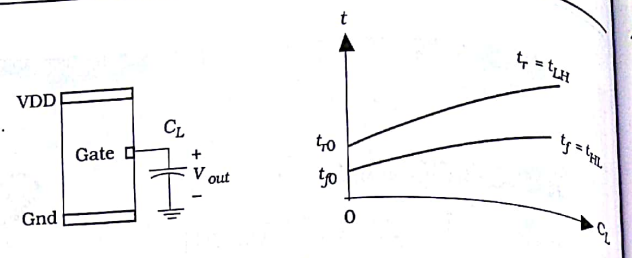


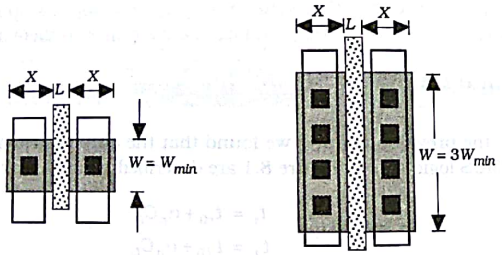
Figure 8.1 Output switching times

opposite problem. It is their responsibility to choose the logic cascades and then specify aspect ratios for every transistor. The system timing specifications must be met while working within a limited real estate allocation. This provides motivation for developing a structured approach to estimating logic delays in CMOS gates.

Let us examine an approach that uses the minimum-size MOSFET as a basis. The layout geometry is shown in Figure 8.2(a). The drawn aspect ratio (W/L) and the active dimension X are determined by the design rules. Once these are known, we can define the parasitics for the device and use them for reference. Let us denote **unit** FET parameters with the subscript 'u' such that the transistor resistance is

$$R_u = \frac{1}{k'\left(\frac{W}{L}\right)_u (V_{DD} - V_T)} \tag{8.2}$$

while



(a) Minimum-size (b) 3X scaled FET

Figure 8.2 Unit transistor reference

$$C_{Gu} = C_{ox}(WL)_u$$
$$C_{Du} = (C_{GD} + C_{DB})_u$$
$$C_{Su} = (C_{GS} + C_{SB})_u \tag{8.3}$$

give the capacitance values. These are assumed to be known parameters in the analysis. To create a design methodology, we will specify that all transistor sizes are integer multiples of the minimum width $W_{min} = W_u$. An example is the $m = 3$ FET shown in Figure 8.2(b). In general, this gives

$$\left(\frac{W}{L}\right)_m = m\left(\frac{W}{L}\right)_u \tag{8.4}$$

with $m = 1, 2, 3, \ldots$ as the size specifier. The resistance and gate capacitance of the m-sized FET are written in terms of the unit transistor as

$$R_m = \frac{R_u}{m}$$
$$C_{Gm} = mC_{Gu} \tag{8.5}$$

We will scale the FET so that X is the same as for the unit FET. For arbitrary m, this implies that the drain and source capacitances scale approximately as

$$C_{Dm} \approx mC_{Du}$$
$$C_{Sm} \approx mC_{Su} \tag{8.6}$$

These will be used as equalities in our treatment. Combining with the resistance formula gives the result

$$R_m C_m = R_u C_u = \text{constant} \tag{8.7}$$

which is very useful in scaling theory.

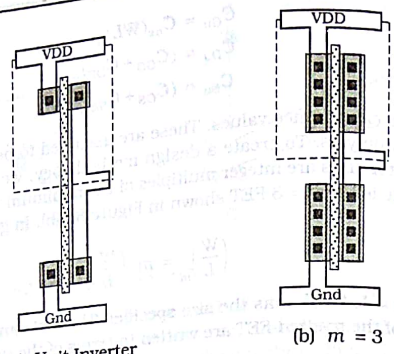
Now suppose that we design an inverter using the minimum-size geometry for both the nFET and the pFET. This results in the layout shown in Figure 8.3(a); note that $\beta_n > \beta_p$ for this design. The rise time for this circuit is controlled by the pFET and can be expressed as

$$t_{ru} = t_{r0} + \alpha_{pu} C_L \tag{8.8}$$

The fall time

$$t_{fu} = t_{f0} + \alpha_{nu} C_L \tag{8.9}$$

is governed by the nFET parameters. Since $R_p > R_n$, $t_{r0} > t_{f0}$, and $\alpha_{pu} > \alpha_{nu}$ so, for a given load C_L, $t_{ru} > t_{fu}$. The midpoint voltage is



(a) Unit Inverter (b) $m = 3$

Figure 8.3 Inverter designs using scaled transistors

$$V_M = \frac{V_{DD} - |V_{Tp}| + \sqrt{r}V_{Tn}}{1 + \sqrt{r}} \tag{8.10}$$

where $r = (\mu_n / \mu_p)$ is the mobility ratio. The input capacitance is a minimum value for a complementary pair

$$C_{in} = 2C_u = C_{min} \tag{8.11}$$

since both transistors are minimum-size devices.

If we scale the FETs by a factor $m = 3$, then we arrive at the layout in Figure 8.3(b). This does not change the midpoint voltage, but does alter the switching times. To find the response of the new circuit, first note that the zero-load times t_{r0} and t_{f0} are (approximately) constants as demonstrated by equation (8.7). The slope parameter α decreases as $(1/m)$ because of the decrease in resistance by the same factor. Thus,

$$t_{r3} = t_{r0} + \frac{\alpha_{pu}}{3} C_L$$
$$t_{f3} = t_{f0} + \frac{\alpha_{nu}}{3} C_L \tag{8.12}$$

describes the scaled circuit. The input capacitance for this gate is

$$C_{in} = 3C_{min} \tag{8.13}$$

Consider next the NAND2 gate in Figure 8.4(a) that uses minimum-size transistors. The switching equations must be modified for this circuit. First, recall that the zero-load times t_{r0} and t_{f0} are proportional to the product of C_{FET} and the resistance. In the inverter, two FETs contribute to the capacitance. Since there are now three FETs that touch the output node, we introduce a factor of (3/2) multiplying the internal capacitance.[1] The resistances scale in a different manner. The pFET resistance R_p is the same as that for an inverter, while the nFET resistance R_n between the output node and ground is doubled because of the series connection; this increases both t_{f0} and α_{nu} by a factor of 2. Including these multipliers in the equation gives

$$t_r = \left(\frac{3}{2}\right) t_{r0} + \alpha_{pu} C_L \quad \text{(Unit NAND2)}$$
$$t_f = 3t_{f0} + 2\alpha_{nu} C_L \tag{8.14}$$

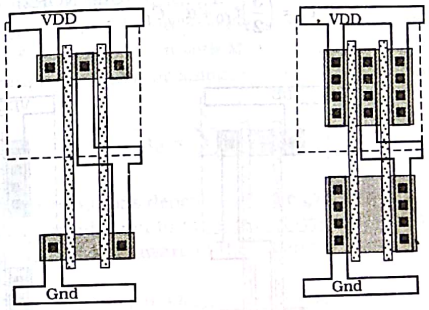
This ignores the capacitance between the series-connected nFETs, but does illustrate the trends. The input capacitance is

$$C_{in} = C_{min} \tag{8.15}$$

since an nFET/pFET pair consists of minimum-size devices.

If we scale the transistors with $m = 3$, as in Figure 8.4(b), then the equations must be modified. Both α factors are reduced by $(1/m)$ because of the decrease in resistance. The decrease in resistance counteracts the increase in C_{FET}, so that the zero-load terms are unchanged. Thus,

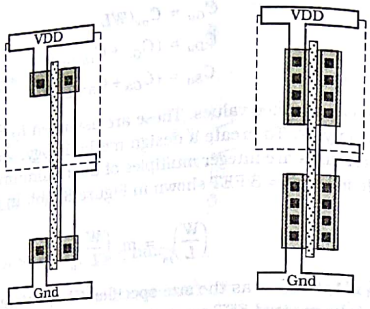
$$t_r = \left(\frac{3}{2}\right) t_{r0} + \frac{\alpha_{pu}}{3} C_L \tag{8.16}$$



(a) Unit transistors (b) $m = 3$ circuit

Figure 8.4 NAND2 gate scaling

[1] This assumes that the nFET and pFET capacitances are equal, which is not true even if they are the same size.



(a) Unit Inverter (b) $m = 3$

Figure 8.3 Inverter designs using scaled transistors

$$V_M = \frac{V_{DD} - |V_{Tp}| + \sqrt{r}V_{Tn}}{1 + \sqrt{r}} \tag{8.10}$$

where $r = (\mu_n / \mu_p)$ is the mobility ratio. The input capacitance is a minimum value for a complementary pair

$$C_{in} = 2C_u = C_{min} \tag{8.11}$$

since both transistors are minimum-size devices.

If we scale the FETs by a factor $m = 3$, then we arrive at the layout in Figure 8.3(b). This does not change the midpoint voltage, but does alter the switching times. To find the response of the new circuit, first note that the zero-load times t_{r0} and t_{f0} are (approximately) constants as demonstrated by equation (8.7). The slope parameter α decreases as $(1/m)$ because of the decrease in resistance by the same factor. Thus,

$$\begin{aligned} t_{r3} &= t_{r0} + \frac{\alpha_{pu}}{3} C_L \\ t_{f3} &= t_{f0} + \frac{\alpha_{nu}}{3} C_L \end{aligned} \tag{8.12}$$

describes the scaled circuit. The input capacitance for this gate is

$$C_{in} = 3C_{min} \tag{8.13}$$

Consider next the NAND2 gate in Figure 8.4(a) that uses minimum-size transistors. The switching equations must be modified for this circuit. First, recall that the zero-load times t_{r0} and t_{f0} are proportional to the product of C_{FET} and the resistance. In the inverter, two FETs contribute to the capacitance. Since there are now three FETs that touch the output node, we introduce a factor of (3/2) multiplying the internal capacitance.[1] The resistances scale in a different manner. The pFET resistance R_p is the same as that for an inverter, while the nFET resistance R_n between the output node and ground is doubled because of the series connection; this increases both t_{f0} and α_{nu} by a factor of 2. Including these multipliers in the equation gives

$$\begin{aligned} t_r &= \left(\frac{3}{2}\right) t_{r0} + \alpha_{pu} C_L \quad \text{(Unit NAND2)} \\ t_f &= 3t_{f0} + 2\alpha_{nu} C_L \end{aligned} \tag{8.14}$$

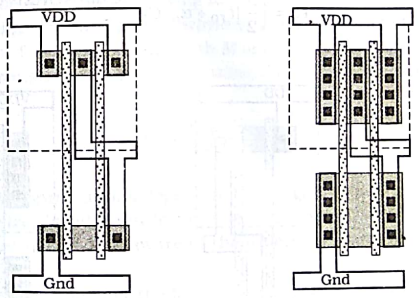
This ignores the capacitance between the series-connected nFETs, but does illustrate the trends. The input capacitance is

$$C_{in} = C_{min} \tag{8.15}$$

since an nFET/pFET pair consists of minimum-size devices.

If we scale the transistors with $m = 3$, as in Figure 8.4(b), then the equations must be modified. Both α factors are reduced by $(1/m)$ because of the decrease in resistance. The decrease in resistance counteracts the increase in C_{FET}, so that the zero-load terms are unchanged. Thus,

$$t_r = \left(\frac{3}{2}\right) t_{r0} + \frac{\alpha_{pu}}{3} C_L \tag{8.16}$$



(a) Unit transistors (b) $m = 3$ circuit

Figure 8.4 NAND2 gate scaling

[1] This assumes that the nFET and pFET capacitances are equal, which is not true even if they are the same size.

and

$$t_f = 3t_{f0} + \frac{2\alpha_{nu}}{3}C_L \tag{8.17}$$

provides the scaled response times. The input capacitance is

$$C_{in} = 3C_{min} \tag{8.18}$$

If N is the fan-in (number of inputs), then we may extrapolate the analysis to write

$$t_r = \left(\frac{N+1}{2}\right)t_{r0} + \frac{\alpha_{pu}}{m}C_L \quad \text{(NAND-}N\text{)}$$
$$t_f = (N+1)t_{f0} + \frac{N\alpha_{nu}}{m}C_L \tag{8.19}$$

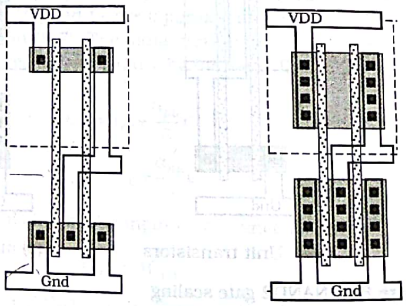
for an N-input NAND gate that uses m-sized FETs. In this case

$$C_{in} = mC_{min} \tag{8.20}$$

gives the input capacitance.

A NOR2 gate can be analyzed using the same techniques. The unit-transistor layout in Figure 8.5(a) has switching times that can be approximated by

$$t_r = 3t_{r0} + 2\alpha_{pu}C_L$$
$$t_f = \left(\frac{3}{2}\right)t_{f0} + \alpha_{nu}C_L \quad \text{(Unit NOR2)} \tag{8.21}$$



(a) Unit transistors (b) $m = 3$ circuit

Figure 8.5 NOR gate scaling

The $m = 3$ scaled circuit in Figure 8.5(b) modifies the expressions to

$$t_r = 3t_{r0} + \frac{2\alpha_{pu}}{3}C_L$$
$$t_f = \left(\frac{3}{2}\right)t_{f0} + \frac{\alpha_{nu}}{3}C_L \tag{8.22}$$

because of the decrease in the slope parameters α. For N inputs and general scaling factor m, these may be extended to

$$t_r = (N+1)t_{r0} + \frac{N\alpha_{pu}}{m}C_L$$
$$t_f = \left(\frac{N+1}{2}\right)t_{f0} + \frac{\alpha_{nu}}{m}C_L \quad \text{(NOR-}N\text{)} \tag{8.23}$$

for an N-input NOR gate. Also,

$$C_{in} = mC_{min} \tag{8.24}$$

gives the input capacitance.

These equations clearly demonstrate the dependence of the switching times and input capacitance on

- Number of inputs N (fan-in)
- Transistor scaling factor m

The input capacitance is important because it is a measure of how much a gate loads the stage that is driving it.

This technique of gate design provides a structured approach for estimating delays. For a logic chain with M stages, we may approximate the total delay through the chain by summing the individual delays:

$$t_d = \sum_{i=1}^{M} t_i \tag{8.25}$$

The individual contributions depend upon the gate type (i.e., NOT, NAND, etc.) and its size, in addition to the size and type of the next gate in the chain. We also need to be aware of the difference between rise and fall times.

As an example, consider the logic chain in Figure 8.6 where the input is originally at 0 and then makes a transition to a 1. The stages are scaled with increasing values of m, and the output is a capacitor with a value of $C = 4\ C_{min}$. The total delay is

$$t_d = t_{NOT}\big|_{m=1} + t_{NAND2}\big|_{m=2} + t_{NOR2}\big|_{m=3} \tag{8.26}$$

where the first and third terms represent fall times, while the second term

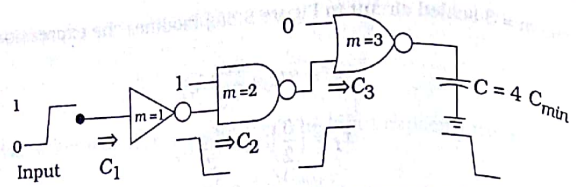


Figure 8.6 Delay time example

is a rise time. Applying the equations above gives the terms as

$$\begin{aligned} t_{NOT}|_{m=1} &= t_{f0} + \alpha_{nu} 2C_{min} \\ t_{NAND2}|_{m=2} &= \left(\frac{3}{2}\right) t_{r0} + \frac{\alpha_{pu}}{2} 3C_{min} \\ t_{NOR2}|_{m=3} &= \left(\frac{3}{2}\right) t_{f0} + \frac{\alpha_{nu}}{3} 4C_{min} \end{aligned} \tag{8.27}$$

so that the total chain delay is

$$\begin{aligned} t_d &= \left(\frac{5}{2}\right) t_{f0} + \left(\frac{10}{3}\right) \alpha_{nu} C_{min} + \left(\frac{3}{2}\right) t_{r0} + \left(\frac{3}{2}\right) \alpha_{pu} C_{min} \\ &= \frac{1}{2}(5t_{f0} + 3t_{r0}) + \left[\left(\frac{10}{3}\right)\alpha_{nu} + \left(\frac{3}{2}\right)\alpha_{pu}\right] C_{min} \end{aligned} \tag{8.28}$$

It is important to note that the expression for t_d will change if different inputs are applied. Overall, the technique allows us to estimate delays through logic cascades in a uniform manner.

Although the analysis has been performed using minimum-size transistors for both the nFET and pFET, it is straightforward to modify the analysis for a symmetrical design with $\beta_n = \beta_p$. In this case, the inverter rise and fall times are equal and given by

$$t_s = t_0 + \alpha C_L \tag{8.29}$$

for a circuit with $W_n = W_{min}$ and $W_p = rW_{min}$. The input capacitance is increased to

$$\begin{aligned} C_{in} &= C_u(1 + r) \\ &= C_{inv} \end{aligned} \tag{8.30}$$

which now becomes the reference. Scaling the transistors in the NOT gate by m gives

$$t_s = t_0 + \frac{\alpha}{m} C_L \tag{8.31}$$

as discussed in the previous chapter. The analysis of multi-input gates such as the NAND and NOR circuits proceeds in the same manner. Note that if m is used to scale both nFETs and pFETs equally, the rise and fall times will be unequal for gates with $N > 1$. Equalization of the switching times can be achieved only if the two FET types are different sizes. If the parallel-connected FETs are increased by m then the series-connected transistors must be increased by a factor mN to obtain a symmetrical design.

Other approaches have been developed to estimate the delay through a logic chain. One simple technique is to use the minimum-size inverter as a basis, and then build up NAND and NOR gates for increasing numbers of inputs N. If the switching delay is plotted as a function of load capacitance C_L, one obtains a trend such as that shown in Figure 8.7. By definition, an inverter is described by the $N = 1$ plot and gives the basis for writing a delay time of

$$t_d = (A + Bn)\tau_{min} \tag{8.32}$$

where A and B are dimensionless constants,

$$\tau_{min} = R_{min} C_{min} \tag{8.33}$$

is the time constant for the minimum size inverter, and

$$n = \frac{C_L}{C_{min}} \tag{8.34}$$

is the number of minimum load factors being driven by the stage. These are taken to be empirically measured quantities, i.e., curve fitting parameters. Alternately, they may be generated by a circuit simulation. If the fan-in is increased to $N = 2$ (for either a NAND2 or a NOR2 gate), then the worst-case delay time has a large zero-load value and a steeper slope. The same comment holds as we increase to $N = 3$. An empirical fit is obtained

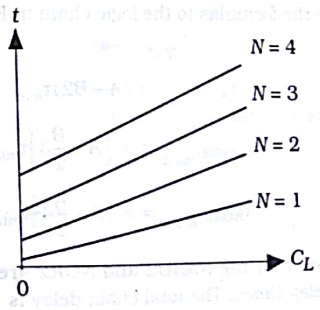


Figure 8.7 Delay times as a function of fan-in N

by multiplying t_d by a factor x_1 that accounts for the increases in the form

$$t_{d,N} = (x_1)^{(N-1)}(A + Bn)\tau_{min} \tag{8.35}$$

For example, if the increase from $N = 1$ to $N = 2$ is 17% per input, this means that $x_1 = 1.17$ and

$$t_{d,N} = (1.17)^{(N-1)}(A + Bn)\tau_{min} \tag{8.36}$$

In practice, an average value of many comparisons would be used. If the transistors are scaled by a factor $m = 1, 2, \ldots$, then we would modify the expression to

$$t^m_{d,N} = (x_1)^{(N-1)}\left(A + \frac{B}{m}n\right)\tau_{min} \tag{8.37}$$

to account for the increased drive strength. Also, for a complex N-input logic gate, the delay would be even larger since the internal circuit capacitances will increase and slow down charging or discharge events. In this case, we multiply by another empirical parameter $x_2 > 1$ to obtain

$$t^m_{d,N} = x_2(x_1)^{(N-1)}\left(A + \frac{B}{m}n\right)\tau_{min} \tag{8.38}$$

In practice, one would expect around a 5 to 20% increase due to additional FET parasitics.

While this approach is approximate in nature, it does reflect the physical fact that the switching times increase with the fan-in. If we apply the delay estimate to gates in a uniform manner, then it allows us to compare the delay through various cascaded arrangements. The actual numerical values are not accurate, but we would expect the *relative* values to have merit.

Example 8.1

Let us apply the formulas to the logic chain in Figure 8.6. The three terms are

$$\begin{aligned} t_{NOT}|_{m=1} &= (A + B2)\tau_{min} \\ t_{NAND2}|_{m=2} &= x_1\left(A + \frac{B}{2}3\right)\tau_{min} \\ t_{NOR2}|_{m=3} &= x_1\left(A + \frac{B}{2}4\right)\tau_{min} \end{aligned} \tag{8.39}$$

where we note that the NAND2 and NOR2 are treated as having equal worst-case delay times. The total chain delay is

$$\frac{t_d}{\tau_{min}} = [x_1 + 1]A + \left[\left(\frac{7}{2}\right)x_1 + 2\right]B \tag{8.40}$$

If $x_1 = 1.17$, then

$$\frac{t_d}{\tau_{min}} = 2.17A + 6.1B \tag{8.41}$$

is the delay compared to a single inverter.

As we will see in later chapters, the ability to estimate the delay through a logic chain is an important skill for high-speed design. In realistic digital systems design, one can usually find distinctly different equations or algorithms that produce the same result. Each, however, will use a different type of logic cascade. Techniques such as these provide a basis for deciding on the design that will be the fastest.

8.2 Driving Large Capacitive Loads

Many of the important points in high-speed design can be obtained from studying the characteristic delay through inverter circuits. The analyses form the basis for several well-known design techniques that can be extended to include arbitrary gates.

Consider the NOT gate in Figure 8.8 where the circuit drives the external load capacitance C_L; the internal parasitic capacitance C_{FET} due to the transistors is not shown explicitly in the drawing. The electrical characteristics are determined by the values of β_n and β_p. A symmetric design with $\beta_n = \beta_p = \beta$ will be used for simplicity. Since $\beta = k'(W/L)$, this means that the aspect ratios are related by

$$\left(\frac{W}{L}\right)_p = r\left(\frac{W}{L}\right)_n \tag{8.42}$$

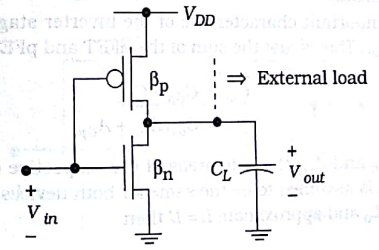


Figure 8.8 CMOS inverter circuit

where r is the mobility ratio

$$r = \frac{\mu_n}{\mu_p} = \frac{k'_n}{k'_p} > 1 \tag{8.43}$$

Assuming equal magnitude threshold voltages $V_{Tn} = |V_{Tp}| = V_T$ gives equal FET resistances of

$$R_n = R_p = R = \frac{1'}{\beta(V_{DD} - V_T)} \tag{8.44}$$

This design yields a VTC with a midpoint voltage of $V_M = (V_{DD}/2)$ and equal rise and fall times. For a 0-to-1 transition at the output, the voltage $V_{out}(t)$ across C_L is of the form

$$V_{out}(t) = V_{DD}[1 - e^{-t/\tau}] \tag{8.45}$$

while a 1-to-0 change is described by

$$V_{out}(t) = V_{DD}e^{-t/\tau} \tag{8.46}$$

In both expressions, the time constant is given by the product

$$\tau = RC_{out} = R(C_{FET} + C_L) \tag{8.47}$$

The generic switching time delay $t_s = t_r = t_f$ is then given in the form

$$t_s = t_0 + \alpha C_L \tag{8.48}$$

where t_0 is the zero-load delay and α is the slope of the t_s vs. C_L plot. The value of t_0 is almost invariant to changes in the circuit, while α is proportional to the resistance R:

$$\alpha \propto R = \frac{1}{\beta(V_{DD} - V_T)} \tag{8.49}$$

The numerical value of β can be chosen to satisfy the transient response requirements.

An important characteristic of the inverter stage is its input capacitance C_{in}. This is just the sum of the nFET and pFET gate capacitances

$$\begin{aligned} C_{in} &= C_{Gn} + C_{Gp} \\ &= C_{ox}(A_{Gn} + A_{Gp}) \end{aligned} \tag{8.50}$$

with A_{Gn} and A_{Gp} the gate areas of the respective devices. The channel length L is assumed to be the same for both devices. If we ignore the gate overlap L_o and approximate $L = L'$ then

$$\begin{aligned} C_{in} &= C_{ox}L(W_n + W_p) \\ &= (1 + r)(C_{ox}LW_n) \\ &= (1 + r)C_{Gn} \end{aligned} \tag{8.51}$$

where we have used equation (8.42) in the second line.

Now suppose that we use the inverter to drive an identical gate as shown in Figure 8.9. In this case, the load C_{L1} seen by gate 1 is

$$C_{L1} = C_{in} \tag{8.52}$$

Since the load capacitance is the same as the gate's own input capacitance, we call this a **unit load** value. The switching time is given by

$$t_{s1} = t_0 + \alpha C_{in} \tag{8.53}$$

which is a convenient reference for analyzing the performance of the gate when it is used to drive other loads.

If the load capacitance is increased to a very large value $C_L >> C_{in}$, then the switching times increase proportionately. To keep t_s small, we may decrease α by using larger transistors to decrease the resistance. Increasing the value of β compensates for the larger load and demonstrates the speed-versus-area trade-off. Suppose that the aspect ratios are increased by the scaling factor $S > 1$. The new device transconductance is

$$\beta' = S\beta \tag{8.54}$$

so that the resistance is reduced to

$$R' = \frac{R}{S} \tag{8.55}$$

The slope is also decreased to a new value of

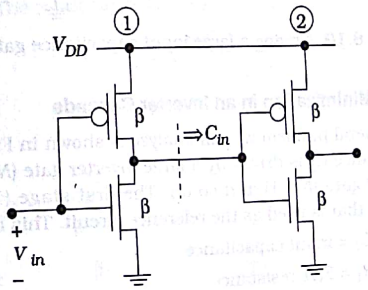


Figure 8.9 Concept of a unit load

$$\alpha' = \frac{\alpha}{S} \tag{8.56}$$

These combine to give the switching time equation for the new inverter as

$$t_s = t_0 + \left(\frac{\alpha}{S}\right) C_L \tag{8.57}$$

The compensation factor $(1/S)$ allows us to drive larger values of C_L. If the load has a value $C_L = SC_{in}$, then the switching time is the same as for a unit load. Scaling the transistor also affects the input capacitance since

$$W_n' = SW_n \tag{8.58}$$

increases the gate area. The new value is given by

$$C_{in}' = SC_{in} \tag{8.59}$$

i.e., it increases by the same scaling factor S. This introduces another problem that is illustrated in Figure 8.10. The large input capacitance acts as the load $C_{L,d} = C_{in}$ to the driving circuit. To compensate for this effect, we must increase the size of the transistor in the driving gate, which increases its value of $C_{in,d}$. This in turn makes it more difficult to drive. Clearly, a methodology would be useful in solving this problem.

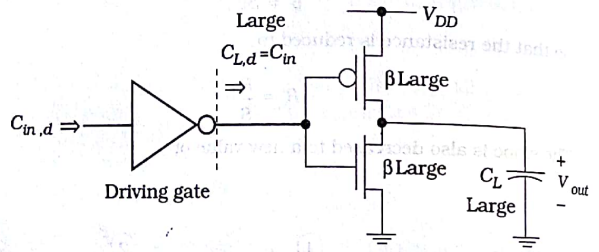


Figure 8.10 Driving a large input capacitance gate

8.2.1 Delay Minimization in an Inverter Cascade

The general problem we will analyze is shown in Figure 8.11. A large load capacitance C_L is driven by a large inverter gate (N), which is driven by a smaller gate (N - 1) and so on. The first stage (1) is a "standard size" inverter that is used as the reference circuit. This has known parameters

C_1 = input capacitance

R_1 = FET resistance

β_1 = device transconductance

The stages are monotonically scaled such that 1 is the smallest and N is the largest:

$$\beta_1 < \beta_2 < \beta_3 < \ldots < \beta_{N-1} < \beta_N \tag{8.60}$$

The sizes of the NOT symbols have been adjusted to show the relative sizing. The simplest scaling is obtained by increasing the size of the transistors by a factor $S > 1$ from one stage to the next. This means that

$$\begin{aligned} \beta_2 &= S\beta_1 \\ \beta_3 &= S\beta_2 \end{aligned} \tag{8.61}$$

and so on. The general expression

$$\beta_{j+1} = S\beta_j \tag{8.62}$$

relates the j-th and $(j+1)$-st stages.

The main problem is as follows. A signal is placed at the input to inverter 1. We want to find the number of states N and the scaling factor S that will minimize the time needed for the signal to reach the load C_L. This can be solved by first studying the characteristics of a typical stage, and then applying the results to the chain.

First note that using β_1 as the reference value in the scaling implies that

$$\begin{aligned} \beta_2 &= S\beta_1 \\ \beta_3 &= S\beta_2 = S^2\beta_1 \\ \beta_4 &= S\beta_3 = S^3\beta_1 \end{aligned} \tag{8.63}$$

or, in general,

$$\beta_j = S^{(j-1)}\beta_1 \tag{8.64}$$

for $j = 2$ to N. The input capacitance scales with β_j, so that

$$C_j = S^{(J-1)}C_1 \tag{8.65}$$

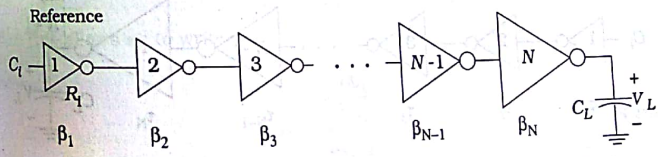


Figure 8.11 Inverter chain analysis

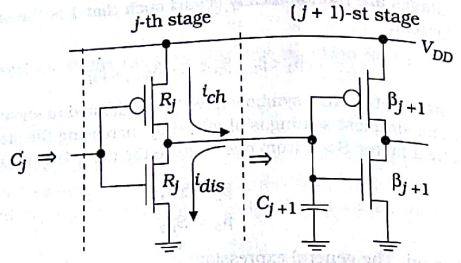


Figure 8.12 Characteristics of a typical stage in the chain

is the value into the j-th stage. The FET resistance scales as $(1/\beta_j)$, leading us to write

$$R_j = \frac{R_1}{S^{(j-1)}} \tag{8.66}$$

as the j-th stage resistance.

Now let us calculate the behavior of a typical stage. Figure 8.12 shows the j-th and $(j+1)$-st stages in the chain. The charging current i_{ch} and discharging current i_{dis} for the j-th stage are shown in the drawing. If we make the simplifying assumption that the load capacitance $C_{j+1} >> C_{FET,j}$ then the time constant for the j-th stage is

$$\tau_j = R_j C_{j+1} \tag{8.67}$$

for $j = 1$ to N. This result can be used to analyze the total delay through the chain. Figure 8.13 shows the time constants for each stage. The total time constant for the chain can be computed by just summing each term to give

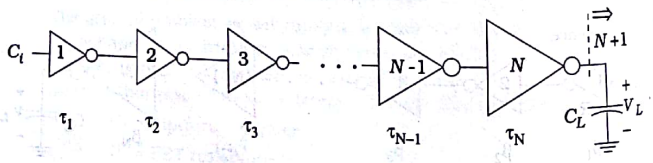


Figure 8.13 Time constants in the cascade

$$\begin{aligned}\tau_d &= \tau_1 + \tau_2 + \tau_3 + \ldots + \tau_{N-1} + \tau_N \\ &= R_1C_2 + R_2C_3 + R_3C_4 + \ldots + R_{N-1}C_N + R_NC_L\end{aligned} \tag{8.68}$$

where we have used the N-stage load of C_L by assigning it the symbol

$$\begin{aligned}C_L &= C_{N+1} \\ &= S^N C_1\end{aligned} \tag{8.69}$$

The second step is added to maintain consistency with the numbering scheme. Substituting the scaling relations in equations (8.65) and (8.66), this reduces to

$$\tau_d = R_1SC_1 + \frac{R_1}{S}S^2C_1 + \frac{R_1}{S^2}S^3C_1 + \ldots + \frac{R_1}{S^{N-2}}S^{N-1}C_1 + \frac{R_1}{S^{N-1}}S^NC_1 \tag{8.70}$$

Simplifying gives

$$\begin{aligned}\tau_d &= SR_1C_1 + SR_1C_1 + SR_1C_1 + \ldots + SR_1C_1 + SR_1C_1 \\ &= N(SR_1C_1)\end{aligned} \tag{8.71}$$

since every term is identical. The total delay is thus given by

$$\tau_d = NS\tau_r \tag{8.72}$$

where $\tau_r = R_1C_1$ is a reference time constant. This is a very important result to remember; qualitatively, it tells us to equalize the signal delay through each stage.

Now let us turn to the problem of minimizing the delay. The unknowns are N and S, so we need two equations. One is equation (8.72) for the total delay. The other can be obtained from equation (8.69) which is a boundary condition for the problem. Starting with the form

$$C_L = S^N C_1 \tag{8.73}$$

we divide by C_1 and take the natural logarithm of both sides to arrive at the form

$$\ln(S^N) = \ln\left(\frac{C_L}{C_1}\right) = N\ln(S) \tag{8.74}$$

This allows us to write

$$N = \frac{\ln\left(\frac{C_L}{C_1}\right)}{\ln(S)} \tag{8.75}$$

as the second equation. Substituting into equation (8.72) then gives the delay time constant in the form

$$\tau_d = \tau_r \ln\left(\frac{C_L}{C_1}\right)\left[\frac{S}{\ln(S)}\right] \tag{8.76}$$

which is a function only of S. To minimize τ_d we apply the derivative condition

$$\frac{\partial \tau_d}{\partial S} = \frac{\partial}{\partial S}\left[\frac{S}{\ln(S)}\right] = 0 \tag{8.77}$$

Differentiating,

$$\frac{1}{\ln(S)} - \frac{S}{S[\ln(S)]^2} = 0 \tag{8.78}$$

or,

$$\ln(S) = 1 \tag{8.79}$$

This is a very interesting expression since the solution is

$$S = e \tag{8.80}$$

i.e., the Euler $e = 2.71...$ is the scaling factor for a minimum delay chain. The number of stages for this design is

$$N = \frac{\ln\left(\frac{C_L}{C_1}\right)}{\ln(S)} = \ln\left(\frac{C_L}{C_1}\right) \tag{8.81}$$

The total delay through the chain is

$$\tau_d = e \ln\left(\frac{C_L}{C_1}\right)\tau_r \tag{8.82}$$

which is the minimum value, and completes the solution to the problem.

Example 8.2

To see how the results may be applied, suppose that we want to drive a load capacitor of value $C_L = 10$ pF (where 1 pF = 10^{-12} F). The input stage is defined with $C_1 = 20$ fF = 20×10^{-15} F and has $\beta_1 = 200\ \mu A/V^2$. The number of stages N needed to minimize the delay is calculated as

$$N = \ln\left(\frac{10\times10^{-12}}{20\times10^{-15}}\right) = \ln(500) \tag{8.83}$$

Since $\ln(500) \approx 6.21$, we will select $N = 6$ to obtain a non-inverting chain. The results gave us a scaling factor of $S = e$ if the N equation is exact. However, since we have rounded N to a (useful) integer value, the scaling factor is more correctly given by rearranging equation (8.73) to the form

$$S = \left(\frac{C_L}{C_1}\right)^{1/N} \tag{8.84}$$

For our example, this gives

$$S = (500)^{1/6} = 2.82 \tag{8.85}$$

which is slightly larger than the ideal value.

The design consists of 6 inverters with device transconductances of

$$\begin{aligned} \beta_2 &= (2.82)\beta_1 \\ \beta_3 &= (2.82^2)\beta_1 = (8)\beta_1 \\ \beta_4 &= (2.82^3)\beta_1 = (22)\beta_1 \\ \beta_5 &= (2.82^4)\beta_1 = (63)\beta_1 \\ \beta_6 &= (2.82^5)\beta_1 = (178)\beta_1 \end{aligned} \tag{8.86}$$

where we have rounded to the nearest integer. Note how rapidly the FET sizes increase when approaching the output stage.

The idealized calculation above tends to underestimate the scaling factor S because the analysis ignored the presence of the parasitic FET capacitance. In practice, $S > e$ and the value depends on the processing. To see the origin of the increase, let us redo the calculation with the parasitic transistor capacitances included.

Figure 8.14 shows the j-th stage circuit with the parasitic FET capacitance C_{Fj} included at the output. The time constant for this stage is now given by the time constant

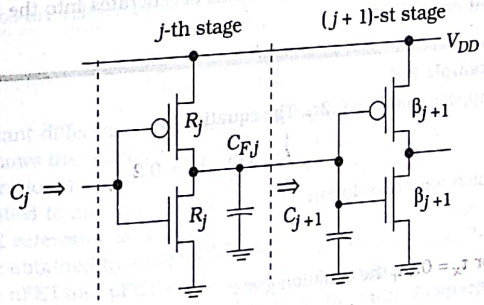


Figure 8.14 Driver chain with internal FET capacitance

$$\tau_J = R_J(C_{F,J} + C_{J+1}) \tag{8.87}$$

since the transistors must drive both C_{FJ} and C_{J+1}. Parasitic FET capacitance is proportional to the width of the transistor, so that the scaling relation is

$$C_{F,J} = S^{(J-1)}C_{F,1} \tag{8.88}$$

where $C_{F,1}$ is the capacitance of the stage 1 FETs. With this, the delay time constant for the entire chain is

$$\tau_d = R_1(C_{F,1} + C_2) + R_2(C_{F,2} + C_3) + \ldots + R_N(C_{F,N} + C_L) \tag{8.89}$$

Using the scaling relations shows that each stage has a parasitic term of $R_1C_{F,1}$ so that the total delay is

$$\tau_d = NR_1C_{F,1} + N(SR_1C_1) \tag{8.90}$$

Using equation (8.75) for N gives the form

$$\tau_d = \left[\frac{\tau_x}{\ln(S)} + \tau_r\left(\frac{S}{\ln(S)}\right)\right]\ln\left(\frac{C_L}{C_1}\right) \tag{8.91}$$

where

$$\tau_x = R_1C_{F,1} \tag{8.92}$$

Differentiating with respect to S and setting the result to 0 gives the minimization condition in the form

$$S[\ln(S) - 1] = \frac{\tau_x}{\tau_r} \tag{8.93}$$

which is a transcendental equation whose solution depends on the ratio of τ_x to τ_r. Note that for $\tau_x = 0$ this degenerates into the simpler equation that gives $S = e$.

Example 8.3

Suppose that $\tau_x = 0.2\tau_r$. The equation is

$$S[\ln(S) - 1] = 0.2 \tag{8.94}$$

which has the solution

$$S \approx 2.91 > e \tag{8.95}$$

For $\tau_x = 0.5\tau_r$ the equation gives

$$S \approx 3.18 \tag{8.96}$$

Finally, for $\tau_x = \tau_r$ we find

$$S \approx 3.59 \tag{8.97}$$

This illustrates the dependence of the scaling factor on the parasitics.

It is important to remember that the algorithm minimizes the time delay from the input to the output, and often specifies transistor sizes that are too large to be practical. This is especially true if we increase the scaling factor to account for the parasitics while attempting to design around a very large output capacitance.

8.3 Logical Effort

The scaling of logic cascades has been a mainstay technique since the beginnings of digital MOS/VLSI circuits. It serves as a guide for designing fast logic chains, and provides many qualitative features that are applied in everyday circuits.

Sutherland et al. have reformulated the ideas contained in the scaling analysis and used them to develop a generalized technique called **Logical Effort**. Logical Effort characterizes gates and how they interact in logic cascades, and provides techniques to minimize the delay. It allows the theory to be extended to include standard logic gates such as NAND and NOR, in addition to complex logic gate circuits. In this section we will examine the basics of the approach to learn how it can be used to design high-speed chains. The interested reader is directed to Reference [8] for a complete and well-written treatment of this useful technique.

8.3.1 Basic Definitions

The starting point is to define an inverter as a reference gate. The simplest approach is to use a symmetric NOT gate where $\beta_n = \beta_p$ and the device aspect ratios are related by

$$\left(\frac{W}{L}\right)_p = r\left(\frac{W}{L}\right)_n \tag{8.98}$$

The important difference between the two FETs is the value of $r > 1$. Figure 8.15 shows the reference circuit for a 1X design. The relative values of the aspect ratios (1 and r) are included next to the transistors. The circuit can be applied to any value of $(W/L)_n$ that defines the reference circuit, but the 1X reference is the smallest sizing in the logic chain. Larger devices are obtained by scaling the circuit. For example, a 4X NOT gate would have nFET and pFET sizes of 4 and 4r, respectively.

The **logical effort** g of a gate is defined by the ratio of capacitance to that of the reference gate:

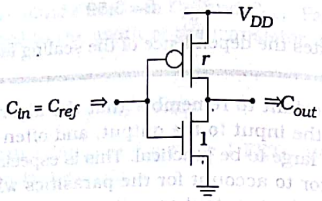


Figure 8.15 Reference inverter for logical effort

$$g = \frac{C_{in}}{C_{ref}} \tag{8.99}$$

Note that the parameter g has the same name as the technique; to distinguish between the two, we will treat the technique as a proper noun and use capital letters: **L**ogical **E**ffort. For the 1X inverter,

$$C_{in} = C_{ox}(A_{Gn} + A_{Gp}) \tag{8.100}$$

where A_{Gn} and A_{Gp} are the areas of the respective gates

$$A_{Gn} = W_n L \qquad \text{and} \qquad A_{Gp} = W_p L \tag{8.101}$$

with L the common channel length. Since $W_p = r\,W_n$,

$$\begin{aligned} C_{in} &= C_{ox} L W_n (1 + r) \\ &= C_{Gn}(1 + r) \\ &= C_{ref} \end{aligned} \tag{8.102}$$

defines the reference input capacitance C_{ref}. Then, by definition, the logical effort of the 1X inverter is

$$g_{NOT} = \frac{C_{ref}}{C_{ref}} = 1 \tag{8.103}$$

The value of $g_{NOT} = 1$ provides the basis for comparing the performance of other gates. Note that the nFET gate capacitance C_{Gn} is the base unit of input capacitance.

The **electrical effort** h is defined by the capacitance ratio

$$h = \frac{C_{out}}{C_{in}} \tag{8.104}$$

where C_{out} is the *external* load capacitance seen at the output. One word of caution in the notation: in the context of Logical Effort, C_{out} is the same as C_L used in the rest of the book. The notation has been changed in this section to allow a smoother transition for those who want to pursue deeper studies in the technique. The electrical effort is the ratio of electrical drive strength that is required to drive C_{out} relative to that needed to drive its own input capacitance C_{in}.

The absolute **delay time** d_{abs} through the inverter is written in the form

$$d_{abs} = \kappa R_{ref}(C_{p,ref} + C_{out}) \quad \text{sec} \tag{8.105}$$

using the circuit drawn in Figure 8.16. The reference FET resistance R_{ref} is the same for both transistors since the design is symmetric. The total capacitance at the output node consists of the external value C_{out} and the internal parasitic capacitance $C_{p,ref}$ (i.e., the FET capacitance C_{FET} in our notation). The factor κ is the scaling multiplier; to obtain correlation with the analysis in Chapter 6, we would choose $\kappa = \ln(9) \approx 2.2$.

Now consider an inverter that is scaled by a factor $S > 1$. The relative transistor sizes are increased to S and rS for the nFET and pFET, respectively. The FET resistance decreases to

$$R = \frac{R_{ref}}{S} \tag{8.106}$$

and the parasitic capacitance increases to

$$C_p = SC_{p,ref} \tag{8.107}$$

The delay for the scaled gate is then

$$\begin{aligned} d_{abs} &= \kappa R(C_p + C_{out}) \\ &= \kappa \frac{R_{ref}}{S}(SC_{p,ref} + C_{out}) \end{aligned} \tag{8.108}$$

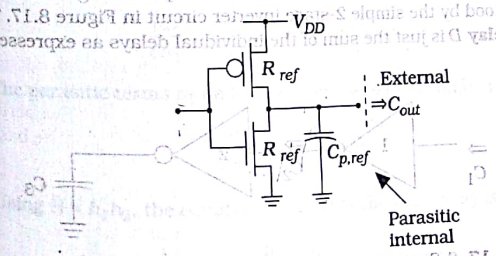


Figure 8.16 Delay circuit for a 1X inverter

Now note that the input capacitance for the scaled gate is

$$C_{in} = SC_{ref} \tag{8.109}$$

Distributing the terms then gives

$$\begin{aligned} d_{abs} &= \kappa\frac{R_{ref}}{S}SC_{p,ref} + \kappa\frac{R_{ref}}{S}C_{out} \\ &= \kappa R_{ref}C_{p,ref} + \kappa\frac{R_{ref}}{S}\left(\frac{C_{out}}{C_{ref}}\right)C_{ref} \\ &= \kappa R_{ref}C_{p,ref} + \kappa R_{ref}C_{ref}\left(\frac{C_{out}}{C_{in}}\right) \end{aligned} \tag{8.110}$$

Defining the reference time constant

$$\tau = \kappa R_{ref}C_{ref} \tag{8.111}$$

allows us to factor the delay into the form

$$d_{abs} = \tau(h+p) \tag{8.112}$$

where h is the electrical effort and

$$p = \frac{\tau_{par}}{\tau} = \frac{R_{ref}C_{p,ref}}{R_{ref}C_{ref}} \tag{8.113}$$

is the delay term associated with the parasitic capacitance. The **normalized delay**

$$d = \frac{d_{abs}}{\tau} = h + p \tag{8.114}$$

is unitless, and provides the important information about the gate. In the technique of Logical Effort, emphasis is placed on finding d for different paths.

The fundamental ideas behind the technique of Logical Effort can be understood by the simple 2-stage inverter circuit in Figure 8.17. The total **path delay** D is just the sum of the individual delays as expressed by

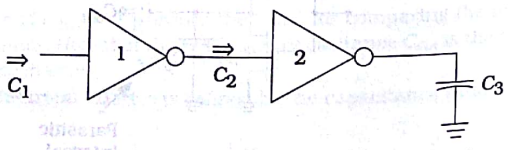


Figure 8.17 2-Stage inverter chain

$$\begin{aligned} D &= d_1 + d_2 \\ &= (h_1 + p_1) + (h_2 + p_2) \end{aligned} \tag{8.115}$$

where

$$h_1 = \frac{C_2}{C_1}, \qquad h_1 = \frac{C_3}{C_2} \tag{8.116}$$

are the individual electrical effort values. The **path electrical effort** H is defined as the ratio

$$H = \frac{C_{last}}{C_{first}} \tag{8.117}$$

and can be expressed as the product

$$H = h_1 h_2 \tag{8.118}$$

as seen from

$$H = \left(\frac{C_2}{C_1}\right)\left(\frac{C_3}{C_2}\right) = \frac{C_3}{C_1} \tag{8.119}$$

The product form is a general property of H, Using

$$h_2 = \frac{H}{h_1} \tag{8.120}$$

the path delay equation becomes

$$D = (h_1 + p_1) + \left(\frac{H}{h_1} + p_2\right) \tag{8.121}$$

The primary goal of Logical Effort techniques is to minimize the delay time though logic chains. For the present case, this condition can be found by calculating the derivative

$$\frac{\partial D}{\partial h_1} = \frac{\partial}{\partial h_1}\left[(h_1 + p_1) + \left(\frac{H}{h_1} + p_2\right)\right] \tag{8.122}$$

The parasitic terms p_1 and p_2 are constants to the differentiation so

$$\frac{\partial D}{\partial h_1} = 1 - \frac{H}{h_1^2} = 0 \tag{8.123}$$

Using $H = h_1 h_2$, the equation shows that the path delay is minimized if

$$h_1 = h_2 \tag{8.124}$$

Since the delay through an inverter is proportional to h, this is equivalent

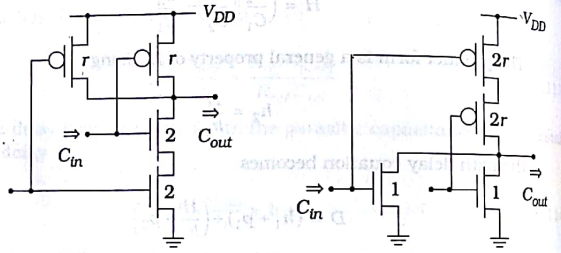
to saying that the path delay is minimized by equalizing the delay throug each stage. This, of course, is the same conclusion we arrived at in th more rigorous analysis.

8.3.2 Generalization

The real power of the Logical Effort technique is that it can be generalize to include arbitrary CMOS logic gates. The calculations allow one to est mate delays through logic cascades and provide scaling relationships fo minimum-delay designs.

The first step toward generalizing the technique is to develop expres sions for the logical effort parameter g of basic CMOS gates. All calcula tions are referenced to the 1X reference inverter with an input capacitance C_{ref} and transistor resistance R_{ref}. The simplest designs are those that maintain a symmetrical design, i.e., $R_n = R_p = R_{ref}$. This requires us to adjust the sizes of series-connected transistors.

Figure 8.18(a) shows a symmetric 1X NAND2 gate. The pFET sizes are still r, since the worst-case path from the output to the power supply is the same as an inverter. The nFETs, however, must be twice as large as



(a) NAND2 (b) NOR2

Figure 8.18 Symmetric NAND and NOR gates

the inverter values since they are in series; their relative values are thus denoted as being 2. For either input, the input capacitance is then

$$C_{in} = C_{Gn}(2 + r) \tag{8.125}$$

so that the logical effort for the NAND2 gate is

$$g_{NAND2} = \frac{C_{Gn}(2+r)}{C_{ref}} = \frac{2+r}{1+r} \tag{8.126}$$

This is sufficient to characterize the gate for the delay calculation.

The 1X NOR2 circuit in Figure 8.18(b) is analyzed in the same manner. The parallel-connected nFETs have a relative size of 1 while the pFETs are chosen to have sizes of $2r$ to make R_p the same as R_{ref}. The input capacitance is then

$$C_{in} = C_{Gn}(1 + 2r) \tag{8.127}$$

so that the logical effort of the gate is

$$g_{NOR2} = \frac{C_{Gn}(1+2r)}{C_{ref}} = \frac{1+2r}{1+r} \tag{8.128}$$

Note that the numerical values of g depend upon the ratio r.

These results may be generalized to larger fan-in gates. An n-input NAND gate will have n-parallel pFETs with size r and n-series nFETs that have a sizing n. The capacitance seen at an input is

$$C_{in} = C_{Gn}(n + r) \tag{8.129}$$

so that the logical effort is

$$g_{NAND} = \frac{n+r}{1+r} \tag{8.130}$$

An n-input NOR gate has a logical effort of

$$g_{NOR} = \frac{1+nr}{1+r} \tag{8.131}$$

as can be verified using the same approach. It is easily seen that any basic CMOS gate can be characterized for a value of logical effort g.

The delay through a general gate is expressed as

$$d = gh + p \tag{8.132}$$

The primary effect of the logical effort parameter g is to modify the first term to account for the difference in drive characteristics among various gates. For a logic cascade with N stages, each gate will be characterized by a delay

$$d_i = g_i h_i + p_i \tag{8.133}$$

for $i = 1$ to N. The **total path delay** D is the sum

$$D = \sum_{i=1}^{N} d_i = \sum_{i=1}^{N} (g_i h_i + p_i) \tag{8.134}$$

The **path logical effort** G is just the product of the individual factors

$$G = \prod_{i=1}^{N} g_i = g_1 g_2 \ldots g_N \tag{8.135}$$

and the **path electrical effort** H is defined in a similar manner by

$$H = \prod_{i=1}^{N} h_i = h_1 h_2 \ldots h_N \tag{8.136}$$

These combine to give the **path effort** F

$$\begin{aligned} F &= GH \\ &= (g_1 h_1)(g_2 h_2)(g_3 h_3)\ldots(g_N h_N) \\ &= f_1 f_2 \ldots f_N \end{aligned} \tag{8.137}$$

A minimum delay through the cascade is achieved if

$$g_i h_i = \text{ constant } = \hat{f} \tag{8.138}$$

for every i. This is consistent with our conclusions for the simple 2-stage inverter chain. The optimum path effort is thus

$$F = \hat{f}^N \tag{8.139}$$

so that the fastest design is where each stage has

$$gh = \hat{f} = F^{1/N} \tag{8.140}$$

This is the main equation of Logical Effort. The composition of an N-stage logic chain allows us to find the value of F. Each stage can be sized to accommodate the optimum electrical effort value

$$h_i = \frac{\hat{f}}{g_i} \tag{8.141}$$

The optimized path delay is then

$$\hat{D} = NF^{1/N} + P \tag{8.142}$$

where

$$P = \sum_{i=1}^{N} p_i \tag{8.143}$$

is the sum of the parasitic delays. In general, p_{ref} for an inverter is the smallest, with multiple-input gates exhibiting larger parasitic delay times. One simple estimate is to write

$$p = np_{ref} \tag{8.144}$$

as the parasitic delay for an n-input gate.

Example 8.4

Let us analyze the logic cascade in Figure 8.19 using the technique of Logical Effort. We will assume values of $C_4 = 500$ fF and $C_1 = 20$ fF. First, the path logical effort is given by

$$\begin{aligned} G &= g_{NOT} g_{NOR2} g_{NAND2} \\ &= (1)\left(\frac{1+2r}{1+r}\right)\left(\frac{2+r}{1+r}\right) \end{aligned} \tag{8.145}$$

Assuming a value of $r = 2.5$, we compute

$$G = (1)\left(\frac{6}{3.5}\right)\left(\frac{4.5}{3.5}\right) = 2.2 \tag{8.146}$$

The path electrical effort is

$$H = \frac{C_4}{C_1} = \frac{500}{20} = 25 \tag{8.147}$$

so that the path effort is

$$F = GH = 55 \tag{8.148}$$

The optimum stage effort is

$$\hat{f} = F^{1/N} = (55)^{1/3} = 3.8 \tag{8.149}$$

which gives a total path delay of

$$\begin{aligned} \hat{D} &= 3(3.8) + P \\ &= 11.41 + P \end{aligned} \tag{8.150}$$

where

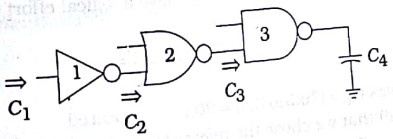


Figure 8.19 Logic cascade for Example 8.4

$$P = (p_{NOT} + p_{NOR2} + p_{NAND2}) \tag{8.151}$$

is the parasitic delay term that is determined by the process specifications.

The sizing equations are obtained from the analysis using the optimized quantities. Starting from the NAND2 gate at the output with $g_{NAND2} = (4.5/3.5) = 1.29$ we have

$$h_3 = \frac{3.8}{1.29} = 2.95 = \frac{C_4}{C_3} \tag{8.152}$$

so that

$$C_3 = \frac{500}{2.95} = 169.5\,\text{fF} \tag{8.153}$$

Since C_3 in the input capacitance into a NAND2 gate, we may use equation (8.125) to write a scaled gate as

$$\begin{aligned} C_3 &= S_3 C_{Gn}(2+r) \\ &= S_3(4.5C_{Gn}) \end{aligned} \tag{8.154}$$

where S_3 is the scaling factor.

The NOR2 gate is analyzed in the same manner. Since $g_{NOR2} = 1.71$, we have

$$h_2 = \frac{3.8}{1.71} = 2.22 = \frac{C_3}{C_2} \tag{8.155}$$

Thus,

$$C_2 = \frac{169.5}{2.22} = 76.35\ \text{fF} \tag{8.156}$$

The input capacitance into the NOR2 gate is

$$\begin{aligned} C_2 &= S_2 C_{Gn}(1+2r) \\ &= S_2(6C_{Gn}) \end{aligned} \tag{8.157}$$

The input NOT gate is defined to have a logical effort of 1 so

$$h_1 = \frac{3.8}{1} = \frac{C_2}{C_1} \tag{8.158}$$

This gives $C_1 = (76.35/3.8) = 20$ fF as required.

Recall that we chose the reference as the input NOT gate with $C_1 = C_{ref} = 2.5\,C_{Gn}$. The NOR and NAND gates then scale as

$$\begin{aligned} S_2 &= \frac{76.35}{(6)(3.5C_{Gn})} = \frac{3.64}{C_{Gn}} \\ S_3 &= \frac{169.5}{(4.5)(3.5C_{Gn})} = \frac{10.76}{C_{Gn}} \end{aligned} \tag{8.159}$$

to achieve the minimum delay. These scaling values are referenced to a capacitance of

$$C_{Gn} = \frac{20}{3.5} = 5.71\ \text{fF} \tag{8.160}$$

where

$$C_{Gn} = C_{ox} W_n L \tag{8.161}$$

gives the reference nFET channel width W_n.

Another approach is to choose a minimum size 1X inverter as the reference. If, for example, $C_{ref} = 8$ fF for a 1X gate, then the scale factors are $S_1 = 2.5$ (for the NOT gate), $S_2 = 1.59$, and $S_3 = 4.71$. Usually the reference can be chosen for convenience.

8.3.3 Optimizing the Number of Stages

A well-known characteristic of CMOS logic cascades is the fact that one can often insert inverters into a logic chain and decrease the total delay time. While this may play against simple intuition developed in introductory logic design courses, it is based in the fact that distributing out the drive strength among several stages is more important than counting the number of logic symbols. Logical Effort shows this feature using the path delay D.

First, note that the logical effort of an inverter is $g_{NOT} = 1$. Since

$$G = g_1 g_2 \cdots g_N \tag{8.162}$$

multiplying by additional factors of g_{NOT} does not change the numerical value of the path effort

$$F = GH \tag{8.163}$$

Delay time minimization is expressed by

$$\begin{aligned} \hat{f} &= F^{1/N} \\ &= (GH)^{1/N} \end{aligned} \tag{8.164}$$

such that the total path delay is

$$\hat{D} = NF^{1/N} + P \tag{8.165}$$

In general, $F^{1/N}$ decreases with increasing N. Thus, it may be possible to obtain a smaller path delay by inserting the inverters. Note, however, that the increased parasitic delay in P due to the extra inverters will often some of the performance.

Example 8.5

To see the dependence, suppose that $F = 200$. For $N = 3$,

$$3(200)^{1/3} = 17.54 \tag{8.166}$$

For $N = 4$,

$$4(200)^{1/4} = 15.04 \tag{8.167}$$

and $N = 5$ gives

$$5(200)^{1/5} = 14.43 \tag{8.168}$$

However, if we try $N = 10$, then the term increases

$$10(200)^{1/10} = 16.99 \tag{8.169}$$

so that we have passed the optimum number of stages.

An analysis of the problem shows that the optimum number of stages for a given F is obtained by solving the transcendental equation [8]

$$F^{1/N}[1 - \ln(F^{1/N})] + p_{ref} = 0 \tag{8.170}$$

This can be rewritten into a simpler looking form by defining

$$\rho = F^{1/N} \tag{8.171}$$

so that

$$\rho[1 - \ln(\rho)] + p_{ref} = 0 \tag{8.172}$$

A moment's reflection confirms that this has the same form as equation (8.93) that was derived from circuit considerations, thus demonstrating the equivalence of the two approaches. The power of Logical Effort is that it is not restricted to inverters.

For small values of p_{ref}, the approximate solutions are

$$\rho \approx 0.71 p_{ref} + 2.82 \tag{8.173}$$

which is useful for estimating the optimum value of N during an initial design phase.

8.3.4 Logical Area

The real estate area is important, particularly in scaled designs. An estimate of the circuit requirements can be obtained using Logical Effort quantities by simply summing the gate areas of each FET by calculating the logical area (LA) for the i-th gate using

$$LA_i = W_i \times L \tag{8.174}$$

where L is the channel length and W_i is determined by the sizing. For example, the logical area of a 1X NOT gate with $L = 1$ unit is

$$LA_{NOT} = 1 + r \tag{8.175}$$

which accounts for the pFET and nFET sizes. If this is scaled by a factor $S > 1$, then the logical area increases to

$$LA_{NOT} = S(1 + r) \tag{8.176}$$

Similarly, a scaled NOR2 gate has

$$LA_{NOR2} = S(1 + 2r) \tag{8.177}$$

while

$$LA_{NAND2} = S(2 + r) \tag{8.178}$$

applies to a NAND2 gate. For a network with M gates, the total logical area is

$$LA = \sum_{i=1}^{M} LA_i \tag{8.179}$$

This allows a simple metric for comparing area requirements of different designs. Note, however, that since it ignores drain and source spacings, interconnect wiring, well, etc., it is only a rough estimate.

8.3.5 Branching

The technique of Logical Effort applies to a well-defined path. When a logic gate drives two or more gates, the data path splits and we must account for presence of the gates that are not in the main path, but contribute capacitance. This situation is portrayed in the logic diagram of Figure 8.20, where the main path of interest from In to Out has been highlighted. Tracing the circuit shows two branching points. In both cases, the NOR2 gates add capacitance to the NAND2 loads and cannot be ignored.

These effects are handled by introducing the **branching effort** b at every branch point such that

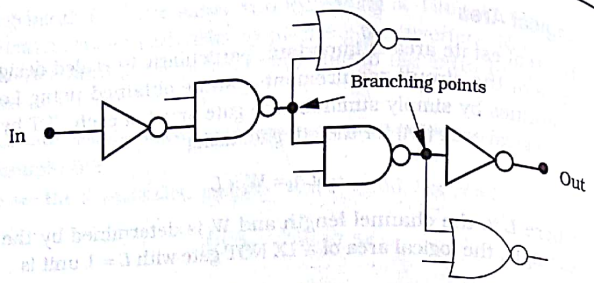


Figure 8.20 Branching

$$b = \frac{C_T}{C_{path}} \tag{8.180}$$

where C_{path} is the capacitance in the main logic path, and

$$C_T = C_{path} + C_{off} \tag{8.181}$$

represents the total capacitance seen at the node. In this equation, C_{off} includes all capacitance contributions that are off of the main path. The branching effort has the property that $b > 1$ and accounts for the additional loading. The **path branching effort** is given by the product

$$B = \prod_i b_i \tag{8.182}$$

where b_i are the individual branching efforts.

Example 8.6

Consider the logic network in Figure 8.20. At the first branch point, a NAND2 gate drives another on-path NAND2, and an off-path NOR2 gate. Assuming unit gate sizes, the branching effort b_1 for this point is

$$b_1 = \frac{C_{NAND2} + C_{NOR2}}{C_{NAND2}} = \frac{(2+r)+(1+2r)}{(2+r)} = \frac{3(1+r)}{(2+r)} \tag{8.183}$$

The second branch point in the drawing is described by

$$b_2 = \frac{C_{NOT} + C_{NOR2}}{C_{NOT}} \tag{8.184}$$

or

$$b_2 = \frac{(1+r)+(1+2r)}{(1+r)} = \frac{(2+3r)}{(1+r)} \tag{8.185}$$

The path branching effort is then

$$B = \frac{3(1+r)}{(2+r)}\frac{(2+3r)}{(1+r)} = \frac{3(2+3r)}{(2+r)} \tag{8.186}$$

for the selected path from In to Out.

Once the path branching effort has been calculated, we modify the path effort F to read

$$F = GHB \tag{8.187}$$

and the calculation proceeds in the same manner as for the simpler case without branching. This allows us to extend Logical Effort to arbitrary logic configurations and analyze every path for relative delay.

8.3.6 Summary

This short discussion of Logical Effort illustrates the usefulness of the technique. It is particularly valuable in advanced systems design where we have the choice of several algorithms that lead to the same result. Logical Effort allows us to compare the performance of the different circuits to see which is better for our design. These considerations will be discussed in later chapters of the book.

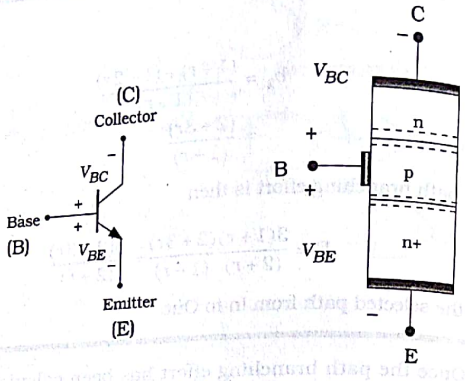
8.4 BiCMOS Drivers[2]

BiCMOS is a modified CMOS technology that includes bipolar junction transistors as circuit elements. In digital design, BiCMOS stages are used to drive high-capacitance lines more efficiently than MOSFET-only circuits. BiCMOS processing is more expensive than standard CMOS, and bipolar transistors have an intrinsic voltage drop that cannot be avoided making them undesirable for low-voltage applications.

8.4.1 Bipolar Junction Transistor Characteristics

A bipolar junction transistor (BJT) is a 3-terminal element that obtains its electrical characteristics from the properties of pn junctions. There are two types of BJTs, npn and pnp. The current flowing through an npn

[2] This section can be skipped without loss of continuity in the discussion.



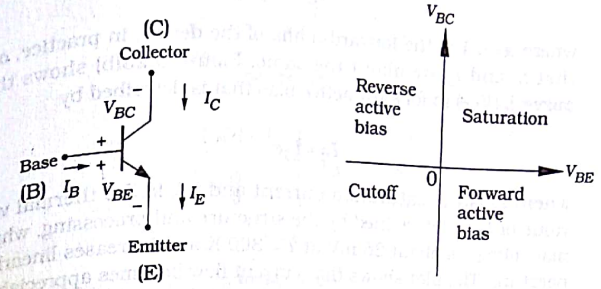
(a) Symbol (b) Structure

Figure 8.21 Symbol and structure of an npn BJT

transistor is due mostly to electrons, while that through a pnp device is due to holes. Since electrons are faster than holes, we concentrate on using npn devices in high-speed BiCMOS circuits.

The circuit symbol for an npn BJT is shown in Figure 8.21(a). The device has three terminals that are called the base (B), the emitter (E) with the arrowhead, and the collector (C). A simplified "prototype" structure of the npn BJT is shown in Figure 8.21(b); this illustrates the npn layer that gives the device its name. The drawing shows that the npn transistor can be viewed as two back-to-back pn junction diodes, one between the base and emitter terminals and the other between the base and collector electrodes. Current flow through the BJT is controlled by two voltages, the base-emitter voltage V_{BE} and the base-collector voltage V_{BC}, that bias the two pn junctions. They are defined to be positive values when the "+" polarity is applied to the p-type base layer. A positive voltage indicates a forward bias on the junction that allows current flow, while a negative voltage is a reverse bias.

The operation of the bipolar transistor is complicated by the fact that the voltages can be either positive or negative (reversed polarity). Consider the situation shown in Figure 8.22(a). The currents I_C, I_B, and I_E are determined by the voltages, but each combination of polarities gives a different mode of operation. These are summarized by the plot shown in Figure 8.22(b) that indicates the polarities of V_{BE} and V_{BC} by quadrants. **Forward-active bias** is defined by $V_{BE} > 0$ and $V_{BC} < 0$, i.e., the base-emitter junction is forward biased and the base-collector junction is reverse biased. This mode of operation allows for amplification and controlled current flow, and is used for analog circuits. The opposite case

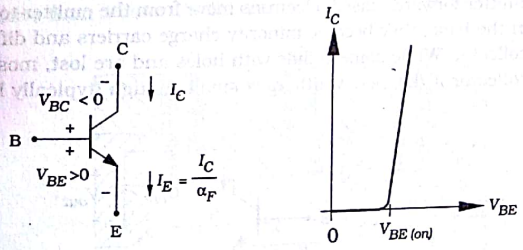


(a) Symbol and parameters (b) Operating regions

Figure 8.22 Operating regions of the bipolar junction transistor

where $V_{BE} < 0$ and $V_{BC} > 0$ is called **reverse-active bias**, and is used only in a few special cases. If both junctions are forward biased with $V_{BE} > 0$ and $V_{BC} > 0$, the device is said to be in **saturation**. In this case, large currents can flow through the device but the transistor does not control the values. It is important to remember that saturation in a BJT has no relation to a saturated FET. The final case is where both junctions are reverse biased with $V_{BE} < 0$ and $V_{BC} < 0$. Only small leakage currents flow and the BJT is said to be in **cutoff**. This can be modeled as an open switch.

Bipolar transistors are faster than MOSFETs but are more complicated to build into an integrated circuit. Let us examine forward-active bias to understand why a bipolar circuit can provide faster switching. Figure 8.23(a) shows the device with this bias. The collector and emitter currents are related by



(a) Bias (b) Transfer curve

Figure 8.23 Forward-active bias in a BJT

$$I_C = \alpha_F I_E \tag{8.188}$$

where $\alpha_F < 1$ is the forward-alpha of the device; in practice, $\alpha_F \approx 0.99$ so that I_C and I_E are about the same. Figure 8.23(b) shows the **transfer curve** $I_C(V_{BE})$ in forward-active bias that is described by

$$I_C \approx I_S e^{V_{BE}/V_{th}} \tag{8.189}$$

where I_S is the saturation current and V_{th} is the thermal voltage. The value of I_S is determined by the structure and processing, while the thermal voltage is about 26 mV at $T = 300$ K and increases linearly with temperature. The plot shows that current flow becomes appreciable when the base-emitter voltage reaches a value of $V_{BE(on)}$, which is usually estimated to be about 0.5 V to 0.7 V. Once this is achieved, the current increases exponentially with increasing V_{BE}.

Consider the simple circuit shown in Figure 8.24. With the BJT in forward-active bias, the current flow out of the capacitor is

$$I_C = -C_{out}\frac{dV_{out}}{dt} = I_S e^{V_{BE}/V_{th}} \tag{8.190}$$

We can estimate the discharge time by

$$\Delta t \approx \frac{(\Delta V_{out})}{I_C C_{out}} \tag{8.191}$$

where ΔV_{out} is the change in voltage. The values of I_C can be large, easily reaching tens to hundreds of milliamperes, which reduces the discharge time Δt even for large value of C_{out}. A BJT accomplishes the task faster than a FET that occupies the same area, making BiCMOS attractive.

Current flow through a BJT is due to the mechanism of particle diffusion, not electric field aided motion as in a FET. The forward active operation of the prototype device is summarized in Figure 8.25. With the base-emitter forward biased, electrons move from the emitter to the base. Once in the base, they become minority charge carriers and diffuse toward the collector. While some collide with holes and are lost, most will reach the collector if the base width x_B is small enough (typically less than about

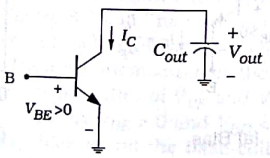


Figure 8.24 Discharge of a capacitor using a BJT

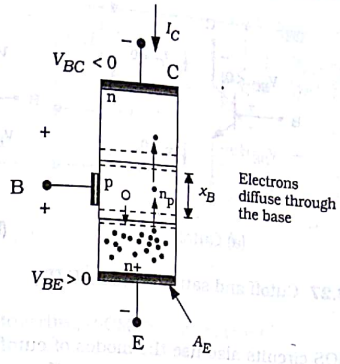


Figure 8.25 Forward-bias operation

0.5 μm). This establishes current flow from collector to the emitter. An analysis shows that the saturation current is given by

$$I_S = qA_E\frac{D_n n_i^2}{x_B N_{aB}} \tag{8.192}$$

where A_E [cm^2] is the emitter area, D_n [cm^2/sec] is the electron diffusion coefficient in the base and is a measure of the diffusive motion, q is the electron charge, and N_{aB} [cm^{-3}] is the acceptor doping in the base. A typical value for the saturation current is $I_S = 0.1$ pA $= 10^{-13}$ A. While this is quite small, the exponential dependence of the current on V_{BE} gives large values of I_C. The cross-sectional view of an integrated bipolar transistor is shown in Figure 8.26. The prototype structure can be seen in the center region underneath the emitter n+ region. Since specialized layers are required to create the device, the processing of a BiCMOS chip is more expensive than a basic CMOS design.

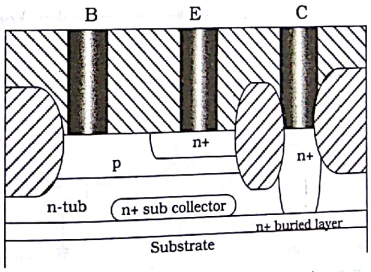
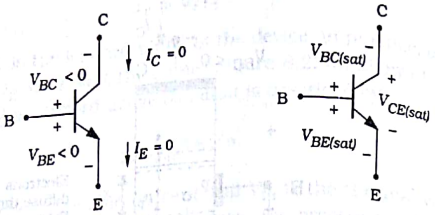


Figure 8.26 An integrated bipolar junction transistor



(a) Cutoff (b) Saturation

Figure 8.27 Cutoff and saturation in a BJT

BiCMOS circuits also use the modes of cutoff and saturation, which are summarized in Figure 8.27. In cutoff, both junctions are reverse-biased and both I_C and I_E are approximately 0 as in Figure 8.27(a). The device is saturated when both junctions are forward biased; this case is shown in Figure 8.27(b). In this case, the values of the currents are determined by the circuits that are connected to the transistor. The junction voltages take on constant values of $V_{BE(sat)}$ and $V_{BC(sat)}$ with typical values of around 0.8 V and 0.7 V, respectively. The collector-emitter voltage is thus about $V_{CE(sat)} \approx 0.1$ V by using Kirchhoff's law.

8.4.2 Driver Circuits

BiCMOS circuits employ CMOS logic circuits that are connected to a bipolar output driver stage. A general structure is shown in Figure 8.28. The CMOS network is used to provide logic operations and drive the output bipolar transistors Q1 and Q2. Only one BJT is active at a time. Transistor Q1 provides the high output voltage while Q1 discharges the output capacitance and gives the low output state.

The inverting circuit in Figure 8.29 gives an example of the operational

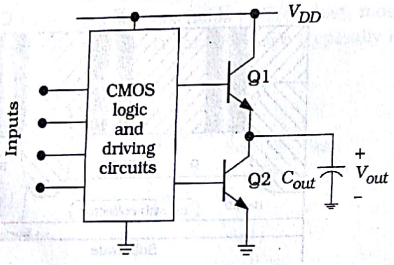


Figure 8.28 General form of a BiCMOS circuit

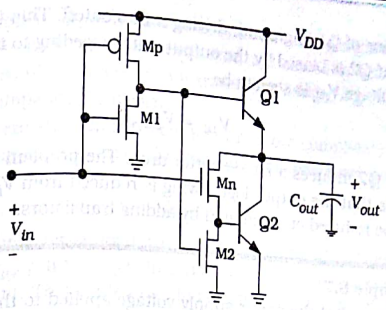


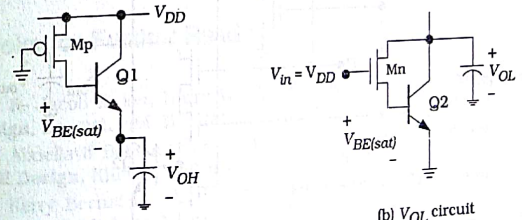
Figure 8.29 An inverting BiCMOS driver circuit

details. The NOT logic operation is performed by FETs Mp and Mn, even though they are separated from each other. The other two FETs M1 and M2 are used to provide paths to remove charge from the base terminals of Q1 and Q2, respectively. This speeds up the switching of the circuit, enhancing its use as an output driver.

Let us examine the DC operation of the circuit. Consider first the case where the input voltage is at a value of $V_{in} = 0$ V. This turns Mp on, while M1 and Mn are off. Since Mp and M1 form an inverter, the base of Q1 is high at a voltage of V_{DD}, and it goes active; the same voltage turns on M2, which grounds the base of Q2 and drives it into cutoff. The output high voltage V_{OH} for this case can be calculated from the subcircuit shown in Figure 8.30(a). Noting that Q1 will eventually enter saturation, we have

$$V_{OH} = V_{DD} - V_{BE(sat)} \tag{8.193}$$

since the voltage is dropped a value of $V_{BE(sat)}$ from the base to the output. The subcircuit for the case where $V_{in} = V_{DD}$ is shown in Figure 8.30(b). Now we see that Mp is off while M1 and Mn are on. M1 connects



(a) V_{OH} circuit (b) V_{OL} circuit

Figure 8.30 DC analysis of the output voltages

the base of Q1 to ground, driving it into cutoff. This in turn shuts off M2 so that Q2 is biased by the output voltage feeding to the base. The output low voltage V_{OL} is seen to be

$$V_{OL} = V_{BE(sat)} \tag{8.194}$$

since Q2 induces a base-emitter drop. The problem with this configuration is that the output logic swing is reduced from V_{DD} by $2V_{BE(sat)}$. This can be reduced or eliminated by adding transistors.

Example 8.7

Suppose that the power supply voltage applied to the BiCMOS circuit is $V_{DD} = 5$ V. Assuming that $V_{BE(sat)} = 0.8$ V,

$$V_{OH} = 5 - 0.8 = 4.3 \text{ V}$$
$$V_{OL} = 0.8 \text{ V} \tag{8.195}$$

which implies a logic swing of 3.4 V at the output. This can be improved by redesigning the output stage.

The CMOS circuitry can be modified to provide logic functions. A NAND2 gate based on this design is shown in Figure 8.31. A careful examination of the circuit shows that the logic is formed by the parallel pFETs driving Q1, and the series nFETs between the collector and base of Q2. The other FETs are used as pull-down devices to turn off the output transistors. Other logic functions can be designed using this as a basis. In

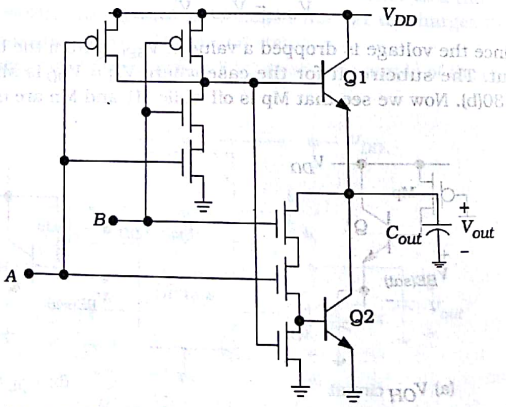


Figure 8.31 A BiCMOS NAND2 circuit

general, the upper output transistor uses a standard-design CMOS circuit as a driver. The nFET section is replicated and placed in between the collector and base of the lower output transistor; adding a pull-down nFET to the base completes the design.

It is apparent that BiCMOS circuits are more complicated than their CMOS equivalents. If we write the total output capacitance as

$$C_{out} = C_{transistors} + C_L \tag{8.196}$$

where C_L is the external load, we see that the parasitic transistor capacitance $C_{transistor}$ will be larger in a BiCMOS circuit due to the additional devices present. This leads to an important conclusion: BiCMOS is only effective for large values of C_L. A typical plot of time delay t_d versus C_L is shown in Figure 8.32. Due to the higher parasitic device capacitance, the CMOS and BiCMOS behaviors cross at a value $C_L = C_X$. For $C_L < C_X$, a standard CMOS design provides faster switching than a BiCMOS circuit. The speed increase is seen only for loads where C_L is much larger than C_X. This restricts the application of BiCMOS circuits to applications such as driving long data buses. Moreover, the cost and problem of V_{BE} drops are important factors in using the technology in digital VLSI.

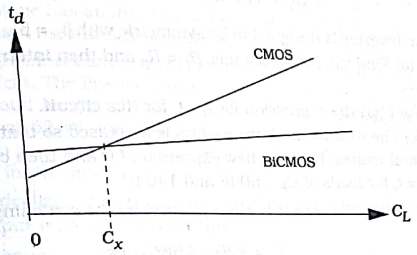


Figure 8.32 Gate delay versus external load capacitance

8.5 Books for Further Reading

[1] R. Jacob Baker, Harry W. Li, and David E. Boyce, **CMOS Circuit Design, Layout, and Simulation**, IEEE Press, Piscataway, NJ, 1998.

[2] Abdellatif Bellaouar and Mohamed I. Elmasry, **Low-Power Digital VLSI Design**, Kluwer Academic Publishers, Norwell, MA, 1995.

[3] Kerry Bernstein, et. al, **High Speed CMOS Design Styles**, Kluwer Academic Publishers, Norwell, MA, 1998.

[4] Ken Martin, **Digital Integrated Circuits**, Oxford University Press, New York, 2000.

[5] Robert F. Pierret, **Semiconductor Device Fundamentals**, Addison Wesley, Reading, MA, 1996.

[6] Jan M. Rabaey, **Digital Integrated Circuits**, Prentice Hall, Upper Saddle River, NJ, 1996.

[7] Jasprit Singh, **Semiconductor Devices**, John Wiley & Sons, New York, 2001.

[8] Ivan P. Sutherland, Bob Sproull, and David Harris, **Logical Effort**, Morgan-Kauffman Publishers, Inc., San Francisco, 1999.

[9] John P. Uyemura, **CMOS Logic Circuit Design**, Kluwer Academic Publishers, Norwell, MA, 1999.

[10] Neil H. E. Weste and Kamran Eshraghian, **Principles of CMOS VLSI Design**, 2nd ed., Addison-Wesley, 1993.

[11] Edward S. Yang, **Microelectronic Devices**, McGraw-Hill, New York, 1988.

8.6 Problems

[8.1] A CMOS inverter circuit has the following characteristics:

$$C_L = 100 \text{ fF} \qquad t_r = 123.75 \text{ ps}$$
$$C_L = 115 \text{ fF} \qquad t_r = 138.60 \text{ ps} \tag{8.197}$$

The inverter is designed to be symmetric with $\beta_n = \beta_p$. and $V_{Tn} = |V_{Tp}|$.

(a) Find the FET resistance $R_n = R_p$ and then internal FET capacitance C_{FET}.

(b) Find the expression for $t_f = t_r$ for this circuit.

(c) The width of both transistors is increased so that they are 3.2× their original values. Find the new expression for and then calculate the values of $t_f = t_r$ for loads of $C_F = 50$ fF and 140 fF.

[8.2] A CMOS inverter is characterized by the switching times

$$t_r = 430 + 3.68C_L \text{ ps}$$
$$t_f = 300 + 2.56C_L \text{ ps} \tag{8.198}$$

with the external load capacitance C_L in units of fF.

(a) Plot the rise and fall times for the range $C_L = 0$ to $C_L = 200$ fF.

(b) A three-inverter cascade is built using identical circuits. Find the worst-case delay through the chain if the output capacitance to each NOT gate is $C_L = 45$ fF.

[8.3] Consider the logic chain shown in Figure P8.1. The input at A is switched from a 1 to a 0. Find an expression for the delay time through the chain using the procedure developed for the network shown in Figure 8.6.

[8.4] A CMOS process is characterized by $C_{ox} = 8$ fF/μm^2, $r = 2.6$, and $L = 0.4$ μm. The magnitudes of the nFET and pFET threshold voltages are

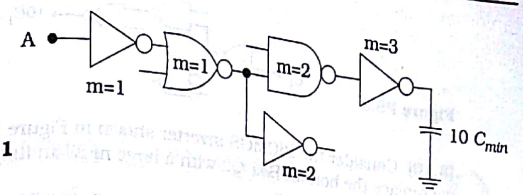


Figure P8.1

equal. A symmetrical inverter is designed using an nFET with a channel width of 2.2 μm. This acts as the input stage to a driver chain that has a load of $C_L = 38$ pF at the end. The design stipulates that the chain must produce an inverted signal with minimum delay from the input stage to the load.

(a) Calculate the input capacitance C_{in} of the inverter in units of fF.

(b) Apply idealized scaling to find the number of stages needed in the chain.

(c) It is known that an nFET with a channel width of W = 1 μm has a resistance of $R_n \approx 1725\ \Omega$. Given this, can you find the total delay time through the chain? If not, what other information is needed?

[8.5] Design a driver chain that will drive a load capacitance of $C_L = 40$ pF if the initial stage has an input capacitance of $C_{in} = 50$ fF. Use ideal scaling to determine the number of stages and the relative sizes.

[8.6] An interconnect line is described by a capacitance per unit length of $c' = 0.86$ pF/cm. The line itself runs over a significant portion of the chip and has a total length of 272 μm. A "standard" inverter has an input capacitance of 52 fF and uses symmetrical devices with $\beta_n = \beta_p$. The mobility ratio is $r = 2.8$ for the process. This is used as the first stage in a driver chain for the interconnect.

Use the idealized theory to design the driver chain with the constraint that the output must be non-inverting.

[8.7] Solve equation (8.93) for the case $\tau_x = 0.72\ \tau_r$.

[8.8] Consider the logic cascade shown in Figure P8.2. Use Logical Effort to find the relative size of each stage needed to minimize delay through the chain. Assume symmetric gates with $r = 2.5$.

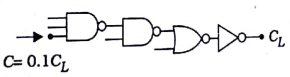


Figure P8.2

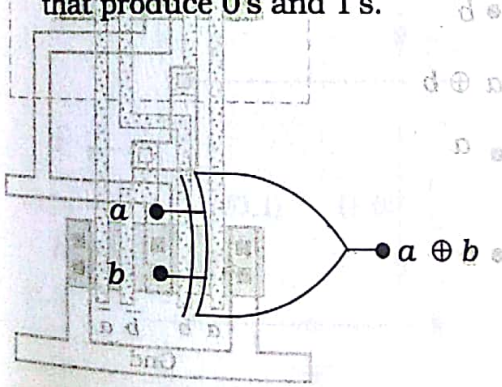
[8.9] The logic chain in Figure P8.3 is constructed in a process with $r = 2.5$. Determine the optimum sizing for each stage for the "highlighted" path indicated using the technique of Logical Effort.

9 Advanced Techniques in CMOS Logic Circuits

A wide variety of CMOS circuit design styles have been published that are useful in the design of high-speed VLSI networks. All are based on simple logic gates, but operate in distinct ways. Most advanced techniques have been developed to overcome one or more problems that have arisen as VLSI applications have increased over the years. Some are very general, while others are used only for special cases. In this chapter we will unleash a sampling of the modern CMOS circuit techniques that are used in VLSI. This will provide a basis for applications in later chapters.

9.1 Mirror Circuits

Mirror circuits are based on series-parallel logic gates, but are usually faster and have a more uniform layout. The basic idea of a mirror is seen from the XOR truth table in Figure 9.1. Output 0's imply that an nFET chain is conducting to ground, while an output 1 means that a pFET group provides support from the power supply. The important aspect of this observation is that there are equal numbers of input combinations that produce 0's and 1's.



a	b	$a \oplus b$	On devices
0	0	0	nFET
0	1	1	pFET
1	0	1	pFET
1	1	0	nFET

Figure 9.1 XOR function table

A mirror circuit uses the same transistor topology for the nFETs and pFETs. Applying this to the XOR function yields the circuit in Figure 9.2(a). The input combinations are shown for each branch. The "mirror" effect can be understood by placing a mirror along the output line, facing either up or down. The mirror image seen in the mirror will be the other side of the circuit. A layout for an XOR cell is shown in Figure 9.2(b); the pFETs are larger than the nFETs to compensate for the lower process transconductance (k') values.

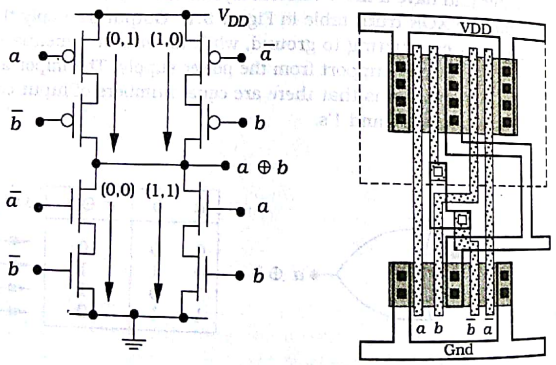
The advantages of the mirror circuit are more symmetric layouts and shorter rise and fall times. The latter comment can be understood using the RC switch model in Figure 9.3. Every path between the output and a power supply rail consists of two resistors and a parasitic inter-FET capacitor. The Elmore time constant is of the form

$$\tau_x = C_{out}(2R_x) + C_x R_x \tag{9.1}$$

where the subscript x is either n or p, depending upon the branch. Approximating the output voltage as being exponential gives the rise and fall time expressions

$$t_r \approx 2.2\tau_p$$
$$t_f \approx 2.2\tau_n \tag{9.2}$$

While the form is the same as for an AOI network, the rise time will be smaller because the parasitic capacitance C_p is smaller. This is due to the fact that the mirror circuit has only two pFETs contributing the C_p, while an AOI network has four transistors at that node.



(a) Circuit (b) Layout

Figure 9.2 XOR mirror circuit

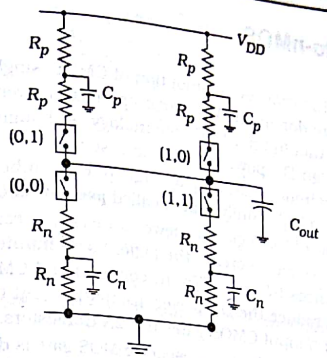
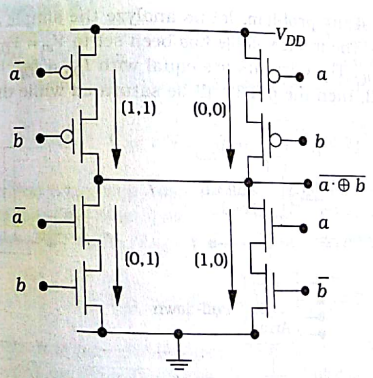


Figure 9.3 Switch model for transient calculations

The idea is easily used to create the XNOR circuit in Figure 9.4. It has the same basic features of the XOR. The relationship

$$\overline{a \oplus b} = \overline{\bar{a} \cdot b + a \cdot \bar{b}} \tag{9.3}$$

shows that only the inputs a and $\bar{a}$ need to be switched. Other mirror circuits will be introduced later in the text in the context of specific applications such as adder circuits.



a	b	$\overline{a \oplus b}$
0	0	1
0	1	0
1	0	0
1	1	1

Figure 9.4 Exclusive-NOR (XNOR) mirror circuit

9.2 Pseudo-nMOS

Prior to the widespread adoption of CMOS, single-FET polarity logic cuits were dominant. Many microprocessors were designed using nFET only circuits in an 'nMOS' technology. Although nMOS was abandoned due to high DC power dissipation, some of the main ideas are used in CMOS technology. Adding a single pFET to otherwise nFET-only circuits produces a logic family that is called **pseudo-nMOS**.

Pseudo-nMOS logic uses fewer transistors because only the nFET logic block is needed to create the logic. For N inputs, a pseudo-nMOS logic gate requires $(N + 1)$ FETs. In conventional CMOS, the pFET group is added to reduce the DC power dissipation, but the logic is superfluous. Standard N-input CMOS gates use $2N$ transistors.

The basic topology of a pseudo-nMOS gate is drawn in Figure 9.5. The single pFET is biased active since the grounded gate gives $V_{SGp} = V_{DD}$. It acts as a **pull-up** device that tries to pull the output f to the power supply voltage V_{DD}. Logic is performed by the nFET array that is designed using the same techniques we have seen. The array acts as a large switch between the output f and ground. If the switch is open, the pFET pulls up the output to a voltage $V_{OH} = V_{DD}$. If the nFET switch is closed, then the array acts as a **pull-down** device that tries to pull f down to ground. However, since the pFET is always biased on, V_{OL} can never achieve the ideal value of 0 V. It is tempting to use pseudo-nMOS circuits to reduce the FET count and area. However, this logic family is more complicated because the relative sizes of the transistors set the numerical value of V_{OL} and care must be taken to insure that V_{OL} is small enough to be an electronic logic 0 voltage.

To illustrate the sizing problem, let us analyze the simple inverter shown in Figure 9.6. The input voltage has been set to $V_{in} = V_{DD}$ so that output voltage is V_{OL}. The currents are equal with $I_{Dn} = I_{Dp}$. If V_{OL} is assumed to be small, then the pFET will be saturated while the nFET

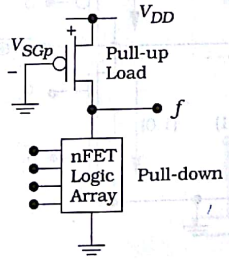


Figure 9.5 General structure of a pseudo-nMOS logic gate

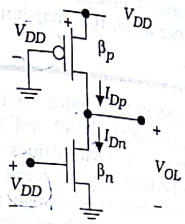


Figure 9.6 Pseudo-nMOS inverter

operates in the non-saturation region. The KCL equation thus assumes the form

$$\frac{\beta_n}{2}[2(V_{DD} - V_{Tn})V_{OL} - V_{OL}^2] = \frac{\beta_p}{2}(V_{DD} - |V_{Tp}|)^2 \tag{9.4}$$

which is a quadratic equation for V_{OL}. Solving gives the physical root

$$V_{OL} = (V_{DD} - V_{Tn}) - \sqrt{(V_{DD} - V_{Tn})^2 - \frac{\beta_p}{\beta_n}(V_{DD} - |V_{Tp}|)^2} \tag{9.5}$$

The value of V_{OL} thus depends on the ratio $(\beta_n/\beta_p) > 1$. Increasing the device ratio decreases the output low voltage. Because of this characteristic, pseudo-nMOS is a type of **ratioed** logic where the relative device sizes set V_{OL} or V_{OH}.

Example 9.1

Consider a CMOS process with $V_{DD} = 5$ V, $V_{Tn} = +0.7$ V, $V_{Tp} = -0.8$ V, $k'_n = 150\ \mu A/V^2$, and $k'_p = 68\ \mu A/V^2$. A pseudo-nMOS inverter sized with $(W/L)_n = 4$ and $(W/L)_p = 6$ gives an inverter with an output-low voltage of

$$V_{OL} = 4.3 - \sqrt{(4.3)^2 - \frac{408}{600}(4.2)^2} = 1.75 \text{ V} \tag{9.6}$$

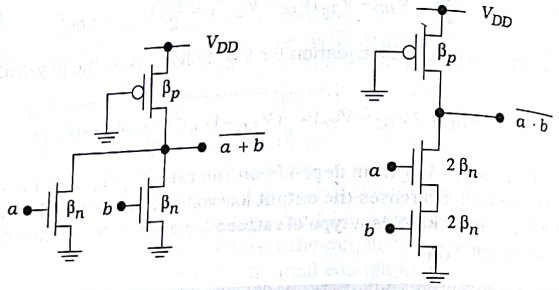
which is too large since it would not be interpreted as a logic 0 by a circuit of the same type. If we increase the nFET size to $(W/L)_n = 8$ and decrease the pFET to $(W/L)_p = 2$, the calculation gives

$$V_{OL} = 4.3 - \sqrt{(4.3)^2 - \frac{136}{1200}(4.2)^2} = 0.24 \text{ V} \tag{9.7}$$

which is acceptable since it is below the voltage $V_{in} = V_{Tn}$ that turns the nFET on. This illustrates that the choice of aspect ratios is critical to this design style. It is important to note that when $V_{in} = V_{DD}$, a current flow

path is established from V_{DD} to ground, leading to a large DC power dissipation. This is another factor that may limit the use of pseudo-nMOS circuits.

General pseudo-nMOS logic gates are designed using the same nFET arrays as in standard CMOS. NOR2 and NAND2 examples are shown in Figure 9.7. Let β_n and β_p be device values for an inverter. The NOR2 gate in Figure 9.7(a) can be based on the same β-values since the worst-case pull-down situation is when only a single nFET is active. This argument can be extended to an N-input NOR gate. The NAND2 gate in Figure 9.7(b) is complicated by the series nFETs. To obtain the same pull-down characteristics of the inverter, the logic transistors must be increased to $2\beta_n$ to provide the same total nFET resistance from the output to ground. This is a general problem with pseudo-nMOS logic gates that require series logic FETs.



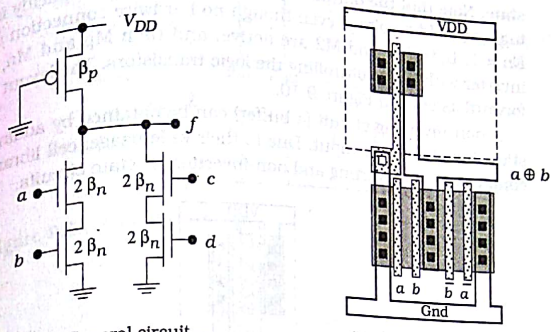
(a) NOR2 gate (b) NAND2 gate

Figure 9.7 Pseudo-nMOS NOR and NAND gates

A basic AOI circuit is shown in Figure 9.8(a) using the same sizing philosophy. The advantage in producing smaller simpler layouts can be seen by the XOR circuit in Figure 9.8(b). Since only a single pFET is used, the interconnect is much simpler. However, the sizes need to be adjusted to insure proper electrical coupling to the next stage. The problems associated with pseudo-nMOS limit its usage to situations where the layout problems are critical, or to some special switching situations where it yields simpler circuitry.

9.3 Tri-State Circuits

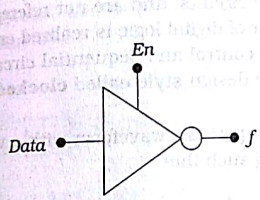
A tri-state circuit produces the usual 0 and 1 voltages, but also has a third **high-impedance** Z (or Hi-Z) state that is the same as an open circuit. Tri-state circuits are useful for isolating circuits from common bus lines.



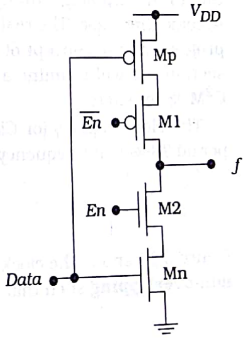
(a) General circuit (b) Layout example

Figure 9.8 AOI gate in pseudo-nMOS logic

The symbol for a tri-state inverter is shown in Figure 9.9(a). The enable signal En controls the operation. With $En = 0$, the output is "tri-stated" which means that $f = Z$. Normal operation occurs with $En = 1$. A CMOS circuit is shown in Figure 9.9(b). FETs M1 and M2 are the tri-stating devices. The $\overline{En}$ signal is applied to the pFET M1, while En controls M2. With $En = 0$, both M1 and M2 are off, and the output is isolated from both the power supply and ground. This is the circuit condition of the Hi-Z



En	f
0	Z
1	$\overline{Data}$



(a) Symbol and operation (b) CMOS circuit

Figure 9.9 Tri-state inverter

state. Note that the output capacitance (not shown explicitly in the dra
ing) can hold a voltage even though no hardwire connection exists. Wh
En = 1, both M1 and M2 are active, and then Mp and Mn act like a
inverter with *Data* controlling the logic transistors. The layout is straigh
forward as seen in Figure 9.10.

A non-inverting circuit (a buffer) can be obtained by adding a regula
static inverter to the input. Due to their wide usage, cell libraries usuall
contain several inverting and non-inverting tri-state circuits.

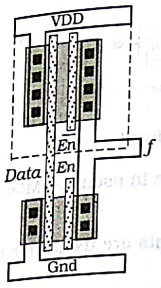


Figure 9.10 Tri-state layout

9.4 Clocked CMOS

Up to this point, all of the circuits we have examined have been com
pletely **static** in nature. The output of a static logic gate is valid so long as
the input values are valid and the circuit has stabilized. Logic delays are
due to the "rippling" through the circuits, and are not referenced to any
specific time base. The real power of digital logic is realized only when we
progress to the concept of clock control and sequential circuits. In this
section, we will examine a basic design style called **clocked CMOS**, or
C^2MOS for short.

The clock signal ϕ (or *Clk*) is a periodic waveform with a well-defined
period T [sec] and frequency f [Hz] such that

$$f = \frac{1}{T} \qquad (9.8$$

Figure 9.11 shows the clock $\phi(t)$ and its complement $\bar{\phi}(t)$. Ideally, these are
non-overlapping such that

$$\phi(t) \cdot \bar{\phi}(t) = 0 \qquad (9.9$$

for all times t. However, if $\phi(t)$ is defined to have a minimum value of 0 V
and a maximum of V_{DD}, then,

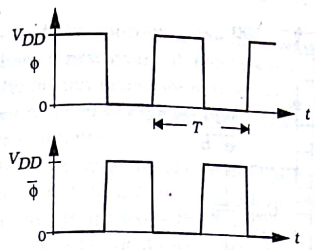


Figure 9.11 Clocking signals

$$\bar{\phi}(t) = V_{DD} - \phi(t) \qquad (9.10)$$

so that the clocks overlap slightly during a transition. It may be advanta
geous to create a set of clocks that are truly non-overlapping for all times.

The general structure of a C^2MOS gate is shown in Figure 9.12. It is
composed of a static logic circuit with tri-state output network (made up
of FETs M1 and M2) that is controlled by ϕ and $\bar{\phi}$. The operation of the cir
cuit can be understood using the clocking waveform shown. When $\phi = 1$,
both M1 and M2 are active. Since both the pFET and nFET logic blocks
are connected to the output node, the circuit degenerates to a standard
static logic gate. The output $f(a, b, c)$ is valid during this time, establish
ing the voltage V_{out} on the output capacitance C_{out}. When the clock
changes to a value of $\phi = 0$, both M1 and M2 are in cutoff, so the output is
in a high-impedance state Hi-Z. During this time interval, the FET logic
arrays are not connected to the output, so the inputs have no effect.
Instead, the output voltage is held on C_{out} until the clock returns to a
value of $\phi = 1$.

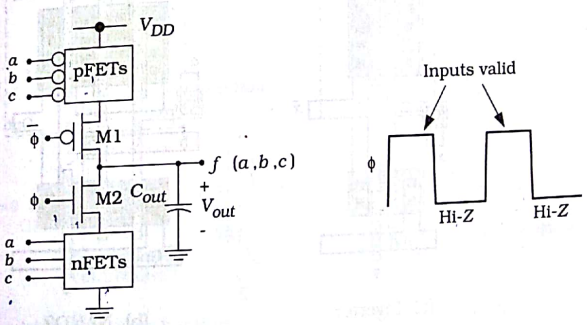
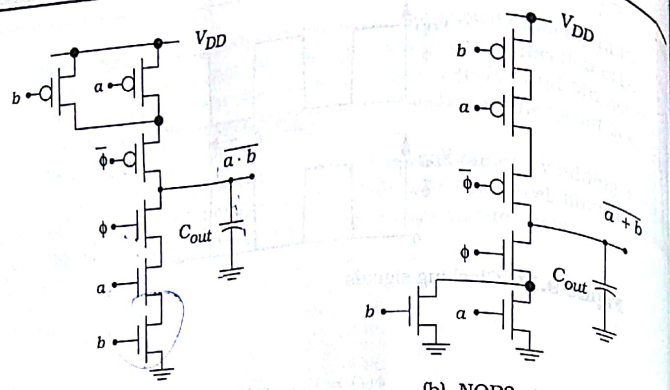


Figure 9.12 Structure of a C^2MOS gate

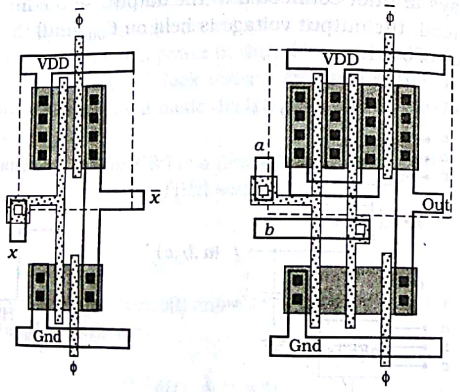


(a) NAND2 circuit (b) NOR2 circuit

Figure 9.13 Example of clocked-CMOS logic gates

The transistor arrays are designed using the same techniques as for standard logic gates. The circuits for a NAND2 and a NOR2 are shown in Figure 9.13, subdrawings (a) and (b), respectively. Layout is similar to the tri-state circuit with the clock replacing the enable signal. The layouts in Figure 9.14 provide one approach to placing and connecting the transistors. Note that the presence of the series-connected clocking FETs automatically lengthens both the rise and fall times of the circuit.

Clocked CMOS is useful because we can synchronize the data flow



(a) Inverter (b) NAND2

Figure 9.14 Layout examples of C^2MOS circuits

through a logic cascade by controlling the internal operation of the gate. Every cycle of ϕ allows a new group of data bits to enter the network. One drawback is that the output node cannot hold the charge on V_{out} very long due to a phenomenon called **charge leakage**. This places a lower limit on the allowable clock frequency.

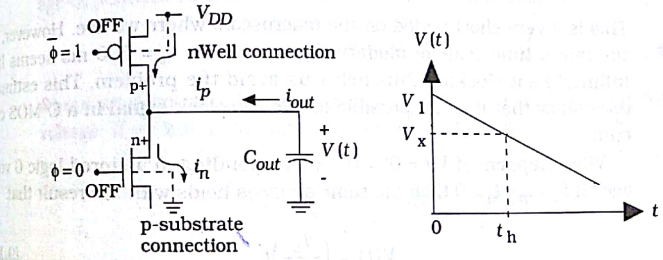
The basics of charge leakage are shown in Figure 9.15(a). Even though the transistors are in cutoff, it is not possible to block all current flow using a FET. If a voltage is applied to the drain or source, a small **leakage current** flows into, or out of, the device. There are many contributions to the leakage current. One is due to the required bulk connections that are shown in the drawing. The pFET bulk is the nWell region, which is connected to the power supply V_{DD}. Since the pFET source is a p+ region, this creates a pn junction (a diode) that admits a small leakage current i_p flowing on to the node. The nFET has the same problem, with i_n flowing from the output to the p-substrate. Denoting the current off of the capacitor by i_{out}, we may sum the contributions to obtain

$$\begin{aligned} i_{out} &= i_n - i_p \\ &= -C_{out}\frac{dV}{dt} \end{aligned} \tag{9.11}$$

where we have used the capacitor I-V relation in the second line; note the presence of a minus sign to indicate that i_{out} flows out of the positive terminal.

To see the effects of the leakage currents, suppose that we have an initial voltage $V(t = 0) = V_1$ stored on the capacitor. If $i_n > i_p$, then $i_{out} = I_L$ is a positive number, indicating current flow off of the capacitor. Rewriting the equation as

$$I_L = -C_{out}\frac{dV}{dt} \tag{9.12}$$



(a) Bulk leakage currents (b) Logic 1 voltage decay

Figure 9.15 Charge leakage problem

we may rearrange it to read

$$\int_{V_1}^{V(t)} dV = -\int_0^t \left(\frac{I_L}{C_{out}}\right) dt \tag{9.13}$$

Assuming that I_L is a constant, the equation may be integrated to yield

$$V(t) = V_1 - \left(\frac{I_L}{C_{out}}\right) t \tag{9.14}$$

which is a linear decay of the voltage with time. This is plotted in Figure 9.15(b). As the voltage decreases, it eventually reaches a minimum logic 1 value that is shown as V_x in the plot. If V falls below this value, it will incorrectly be interpreted as a logic 0 voltage. The **hold time** t_h corresponds to the maximum time that the logic 1 voltage can be stored. By definition, this occurs when

$$V(t_h) = V_1 - \left(\frac{I_L}{C_{out}}\right) t_h = V_x \tag{9.15}$$

Rearranging,

$$t_h = \left(\frac{C_{out}}{I_L}\right)(V_1 - V_x) \tag{9.16}$$

gives the hold time for this case. An order of magnitude estimate of the hold time can be obtained by estimating the capacitance as 50 fF, the leakage current as 0.1 pA, and the voltage change as 1 V. These values give

$$t_h = \left(\frac{50\times10^{-15}}{10^{-13}}\right)(1) = 0.5 \text{ sec} \tag{9.17}$$

This is a very short period on the macroscale where we live. However, on the micro time scale of modern digital CMOS, t_h = 500 ms seems like infinity! Fast clocking thus helps us avoid the problem. This estimate does show that it is not possible to idle the clock signal in a C^2MOS circuit.

What happens if $V(t = 0) = 0$ V corresponding to a stored logic 0 voltage? If $I_L = i_p - i_n > 0$ then the same analysis holds with the result that

$$V(t) = \left(\frac{I_L}{C_{out}}\right) t \tag{9.18}$$

i.e., the charging current I_C increases the voltage in time. This means that the logic 0 voltage may drift, so that we again require a minimum clock frequency.

In submicron devices, the charge leakage problem is exacerbated by the existence of another FET leakage current called the **subthreshold current** I_{sub}. This is a drain-source current that flows even though the gate voltage is less than V_T. A simple estimate for the subthreshold current is

$$I = I_{D0}\left(\frac{W}{L}\right) e^{-(V_{GS}-V_T)/(nV_{th})} \tag{9.19}$$

where I_{D0} varies with V_{DS}, V_{th} is the thermal voltage (kT/q) ≈ 26 mV at 300 K, and n is a parameter that varies with capacitance. A conservative value of I_{D0} is around 10^{-9} A, which noticeably reduces the hold time. With the previous values of capacitance and voltage and $V_{GS} = 0$, the hold time estimate is

$$t_h = \left(\frac{50\times10^{-15}}{10^{-9}}\right)(1) = 50 \text{ μs} \tag{9.20}$$

for leakage through a unity aspect ratio FET. In addition, other contributions to the leakage current originate from the physical structure and the materials used to create the silicon circuit. It would not be unreasonable to find a total charge leakage current of I_L = 0.1 μA = 10^{-7} A in a submicron device. With this level of leakage, the hold time is reduced to

$$t_h = \left(\frac{50\times10^{-15}}{10^{-7}}\right)(1) = 0.5 \text{ μsec} \tag{9.21}$$

This clearly indicates that charge storage on a capacitive node is a limited-time event, and places important constraints on our logic circuits.

Although we have been approximating the leakage currents as having a constant value for simplicity, a deeper analysis shows that they are voltage-dependent functions. The general differential equation is of the form

$$I_L(V) = -C_{out}(V)\frac{dV}{dt} \tag{9.22}$$

where we have noted that the output capacitance C_{out} also depends on voltage. If we know the explicit functions for $I_L(V)$ and $C_{out}(V)$, then

$$\int_0^t dt = \int_{V_x}^{V(t)} \frac{C_{out}(V)}{I_L(V)} dV = t \tag{9.23}$$

can be integrated to give $V(t)$. A more practical approach is to use a numerical solution. The dependence of the quantities on V results in a non-linear decay, such as the example illustrated in Figure 9.16. The hold time is still defined in the same manner. At the circuit design level, charge

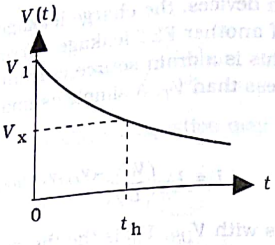


Figure 9.16 General voltage decay

leakage information is usually obtained from circuit simulations.

Charge leakage occurs whenever we attempt to hold charge on a node capacitance using a MOSFET in cutoff. Many of the advanced circuits in the remainder of this chapter have this characteristic, and it is important to remember to check for the problem. Simple SPICE models of MOSFETs do not accurately account for leakage currents. The best results to date are obtained using the BSIM equations.

Motivation for Future Research

While charge leakage is an important problem in dynamic circuits, this discussion highlights the problem of achieving an "open switch" using a MOSFET. As the dimensions shrink, the drain-to-source leakage current increases and the device looks less and less like the idealized switch that was used to design CMOS logic networks. This is one of the most critical problems in digital submicron VLSI. Device researchers are continually looking at the problem. In terms of silicon technology, two main approaches are prevalent. One technique is to reduce the leakage by refining the fabrication process using different materials and variations in the FET structures. Over the years, this has resulted in better devices that have "manageable" leakage current levels that circuit designers must work around.

The other approach is to develop new types of transistors to replace the standard MOSFET. Novel devices with improved characteristics have been proposed and built, and many promising structures have appeared in the literature. However, device research tends to be initially concerned with creating a single transistor, not a high-density VLSI chip. Manufacturing problems often limit the usage of the device in these applications. Another problem is that circuit and logic designers must learn the characteristics of a device before they can develop digital design methodologies. A technique that works with standard MOSFETs probably won't be the best choice for circuits based on transistors that have different I-V characteristics, if it works at all.

Shrinking the size of a MOSFET is often taken as natural evolution of the processing technology. The development of submicron sized FETs had a marked effect on circuit design techniques. Introducing new switching devices would affect all levels of the VLSI design hierarchy, and much research would have to be completed before high-density designs could be implemented. VLSI designers must be continually aware of changes in the field.

9.5 Dynamic CMOS Logic Circuits

A **dynamic logic gate** uses clocking and charge storage properties of MOSFETs to implement logic operations. The clock provides a synchronized data flow which makes the technique useful in designing sequential networks. The characterizing feature of a dynamic logic gate is that the result of a calculation is valid only for a short period of time. While this makes the circuits more difficult to design and use, they require fewer transistors and may be faster than static cascades.

Dynamic circuits are based on the circuit illustrated in Figure 9.17. The clock ϕ drives a complementary pair of transistors Mn and Mp; these control the operation of the circuit and provide synchronization. Logic is implemented using an nFET array between the output node and ground. The output voltage V_{out} is taken across the output capacitor C_{out}.

The clocking signal ϕ defines two distinct modes of operation during every cycle. When $\phi = 0$ the circuit is in **precharge** with Mp on and Mn off. This establishes a conducting path between V_{DD} and the output, allowing C_{out} to charge to a voltage of $V_{out} = V_{DD}$. Mp is often called the precharge FET. Since the bottom of the nFET logic block is not connected to ground during precharge, the inputs have no effect.

A clock transition to $\phi = 1$ drives the circuit into the **evaluation** mode where Mp is off and Mn is on. The inputs are valid and control the switching in the nFET logic array; Mn is usually called the evaluate transistor. If

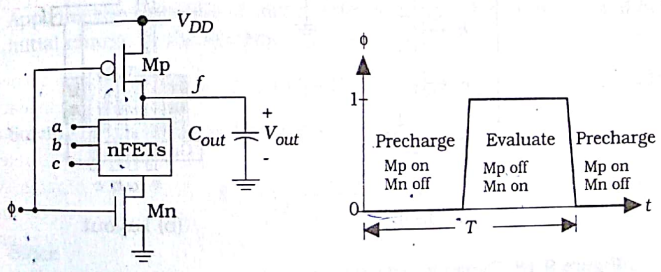


Figure 9.17 Basic dynamic logic gate

the logic block acts like a closed switch, then C_{out} can discharge through the logic array and Mn; this gives the final result of $V_{out} = 0$ V, corresponding to a logic $f = 0$. If the inputs cause the block to behave like an open switch from top to bottom, the charge on C_{out} is held and $V_{out} = V_{DD}$; logically, this is an output of $f = 1$. Charge leakage eventually drops the output to $V_{out} \rightarrow 0$ V, which would be an incorrect logic value. The hold time is determined by the circuitry. In general, this consideration places a minimum frequency stipulation on the clock.

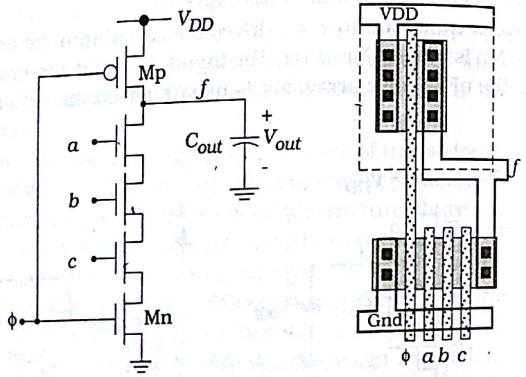
A dynamic NAND3 circuit is shown in Figure 9.18(a). Logic formation is achieved using the three series-connected FETs. The output

$$f = \overline{a \cdot b \cdot c} \tag{9.24}$$

is valid only during the evaluation period when $\phi = 1$. Layout is straightforward as shown by the example in Figure 9.18(b). Since the evaluation nFET Mn is in series with the logic block, C_{out} must discharge through four transistors. Increasing the sizes of the nFETs will reduce the fall time.

As mentioned above, charge leakage reduces the voltages held on the output node when $f = 1$. A detailed analysis of the circuit shows that another problem called **charge sharing** can occur when the clock makes the transition to $\phi \rightarrow 1$. It has the effect of reducing the output voltage even before charge leakage effects become noticeable.

The origin of the charge sharing problem is the parasitic node capacitance C_1 and C_2 between FETs as shown in Figure 9.19. The clock has been set at $\phi = 1$ so that Mp is off, isolating the output node from the power supply. The initial voltage on C_{out} at the start of the evaluation



(a) Circuit (b) Layout

Figure 9.18 Dynamic logic gate example

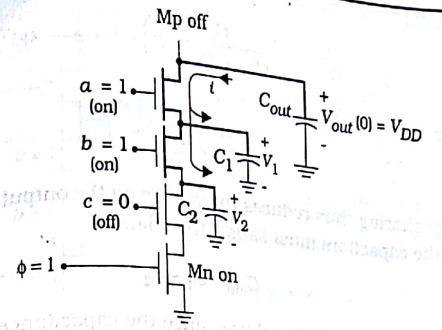


Figure 9.19 Charge sharing circuit

interval is $V_{out} = V_{DD}$ as shown. Assuming that the capacitor voltages V_1 and V_2 are both 0 V at this time, the total charge on the circuit is

$$Q = C_{out} V_{DD} \tag{9.25}$$

The worst-case charge sharing condition for this circuit is when the inputs are at (a, b, c) = (1, 1, 0). With c = 0, there is no discharge path to ground, so that the output voltage should remain high. However, since the a- and b-input FETs are on, C_{out} is electrically connected to C_1 and C_2 as indicated by the darkened lines. The current i flows because V_{out} is initially larger than V_1 or V_2. This corresponds to the transfer of charge from C_{out} to both C_1 and C_2. Using the relationship $Q = CV$ shows that V_{out} decreases while V_1 and V_2 increase. The current flow ceases when the voltages are equal with a final value

$$V_{out} = V_2 = V_1 = V_f \tag{9.26}$$

The total charge on the circuit is then distributed according to

$$\begin{aligned} Q &= C_{out} V_f + C_1 V_f + C_2 V_f \\ &= (C_{out} + C_1 + C_2) V_f \end{aligned} \tag{9.27}$$

Applying the principle of conservation of charge, this must be equal to the initial charge in the system:

$$Q = (C_{out} + C_1 + C_2) V_f = C_{out} V_{DD} \tag{9.28}$$

Solving for the final voltage gives

$$V_f = \left(\frac{C_{out}}{C_{out} + C_1 + C_2} \right) V_{DD} \tag{9.29}$$

Since

$$\left(\frac{C_{out}}{C_{out}+C_1+C_2}\right)<1 \tag{9.30}$$

we see that

$$V_f < V_{DD} \tag{9.31}$$

Charge sharing thus reduces the voltage on the output node. To keep V_{out} high, the capacitors must satisfy the relation

$$C_{out} \gg C_1 + C_2 \tag{9.32}$$

This may be difficult to achieve since the capacitance values are determined by the layout dimensions. After charge sharing takes place, the node is still subject to charge leakage, which continues to drop the voltage with time.

9.5.1 Domino Logic

Domino logic is a CMOS logic style obtained by adding a static inverter to the output of the basic dynamic gate circuit. The resulting structure is shown in Figure 9.20. The precharge and evaluate events still occur, but now it is the capacitor C_X between the dynamic stage and the inverter that is affected. A clock value of $\phi = 0$ defines the precharge. During this time, C_X is charged to a voltage $V_X = V_{DD}$ which forces the output voltage to $V_{out} = 0$ V. Inputs are valid during the evaluation interval when $\phi = 1$. If C_X holds its charge, V_X remains high and $V_{out} = 0$ V indicates a logic 0 output. If C_X discharges, then $V_X \rightarrow 0$ V and $V_{out} \rightarrow V_{DD}$. This corresponds to a logic 1 output.

Domino logic gates are **non-inverting** because of the output inverter. Two examples of this characteristic are shown in Figure 9.21. The AND gate in Figure 9.21(a) is easily understood: if $a = b = 1$, then the internal node discharges to 0 V, forcing the output to a logic 1 (V_{DD}). Similarly, the OR gate in Figure (9.21b) gives a 1 output if either $a = 1$ or $b = 1$. This

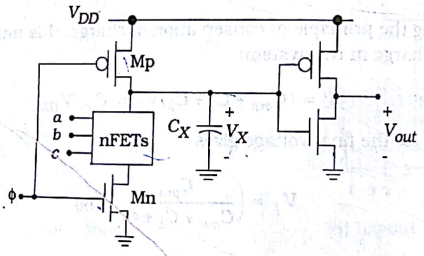
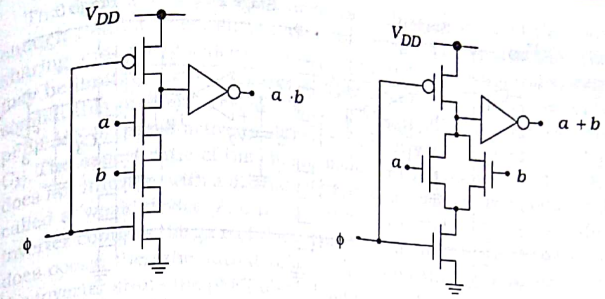


Figure 9.20 Domino logic stage



(a) AND gate (b) OR gate

Figure 9.21 Non-inverting domino logic gates

makes logic design using only domino gates somewhat tricky since the NOT operation is required for a complete set of logic operations.[1] While one can add inverters, it is found that this causes the possibility of introducing a hardware glitch into the circuit, and is usually avoided. Inverters are used only at the beginning or the end of a domino chain. An example of a domino layout is shown in Figure 9.22 for an AND3 gate. This is just a dynamic NAND3 circuit cascaded into a static inverter, so the layout preserves the features of general dynamic logic.

Domino logic derives its name from the manner in which a cascade operates. A 3-stage network is shown in Figure 9.23. Every stage is con-

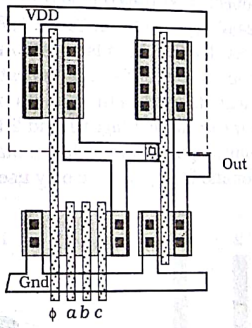


Figure 9.22 Layout for a domino AND gate

[1] A complete set of logic operations is one that is capable of producing any logic combination. Without the NOT operator, functions such as the XOR and XNOR are not possible.

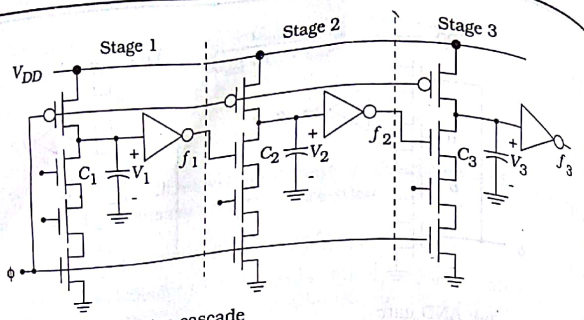


Figure 9.23 A domino cascade

trolled by the same clock phase ϕ. During a precharge event with $\phi = 0$, capacitors C_1, C_2, and C_3 are simultaneously charged to V_{DD}. This causes the outputs f_1, f_2, and f_3 to all be 0. When $\phi = 1$, the entire chain undergoes evaluation. In a domino cascade, this is like a "domino chain reaction" that must start at the first stage and then propagate stage by stage to the output. To understand this comment, suppose that we monitor the second stage output f_2 and see it switch from its precharge value $f_2 = 0$ to $f_2 = 1$ during the evaluation interval. The only way this could have happened is if C_2 discharged, but this requires that $f_1 = 1$ to turn on the nFET in the discharge chain. Applying the same logic to the first stage, f_1 can switch to 1 only if C_1 has discharged. Extending this argument, we see that $f_3 \rightarrow 1$ occurs only if both Stage 1 and Stage 2 have made the same transition.

The domino effect is portrayed in Figure 9.24 to help visualize the process. Figure 9.24(a) represents the precharge event by dominos standing on end. Evaluation for the chain is shown in Figure 9.24(b). A discharge event that gives an output of $f \rightarrow 1$ is indicated by a falling domino. This can topple the next stage, but other inputs may keep the discharge from taking place. In the drawing, Stages 1 and 2 have undergone a discharge, but Stage 3 remain high (in its precharge state). Note that the operation indicates that domino logic gates are only useful in cascades.

(a) Precharge (b) Evaluate

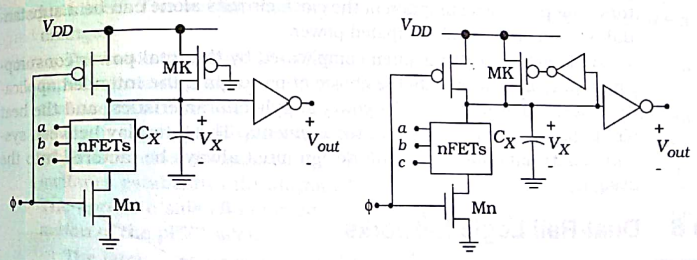
Figure 9.24 Visualization of the domino effect

The domino cascade must have an evaluation interval that is long enough to allow every stage time to discharge. This means that charge sharing and charge leakage processes that reduce the interval voltage V_X may be limiting factors. **Charge-keeper** circuits have been developed to combat this problem. Two are shown in Figure 9.25. In Figure 9.25(a), a pFET MK is biased active to allow a small current to replenish charge on C_X. The aspect ratio of the charge-keeper FET must be small so that it does not interfere with a discharge event in any significant manner; this is called a 'weak' device. Another approach is shown in Figure 9.25(b). An inverter controls the gate of the weak pFET. If an internal discharge of C_X does occur, then the output voltage V_{out} increases. Feeding this through the inverter shuts the pFET off and allows the discharge to continue.

An interesting extension of the basic domino circuit is that of **Multiple-Output Domino Logic (MODL)**. This type of circuit allows two or more outputs from a single logic gate, making it quite unique. The structure of a 2-output MODL stage is shown in Figure 9.26. The logic array has been split into two separate blocks denoted as F and G, which creates an additional output node. Adding an inverter and a precharge transistor results in the two outputs

$$f_1 = G$$
$$f_2 = F \cdot G \qquad (9.33)$$

This is easily understood by studying the logic network. If the G-logic block acts like a closed switch, then it produces an output of $f_1 = G$. If this occurs, then it is possible for the second logic block F to induce a discharge by also acting as a closed switch. This dependence produces the ANDing relation between the two outputs. While this is quite restrictive, the nesting of the AND operation does appear in several important computational algorithms such as the carry look-ahead adder.



(a) Single-FET charge keeper (b) Feedback controlled keeper

Figure 9.25 Charge-keeper circuits

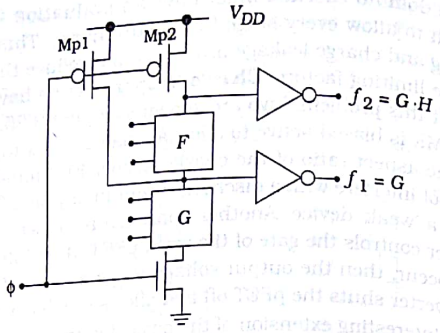


Figure 9.26 Structure of a MODL circuit

9.5.2 Power Dissipation of Dynamic Logic Circuits

CMOS dynamic logic circuits can be designed to provide very fast switching with modest real estate consumption. They have been successfully used in several well-known chips and are the basis of DRAMs and other important computer components. Unfortunately, they can be quite power hungry which may limit their usage.

In a dynamic circuit, the clock ϕ defines the precharge and evaluate operations in every cycle. Since charge cannot be held on a capacitive node, every precharge cycle will pull current from the voltage source, adding to the overall power dissipation of the circuit. The clock circuits themselves require dynamic power to drive the FETs. In the standard configuration, every stage presents a capacitance of

$$C_L = C_{Gp} + C_{Gn} \tag{9.34}$$

to the clock drivers corresponding to the precharge and evaluate transistors. The power consumption of the clock circuits alone can be a substantial portion of the total dissipated power.

VLSI system design is often complicated by the total power consumption of a chip. This affects the choice of packaging, the intended application (desktop or portable), the power supply characteristics, and the heat sinking and cabinet ventilation requirements. The interplay between system constraints and the circuit design must always be factored into the design.

9.6 Dual-Rail Logic Networks

We have been concentrating on **single-rail** logic circuits where the value of a variable is either a 0 or a 1 only. In **dual-rail** networks, both the variable x and its complement $\bar{x}$ are used to form the difference

$$f_x = (x - \bar{x}) \tag{9.35}$$

Using the quantity f_x provides an increase in the switching speed. This can be seen by calculating the time derivative as

$$\frac{df_x}{dt} = \left(\frac{dx}{dt} - \frac{d\bar{x}}{dt}\right) \tag{9.36}$$

and noting that

$$\frac{d\bar{x}}{dt} \approx -\left|\frac{dx}{dt}\right| \tag{9.37}$$

since x increases while $\bar{x}$ decreases, and vice versa. Thus

$$\frac{df_x}{dt} \approx 2\left|\frac{dx}{dt}\right| \tag{9.38}$$

so that the rate of change of f_x is approximately twice that of a single variable. Translated into logic terms, this means that the switching speed is almost twice as fast as can be obtained in a single-rail circuit.

The complicating factor in dual-rail circuits is the increase in circuit complexity and wiring overhead. Every input and output is now a doublet consisting of the variable and its complement. The circuits are correspondingly more complicated, and can be tricky to deal with. However, the speed advantage makes them worth studying. Some even provide structured and compact layout schemes.

9.6.1 CVSL

Most dual-rail CMOS circuits are loosely based around **differential cascode voltage switch logic**, which goes under the acronyms **DCVS logic** or **differential CVSL**; we will adopt the latter one here. CVSL provides for dual-rail logic gates that have latching characteristics built into the circuit itself. The output results f and $\bar{f}$ are held until the inputs induce a change.

The basic structure of a CVSL logic gate is shown in Figure 9.27. The input set consists of the variables (a, b, c) and their complements ($\bar{a}$, $\bar{b}$, $\bar{c}$) that are routed into an nFET 'logic tree' network. The logic tree is modeled as a pair of complementary switches Sw1 and Sw2 such that one is closed while the other is open as determined by the inputs. The state of the switches establishes the outputs. For example, if Sw1 is closed then $f = 0$. The opposite side ($\bar{f}$) is forced to the complementary state ($\bar{f} = 1$) by the action of the pFET latch.

The latch is controlled by the left and right source-gate voltages V_l and V_r shown in the drawing. Suppose that Sw2 is closed, forcing $\bar{f} = 0$ on the right side. In this case,

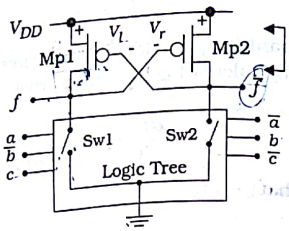


Figure 9.27 Structure of a CVSL logic gate

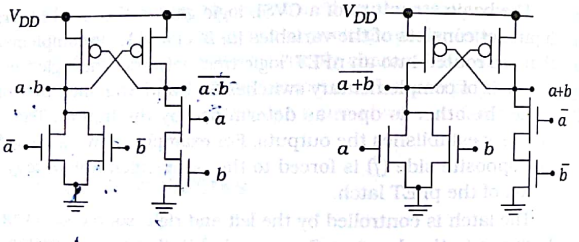
$$V_l = V_{DD} \tag{9.39}$$

which turns on Mp1. With Mp1 conducting, the left output node sees a path to the power supply, giving V_{DD} there; this is the $f = 1$ state. The ability to set the latch using a pull down on one side helps make the stage react quickly.

Several techniques have been published for designing the logic network. A straightforward approach is to use separate circuits for the left and right sides. Figure 9.28(a) is an AND/NAND circuit that has inputs of (a, b) on the right and $(\bar{a}, \bar{b})$ on the left; it is important to remember that dual-rail logic gates require pairs of complementary inputs and outputs. The formation of the NAND operation on the right side uses series-nFETs which is identical to nFET logic in standard CMOS. To obtain the left circuit, we simply use the DeMorgan identity

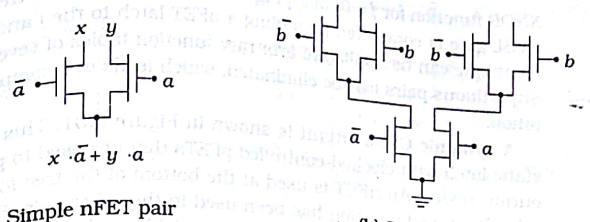
$$\overline{a \cdot b} = \bar{a} + \bar{b} \tag{9.40}$$

which, from our study of bubble pushing, indicates parallel nFETs with complemented inputs. An OR/NOR circuit is drawn in Figure 9.28(b). The logic formation follows the same approach as for the AND/NAND circuit.



(a) AND/NAND (b) OR/NOR

Figure 9.28 CVSL gate examples



(a) Simple nFET pair (b) Stacked pairs

Figure 9.29 nFET logic pairs

A more important observation is that the OR/NOR and AND/NAND gates are identical in form; only the locations of the inputs are different. This symmetry is due to the fact that OR and AND are logical duals.

Logic trees provide a more structured approach to designing the switching network. These are based on pairs of nFETs that are driven by complementary inputs as shown in Figure 9.29(a). With x and y applied to the top of the pair, the pair acts like a 2:1 MUX with a (bottom) output of

$$x \cdot \bar{a} + y \cdot a \tag{9.41}$$

Qualitatively, this says that x is transmitted if $a = 0$, while the output is y if $a = 1$. The pair $(\bar{a}, a)$ thus corresponds to an input pattern of (0,1) which is the same way that input combinations are listed in a function table. If $x = y$, then the output is always x and the FETs can be eliminated. A 2-level stack of nFET pairs is shown in Figure 9.29(b). The b-input pairs on the upper row correspond to the input sequence (01) (01), while the bottom pair (a-inputs) has the sequence (01). This provides a one-to-one mapping from a 2-input function table to the nFET arrays.

An example is the gate in Figure 9.30. The output f of the truth table has the sequence (1001) indicating the XOR function for $f = 1$, and the

f	1	0	0	1
b	0	1	0	1
a	0	0	1	1

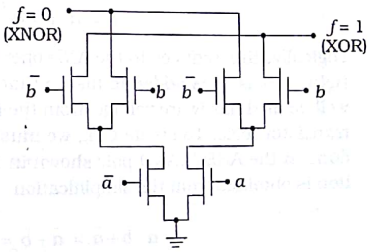
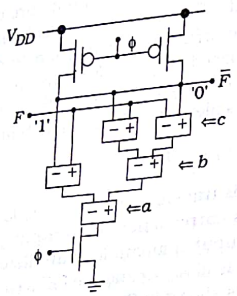


Figure 9.30 Example of a logic tree using nFET pairs

XNOR function for $f = 0$. Mapping the table gives the logic tree shown. The CVSL gate is completed by adding a pFET latch to the f and $\bar{f}$ lines. This technique can be applied to arbitrary function tables of several variables. Superfluous pairs can be eliminated, which leads to a compact representation.

A dynamic CVSL circuit is shown in Figure 9.31. This replaces the static latch with clocked-controlled pFETs that are used to precharge the output nodes. An nFET is used at the bottom of the tree for the evaluation. Simplified notation has been used in the schematic. Each '−+' box corresponds to an nFET pair with the variable applied to the '+' side, and the complement to the '−' side. Two reductions have been made translating the function table to the logic tree. This is because the left entries for f have the sequence 00 11, which allows both c-level pairs to be eliminated.



F	0	0	1	1	0	1	1	0
c	0	1	0	1	0	1	0	1
b	0	0	1	1	0	0	1	1
a	0	0	0	0	1	1	1	1

Figure 9.31 Dynamic CVSL circuit with 3-level logic tree

9.6.2 Complementary Pass-Transistor Logic

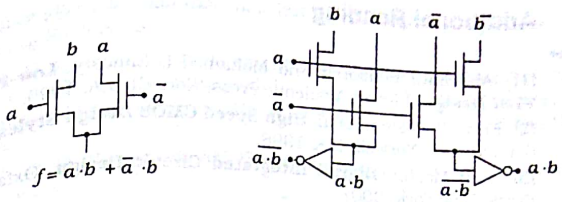
Complementary pass-transistor logic (**CPL**) is an interesting dual-rail technique that is based on nFET logic equations. Let us examine the nFET pair in Figure 9.32(a). The output is given by

$$f = a \cdot b + \bar{a} \cdot a \tag{9.42}$$

Logically, this reduces to the AND operation $f = a \cdot b$ since $\bar{a} \cdot a = 0$. The right transistor is added to insure that the output $f = 0$ when $a = 0$ is a well defined hardware voltage (from the input a). This is the basis of pass-transistor logic. To create CPL, we must add the NAND function. This is done in the AND/NAND pair shown in Figure 9.32(b). The NAND operation is obtained from the simplification

$$a \cdot \bar{b} + \bar{a} = \bar{a} + \bar{b} = \overline{a \cdot b} \tag{9.43}$$

Since nFETs suffer from threshold losses, static output inverters have



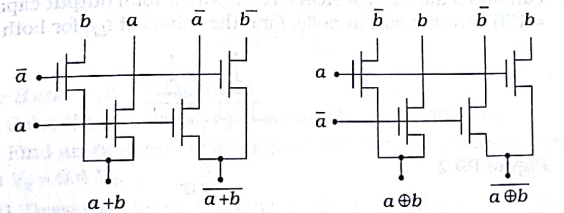
(a) AND gate (b) AND/NAND array

Figure 9.32 CPL AND/NAND circuit

been added to restore the voltages to full-rail values. These are not necessary until the full power supply is required, but they also help to speed up the circuit.

A unique feature of CPL is that several 2-input gates can be created by using the same transistor topology with different input sequences. Figure 9.33(a) shows an OR/NOR array. Comparing this with the AND/NAND shows that we have simply switched a and $\bar{a}$ on the FET inputs. An XOR/XNOR pair is shown in Figure 9.33(b). This is achieved by changing the top (drain) inputs. CPL also allows for 3-input logic gates with similar properties.

CPL is an interesting approach because it provides compact logic gates and the cell layout is reusable. The main drawbacks are the threshold loss and the fact that an input variable may have to drive more than one FET terminal. Similar approaches designed to overcome these problems have been proposed in the literature, but all result in more complex circuits.



(a) OR/NOR (b) X OR/XNOR

Figure 9.33 2-input CPL arrays

System Specifications Using Verilog® HDL

10

Hardware description languages (HDLs) are an ideal vehicle for hierarchical design. A system can be specified from the highest abstract architectural level down to primitive logic gates and switches.

Two HDLs dominate the field: VHDL (VHSIC HDL)[1] and Verilog® HDL. VHDL started as a government effort to unify projects from different contractors, while Verilog was the result of private development. Both are now standardized and widely used in industry, so either one could be presented here. Verilog was chosen because of its popularity in VLSI design. Compared to VHDL, it is a relatively loose and free-flowing language, and most chip designers feel that it adheres to their way of thinking. Verilog is structured after the C programming language and uses similar procedures and constructs. We should note, however, that C or C++ themselves can be used as an HDL [9], and several companies develop their own language.

This chapter introduces the basic concepts of the Verilog language. If you are familiar with VHDL from another course, you will find that learning Verilog is straightforward. If this is your first trek into an HDL, don't worry; the road is smooth and the ideas are easy to master.

10.1 Basic Concepts

A hardware description language allows us to specify the components that make up a digital system using words and symbols instead of having to use a pictorial representation like a block or logic diagram. Every component is defined by its input and output ports, the logic function it per-

[1] VHSIC is a DoD acronym for Very High-Speed Integrated Circuits; DoD is an acronym for the Department of Defense.

forms, and timing characteristics such as delays and clocking. An entire digital system can be described in text format using a prescribed set of rules and **keywords** (reserved words). The file is then processed with the language compiler, and the output can be analyzed for proper operations. This can be applied to simple logic gates or to an entire microprocessor design. Logic verification using an HDL is usually considered mandatory to validate the design.

A typical design hierarchy is portrayed in Figure 10.1. At the highest level is a **behavioral** description that describes the system in terms of its architectural features. This is generally quite abstract in that it does not contain any details on how to implement the design. Once the behavioral model is simulated and refined, the design moves down to the **register-transfer level (RTL)**. An RTL description of a digital network concentrates on how the data moves about the system from unit to unit, and the main operations. State machines and sequential circuits can be introduced at this level. Timing windows are checked and rechecked, and validation of the design is again a primary objective.

The next level in the design process is called **synthesis**. In fully automated design, the RTL description is sent through a synthesis tool that produces a netlist of the hardware components needed to actually build the system. One of the more popular synthesis tools is Synopsis®. The success or failure of the synthesis process often depends upon the skill of the code writer. Not all HDL constructs can be synthesized, with a typical estimate hovering somewhere around 50%.

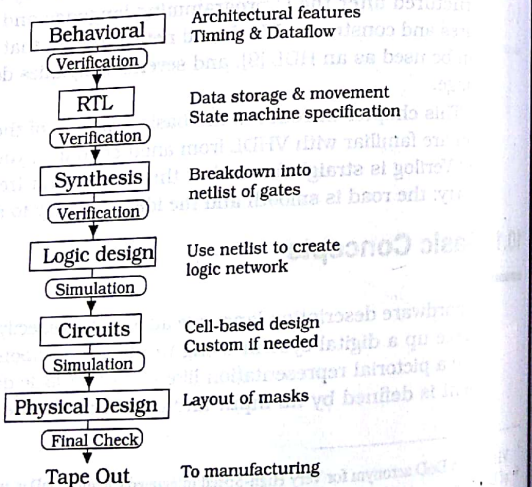


Figure 10.1 Example of a VLSI design flow

After the synthesis step, the netlist is used to design the logic network. Verification at this level consists of simulations to insure that the logic is correct. Once the logic is validated, the cell library can be used to design the circuits. Components are wired together, and both the electrical characteristics and the logic are verified using simulation. The cell instances and wirings are translated into silicon patterns in the physical design phase. After verifying the layout, the design is (at last!) complete and sent to manufacturing for the first silicon test chip.

Verilog HDL provides for descriptions of a digital system at all of the levels listed above. Every level is related to every other level, and the hierarchical design philosophy is linked by the different types of code. Each level has its own coding style using certain sets of commands and constructs. Verilog even provides for switch modeling of MOSFETs, although it is not as robust and sensitive to the CMOS processing variables as a circuit simulator such as SPICE. Verilog-A is an extension of an intrinsically digital language to the analog world.

The concept that links the various levels is that of a **module**. A Verilog module is the description of a unit that performs some function. It may be as simple as a basic FET switch, or as complex as a 64-bit ALU. Instantiations of simple modules are used to create more complex modules. The hierarchical structure is analogous to that used in the design of cells in a layout editor that was discussed earlier in the book.

Our treatment of Verilog will start at the digital logic level where simple gates are used to build more complex logic units. Once the structure of the language is understood, higher levels of abstraction are introduced.

10.2 Structural Gate-Level Modeling

Structural modeling describes a digital logic network in terms of the components that make up the system. Gate-level modeling is based on using primitive logic gates and specifying how they are wired together. It is the easiest to learn since it parallels the ideas developed in elementary logic.

Verilog is built using certain keywords that are understood by the compiler. Included in the group are primitives (such as logic gates), signal types, and commands. In our listings, Verilog code will use a sans serif font of this type, and will be indented from the main text. Keywords will be **boldface** using the same font. At the structural modeling level, the keywords are often primitive logic operations (gates) which results in a very readable coding style. A straightforward approach to learning Verilog is to study how a logic network is translated into a Verilog description using a line-by-line analysis. This will illustrate the ideas and syntax in a direct manner.

10.2.1 Verilog by Example

Consider the 4-input AOI circuit shown in Figure 10.2. The logic is constructed using primitive AND and NOR gates that take the inputs a, b, c, and d and produce an output of

$$f = \text{NOT}(a \cdot b + c \cdot d) \tag{10.1}$$

Let us examine the listing for the Verilog module that describes the network by its internal structure. We will then study the details to learn how the module was constructed.

```
module AOI4 (f, a, b, c, d) ;
    input a, b, c, d ;
    output f ;
    wire w1, w2 ;
    and G1 (w1, a, b ) ;
    and G2 (w2, c, d ) ;
    nor G3 (f, w1, w2) ;
endmodule
```

A first reading of the listing exhibits the structure and syntax of a Verilog module. The keyword **module** defines the start of the listing for a network that has the name AOI4. The last line of the listing **endmodule** indicates that the description of the module is complete. The names of output and input "identifiers" are then listed in parentheses, with the output f first and then the inputs a, b, c, d. Semicolons are used as delimiters in Verilog; their usage should be memorized.

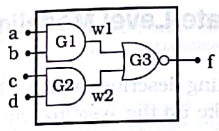


Figure 10.2 AOI module example

The next group of lines are the port keywords **input** and **output** that identify the input and output variables. The **wire** keyword identifies w1 and w2 as internal values that are needed to describe the network, but are not **input** or **output** ports. A **wire** declaration is a datatype called a net. A net value is determined by the output of the driving gate. In this case, w1 and w2 are the outputs of AND2 gates, which are in turn determined by the input values.

The structure of the logic is specified by the next three lines. These are instances of primitive AND and NOT gates that are part of the Verilog language. A gate instance has the form

```
gate_name instance_name (out, in_1, in_2, in_3, ... ) ;
```


where instance_name is an optional specifier that is used to correlate gates to their listing. In our example, we have named the gates G1, G2, and G3, so these appear in the listing. The compiler will interpret the code in the same manner if these are left out.

A structural listing provides a unique one-to-one correspondence with the components of a logic network. Suppose that we start with the following module description and then construct the logic diagram from it.

```
module Example (s_out, c_out , in_0, in_1 ) ;
    input in_0, in_1 ;
    output s_out, c_out ;
    xor (s_out, in_0, in_1) ;
    and (c_out ,in_0, in_1 ) ;
endmodule
```

This results in the internal details shown in Figure 10.3. This was drawn by starting with the input ports for in_0 and in_1, adding the gates (**xor** and **and**) with the specified wiring, and then pulling the outputs (s_out, and c_out) from the central region of the module. The logic equations are

$$\text{s\_out} = (\text{in\_0}) \oplus (\text{in\_1})$$

$$\text{c\_out} = (\text{in\_0} \cdot \text{in\_1})$$

which is recognized as the sum and carry-out of a half-adder. These examples illustrate the fact that a Verilog structural description is equivalent to the information contained in a standard logic diagram.

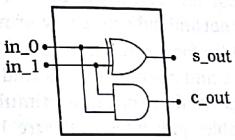


Figure 10.3 Logic network from the Verilog listing

Before proceeding further, let us examine some of the basics of writing Verilog descriptions.

Identifiers

Identifiers are names of modules, variables, and other objects that we can reference in the design. Examples of identifiers used so far include AOI4, a, b, in_0 , and s_out. Identifiers consist of upper- and lowercase letters, digits 0 through 9, the underscore character (_), and the dollar sign ($). The first character must be a letter or the underscore in normal usage. An identifier must be a single group of characters. For example, input_control_A is a single object, but input control A is not allowed as a single identifier.

It is important to point out that the Verilog language is **case sensitive**.

Figure 10.10 Testbench concept

```
A = 1 ; B = 0 ; C = 0 ;
```

These are interpreted as default binary values. Values can also be specified in base-*r* for radix values of 2 (binary, b), 8 (octal, o), 10 (decimal, d), and 16 (hexadecimal, h) using a format of

```
<size> '<base designator> <value>
```

with <size> a decimal number indicating the number of bits in the number. Some examples are

```
1'b0          // 1-bit binary number with a value of 0
4'b1011       // 4-bit binary word with a value of 1011
16'h1a36      // 16-bit number with a value of hexadecimal 1a36
3'd4          // 3-bit number with a decimal value of 4 = 100₂
```

Values can be declared in a listing. For example, the code

```
reg reset ;
initial
  begin
  reset = 1'b1 ;  // initialize reset to a value of 1
  #10 reset = 1'b0 ; // reset to 0 after 10 time units
  end
```

allows us to specify the value of reset as required.

10.3 Switch-Level Modeling

Verilog allows switch-level modeling that is based on the behavior of MOSFETs. Although circuit-level simulators (such as SPICE) are much more accurate for performing critical electrical calculations, Verilog coding is useful for verifying logic flow through networks that consist of both transistors and logic gates. More importantly, switch-level models have a direct one-to-one correspondence with CMOS circuits and logic gates as discussed in Chapter 2. The ability to construct Verilog descriptions of complex system-level designs all the way down to basic CMOS circuits demonstrates the power of hierarchical design.

The switch primitives are named **nmos** and **pmos**, and behave in the same manner as the transistors with the same names. Figure 10.11 summarizes the behavior of both. Verilog syntax for these primitives is in the

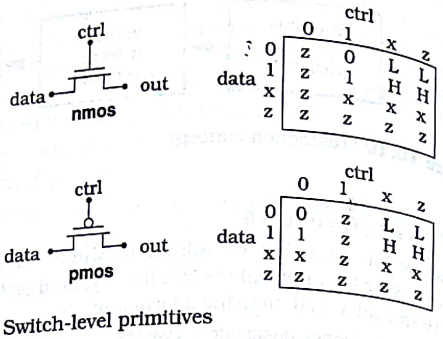


Figure 10.11 Switch-level primitives

form

```
nmos name (out, data, ctrl) ;
pmos name (out, data, ctrl) ;
```

where name is the optional instance identifier. For ctrl values (applied to the gate) that are 0 **and** 1, the behavior is identical to FETs. The **nmos** switches are open for ctrl = 0 and closed for ctrl = 1, while **pmos** switches are closed for ctrl = 0 and open for ctrl = 1. An open switch induces a high-impedance state with out = z. The tables also list two new entries, L and H, for the value of out when ctrl is x or z. The (low) symbol L stands for 0 or z, while the (high) symbol H represents 1 or z. The basis of this ambiguity is non-trivial. It is related to the physical concept that the output node can store charge, so that out may be related to an earlier value.

MOS switches can be used to describe CMOS logic gates. The simple NOT circuit in Figure 10.12 has the Verilog description

```
// CMOS inverter switch network
module fet_not (out, in) ;
   input input ;
   output output ;
   supply1 vdd ;
   supply0 gnd ;
   pmos p1 (vdd, output, input) ;
   nmos n1 (gnd, output, input) ;
endmodule
```

The circuit and listing has been used to introduce two new Verilog keywords **supply1** and **supply0** that define the power supply vdd and ground gnd connections. These represent the strongest logic 1 and logic 0 drivers, respectively. The Verilog module treats these as the data into the FETs, while the gate input is the switch ctrl.

The same constructs can be used to model arbitrary CMOS logic gates.

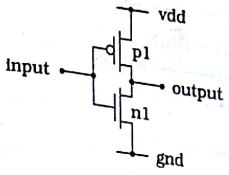


Figure 10.12 CMOS inverter using Verilog switches

The NAND2 and NOR2 switching networks in Figure 10.13 are described by the module

```
// CMOS logic gates
module fet_nand2 (out, in_a, in_b) ;
   input in_a, in_b ;
   output out ;
   wire wn ; // This wire connects the series nmos switches
   supply1 vdd ;
   supply0 gnd ;
   pmos p1 (vdd, out, in_a) ;
   pmos p2 (vdd, out, in_b) ;
   nmos n1 (gnd, wn, in_a ) ;
   nmos n2 (wn, out, in_b ) ;
endmodule
```

for the NAND gate, and

```
module fet_nor2 (out, in_a , in_b ) ;
   input in_a, in_b ;
   output out ;
   wire wp ; // This connects the series pmos switches
   supply1 vdd ;
   supply0 gnd ;
   pmos p1 (vdd, wp, in_a) ;
   pmos p2 (wp , out, in_b) ;
   nmos n1 (gnd, out, in_a ) ;
   nmos n2 (gnd, out, in_b ) ;
endmodule
```

for the NOR gate. These can be verified using a line-by-line comparison.

Another useful set of primitives includes pull-up and pull-down components that have the keywords **pullup** and **pulldown**. These can be modeled as resistors that are connected to **supply1** and **supply0** as shown in Figure 10.14(a) and are described by

```
pullup (out_1) ; // This gives a high output
```

HDLs provide a powerful vehicle for system-level design by introducing different levels of abstraction. The highest Verilog level is called **behavioral** modeling. As implied by its name, it concentrates on describing the general *behavior* of units to characterize how they will work when embedded in a larger system. Timing is often the most critical feature in a behavioral model. The internal details of a unit are not specified, nor do they affect the modeling; it is assumed that the specifications are a result of physically realizable internal circuitry.

The next level of abstraction down is usually termed Register-Transfer Level (**RTL**) modeling. RTL concentrates on specifying the movement of data among hardware sections. The name itself arises from the fact that synchronous digital systems rely very heavily on the use of clock-controlled storage registers. Data transfers take place at specific times dictated by the clocking. An RTL specification is viewed as being the link between purely abstract modeling and hardware design. RTL code is often the input to the synthesis stage of design (see Figure 10.1) that produces gate netlists.

The remaining section of this chapter is an introduction to high-level behavioral modeling in Verilog. The treatment covers the basics of behavioral and RTL coding with short examples to clarify the structure and concepts. Advanced constructs and coding techniques are introduced for specific applications in later chapters.

10.5 Behavioral and RTL Modeling

Verilog behavioral modeling is based on specifying a group of concurrent procedures that characterize a block. Emphasis is on an accurate representation of the architecture, with most of the implementation details ignored. This feature makes the coding style quite abstract.

The basis for behavioral modeling is the construction of **procedural blocks**. As implied by its name, a procedural block is a listing of statements that describe how a set of operations are performed. Many of these resemble constructs in the C programming language, and they introduce a new level of abstraction to the design process. Procedural blocks contain assignment statements, high-level constructs such as loops and conditional statements, and timing controls. There are two types of block that start with the keywords **initial** and **always**. An **initial** block executes once in the simulation and is used to set up initial conditions and step-by-step dataflow. An **always** block executes in a loop and repeats during the simulation. Block statements are used to group two or more statements together. Sequential statements are inserted between the keywords **begin** and **end**. It is also possible to write concurrently executed statements using the **fork** and **join** keywords.

Let us start by writing a module for a clock variable clk. We will assume

a clock period of 10 time units so that the variable must change every 5 time units as illustrated in Figure10.21.

```
module clock ;
reg clk ;
// The next statement starts the clock with a value of 0 at t = 0
  initial
    clk = 1'b0 ;
// When there is only one statement in the block, no grouping is required
  always
    #5 clk = ~ clk ;
  initial
    #500 $finish ; // End of the simulation
endmodule
```

The cyclic action is obtained using the NOT operator ~ in the statement

```
# 5 clk = ~ clk ;
```

Since this falls within an **always** statement the command is executed in a loop until the simulation ends at 500 time units.

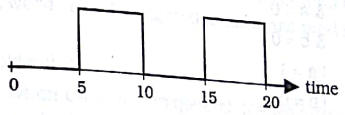


Figure 10.21 Clocking waveform clk

Operators

The Verilog operators such as ~ are summarized in Figure 10.22 for future reference. Note that some symbols such as & are used differently depending upon the context. We will study a few to understand how they work.

Consider first the behavior of the reduction or unary operators (i.e., operations on a single number.) Suppose we assign the binary values a = 1101 and b = 0000. Bit-wise negation gives

```
~ a = 0010
~ b = 1111
```

as it operates on each bit independently. A logical negation evaluates to

```
! a = 0
! b = 1
```

The logical operator !A gives the logical inverse of A. If A contains all zeros, then it is false (0). If it is non-zero, then it is true (1); !A gives the inverse of the value of A. Reduction operators operate on each bit of the number

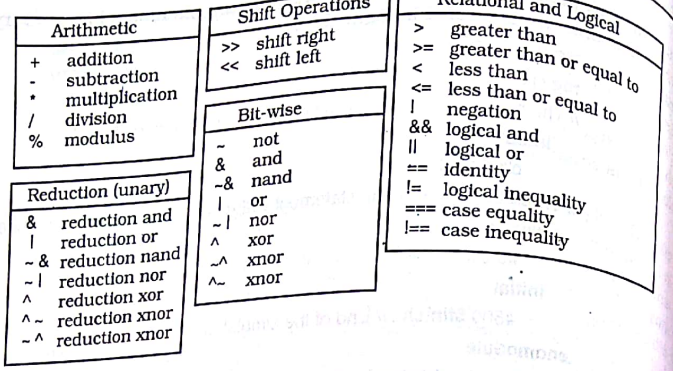


Figure 10.22 Verilog operators

and result in a single bit true (1) or false (0) value. For example, with a and b defined as previously stated,

```
& a = 0
& b = 0
| a = 1
| b = 0
^ a = 1
^ b = 0
```

The symbol '|' used for the OR is called a **pipe**.

The next group are the binary operators that have two operands. These are used in both bit-wise and logical contexts. With a = 1010 and b = 0011, the bit-wise application of the operators acts in a bit-by-bit manner:

```
a & b = 0010
a | b = 1011
a ^ b = 1001
```

In a logical context, the answer is a single true (non-zero) or false (all zeros) number.

```
a && b = 1
a || b = 1
a && c = 0
```

where c = 0000.

The equality operators are =, ==, and ===. The assignment operator = is used to copy the value from the right side of an expression to the left side as in

```
a = 4'b1010
```


The equality operator == is used in

```
a == b
```

to express "a is equal to b." Identity is written as

```
c === d
```

which says that "c is identical to d."

Timing Controls

Timing controls statements dictate the times when actions take place. There are three types of timing controls that are used in a procedural block. A simple delay is specified using # <time> as the clock example. An edge-triggered control is of the form @ (signal). In the statement

```
@(posedge clk) reg_1 = reg 2 ;
```

the **posedge** keyword is used to induce the assignment when the clock clk is rising from a 0 to a 1, or from x or z to 1. The positive edge of the clock is shown in Figure 10.23. Similarly, a negative-edge triggered event can be described by a statement of the form

```
@(negedge clk) output = a_in ;
```

A negative edge transition is from a 1 to a 0, or from x or z to 0. Edge-triggered statements can include the possibility of several signals changing by using the **or** keyword. Level-triggered events are modeled using the **wait** keyword.

```
wait (clk) q_out = d_in ;
```

effects the transfer when clk is a 1. In general, the **wait** directive executes when the expression is logically TRUE (i.e., non-zero).

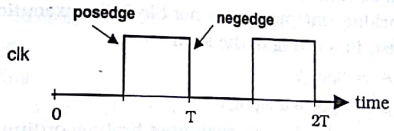


Figure 10.23 Clock edges

Procedural Assignments

A procedural assignment is used to change or update the values of reg and other variables. They are usually divided into **blocking** and **non-blocking** assignments.

Blocking assignments are executed in the order that they are listed, and allow for straightforward sequential and parallel blocks. The assignment operator " = " is used in these statements. Consider the simple code listing

```
reg a, b, c, reg_1, reg_2 ;
initial
```

General VLSI System Components

11

VLSI systems design revolves around a library of component functions. Primitive entries include FETs and basic logic gates, but higher level functions are needed to build the system hierarchy. In this chapter we will study a few examples of system components that are used to build large-scale systems and are commonly found in a VLSI cell library. The list of cells presented here is not comprehensive; additional components will be introduced in later chapters. Instead, the approach is intended to emphasize the connection between a high-level architectural specification and the resulting circuits and silicon implementation.

11.1 Multiplexors

Multiplexors are indispensable in modern digital design. A MUX consists of n input lines and one output f. The main function of the component is to use an m-bit select word to connect one of the inputs to the output. To cover every input, we must choose m such that $n = 2^m$. An alternate way to specify this is to introduce the base-2 logarithm such that $\log_2(2) = 1$. Then, using

$$m = \log_2(2^m) \tag{11.1}$$

we can say that

$$m = \log_2(n) \tag{11.2}$$

gives the number of select lines.

The simplest example is a 2-to-1 multiplexor. There are several ways to describe the component. A behavioral description using the **case** statement was presented in the previous chapter, and is repeated here for ref-

erence. The input lines are designated as p0 and p0, and the select bit is denoted by the identifier select.

```
module simple_mux (mux_out, p0, p1, select ) ;
input p0, p1 ;
input select ;
output mux_out ;
always @ (select )
   case ( select )
   1'b0 : mux_out = p0 ;
   1'b1 : mux_out = p1 ;
   endcase
endmodule
```

Given this description, several different logic and circuit implementations can be obtained. A gate-level NAND implementation is illustrated in Figure 11.1. Applying DeMorgan's theorem allows us to bubble-push to obtain the SOP (AND-OR) form

$$f = p_0 \cdot \bar{s} + p_1 \cdot s \tag{11.3}$$

Since a NAND2 gate requires four FETs, the entire network with driver can be implemented in 16 transistors.[1]

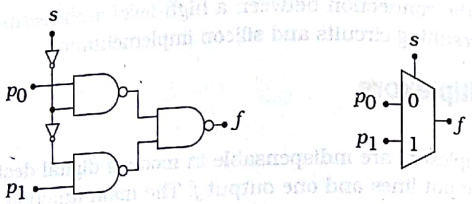
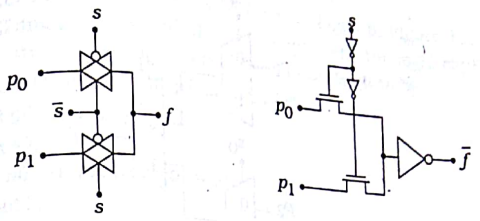


Figure 11.1 Gate-level NAND 2:1 multiplexor

The transmission-gate circuit in Figure 11.2(a) could also be a candidate in a CMOS technology. This circuit uses four FETs for the path logic (two for each TG). If we include a buffering NOT pair for the select bit, then the total FET count is increased to 8. The main problem with this circuit is that the TGs have parasitic resistance and capacitance that slow down the response. Figure 11.2(b) shows the same configuration using only nFET switches. With the select drivers, the FET count is reduced to 6. However, we have added an inverting driver at the output to compen-

[1] Note that this network can also be described using a structural listing.



(a) TG circuit (b) Pass transistors

Figure 11.2 Multiplexor using switch logic

sate for the fact that the nFET passes only a voltage range of 0 V to V_{max} where

$$V_{max} = V_{DD} - V_{Tn} \tag{11.4}$$

with V_{Tn} the threshold voltage. The inverter helps to restore the output to the full-rail range [0, V_{DD}]. Although the FET count is the same as for the TG circuit, the layout wiring will be easier. When translating the high-level description to silicon, considerations such as these become important factors.

Larger multiplexors can be designed using primitive gates or by instancing 2:1 devices. Consider a 4:1 MUX described by

```
module bigger_mux (out_4, p0, p1, p2, p3, s0, s1) ;
input p0 , p1, p2, p3 ;
input s0, s1 ;
output out_4 ;
assign out_4 = s1 ? (s0 ? p3 : p2 ) : (s0 ? p1 : p0) ;
endmodule
```

Since this is a high-level abstraction, no details about the internal structure are given. However, the **assign** statement can be interpreted as three 2:1 separate multiplexors with the first ? : using s1, and the second and third occurrences based on s0. This implies the structure illustrated in Figure 11.3. The select bit s0 is used to select (p0, p2) or (p1, p3) in the first stage devices. The final selection is achieved with s1 determining the actual output f which is the same as out_4 in the Verilog listing.

Another implementation is the gate-level construction shown in Figure 11.4. Using this as a guide, the equivalent Verilog structural description would be

```
module gate_mux_4 (out_gate, p0, p1, p2, p3, s0, s1) ;
input p0, p1, p2, p3 ;
input s0, s1 ;
```

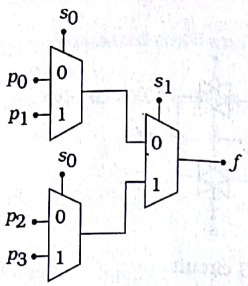


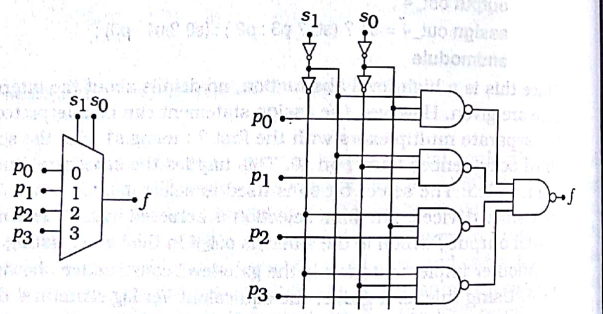
Figure 11.3 A 4:1 MUX using instanced 2:1 devices

```
    wire w1, w2, w3, w4 ;
    output out_gate ;
    nand (w1, p_0, ~s1, ~s0) ,
         (w2, p_1, ~s1, s0) ,
         (w3, p_2, s1, ~s0) ,
         (w4, p_3, s1, s0) ,
         (out_gate, w1, w2, w3, w4) ;
    endmodule
```

The NOT gates have been modeled using ~ operators, but they could have been instanced with primitive **not** gates with the same result. In standard logic, this is equivalent to the SOP expression

$$f = p_0 \cdot \overline{s_1} \cdot \overline{s_0} + p_1 \cdot \overline{s_1} \cdot s_0 + p_2 \cdot s_1 \cdot \overline{s_0} + p_3 \cdot s_1 \cdot s_0 \qquad (11.5)$$

obtained from applying basic logic.



(a) Symbol (b) Logic diagram

Figure 11.4 Gate-level 4:1 MUX

Yet another network is the pass-FET array shown in Figure 11.5. This uses the ANDing properties of nFETs to implement the logic expression directly. The structural description of this circuit is given by

```
    module tg_mux_4 (f, p0, p1, p2, p3, s0, s1) ;
    input p0, p1, p2, p3 ;
    input s0, s1 ;
    wire w0, w1, w2, w3, w_o , w_x ;
    output f ;
    nmos (p0, w0, ~ s1), (w0, w_o, ~ s0) ;
    nmos (p1, w1, ~ s1), (w1, w_o, s0) ;
    nmos (p2, w2, s1), (w2, w_o, ~ s0) ;
    nmos (p3, w3, s1), (w3, w_o, s0) ;
    not (w_x, w_o), (f , w_x ) ;
    endmodule
```

where the **nmos** instances have been grouped to make them easier to trace. The wires between the FETs in the i-th line are labeled wi for i = 0, 1, 2, 3 and w_o is the common output wire for the four paths. The wire between the inverters is w_x. A simple layout strategy for the cell is shown in Figure 11.6. The one-to-one correspondence is obvious, but the packing density could be improved by moving the FETs and rerouting.

The split-array nMOS/pMOS circuit shown in Figure 11.7 provides the function using similar reasoning. Every input sees a path to the output through both an nFET and a pFET chain. Since pFETs can pass logic 1 voltages while nFETs pass logic 0 voltages, the output has a full-rail swing from 0 V to V_{DD}. Output restoring buffers are not required in this case, but the circuit will be much faster if they are added. This arrangement uses the ideas of complementary pairs that are wired in a manner similar to TGs. However, since the nFETs and pFETs circuits are sepa-

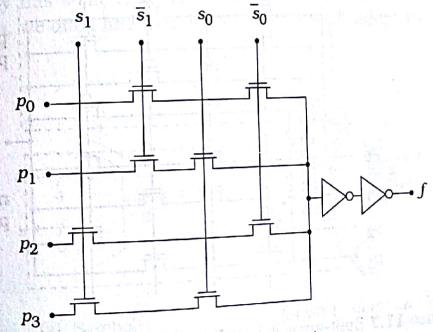


Figure 11.5 4:1 MUX using nFET pass transistors

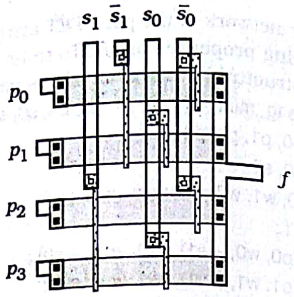


Figure 11.6 Simple 4:1 pass-FET MUX layout

rate, the interconnect wiring will be simpler than if TGs were used for every switch. These examples illustrate the variations that are possible in translating a high-level HDL construct to basic logic circuits.

Architectural specifications treat n-bit words in about the same manner as single-bit entities. Suppose that we have two 8-bit words

$$a = a_7a_6a_5a_4a_3a_2a_1a_0$$
$$b = b_7b_6b_5b_4b_3b_2b_1b_0 \quad (11.$$

that we want to use as inputs to a 2:1 MUX. The output

$$f = f_7f_6f_5f_4f_3f_2f_1f_0 \quad (11.$$

is determined by the select bit s such that

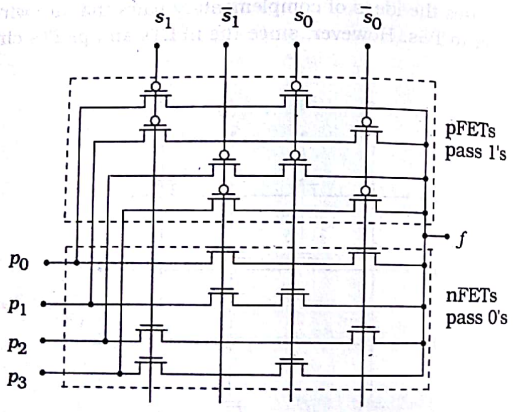


Figure 11.7 Split-array 4:1 MUX for full-rail output

$$f_i = a_i \cdot \bar{s} + b_i \cdot s \quad (11.8)$$

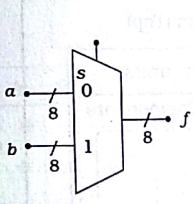
for $i = 0, \ldots, 7$. This of course implies that we should use 8 identical 2:1 MUXes that are all controlled by the same select bit s.

At the system level, we tend to treat a and b as single objects. The MUX symbol in Figure 11.8(a) identifies the width of the word using the slash notation (/) across the lines. A Verilog description can be written using 8-bit vectors as specified by the [7 : 0] in the listing

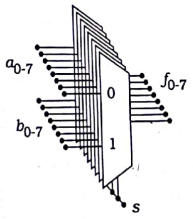
```
module mux_2-1_8b (f, a, b, s );
input [7 : 0] a, b ;
input s ;
output [7 : 0] f ;
  assign f = s ? b : a ;
endmodule
```

The extension to larger word sizes is accommodated by simply resizing the vectors. The bit-level implementation is more complicated, since we must take n parallel 2:1 MUX units as in Figure 11.8(b). At the physical level, every 1-bit MUX consumes area and is characterized by a set of delay times. To build the 8-bit network, we can tile eight identical cells as shown in Figure 11.9. Since silicon circuits (and all logic gates) are designed at the bit level, the layout area and wiring may become the limiting factor.

This discussion is intended to illustrate an important point. Given a high-level architectural description of a digital network, several choices can be made for the circuitry. Each choice leads to a distinct physical design with its own layout and switching characteristics. In a critical timing path the TG circuits may not be fast enough. In a different situation, a limited real estate allocation may make area more important. VLSI system design is not just writing good code or pushing polygons. Every level of the hierarchy has some effect on every other level. In a top-down design approach, we often find that the behavioral specifications can be imple-



(a) Symbol



(b) Bit-level realization

Figure 11.8 A vector 2:1 MUX

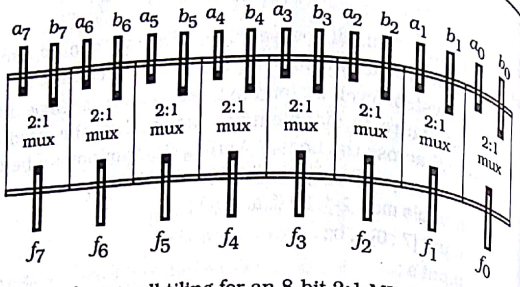


Figure 11.9 Single-bit cell tiling for an 8-bit 2:1 MUX

mented in several ways. Some are better than the others, but the choi depends upon the metrics as they are applied throughout the desig cycle.

This theme will be maintained throughout the chapter. Several log components will be analyzed at different levels to illustrate the connection between a high-level architectural description and the circuit or silicon network. While the components are important in their own right, the crit ical system-level features emerge only after the components are used t construct more complex logic units. The hierarchy is illustrated by the nesting diagram in Figure 11.10. In top-down design, we start with a set of specifications and design the high-level architectural model in an HDL. The progression downward to silicon eventually results in a silicon device, but the interaction among the various levels is linked in a critical manner. No system can operate faster than each subsequent level allows, while the silicon real estate budget is the bottom line limit. And, of course, we must be able to manufacture the chip and sell it at a price that users are willing to pay while still maintaining a profit margin!

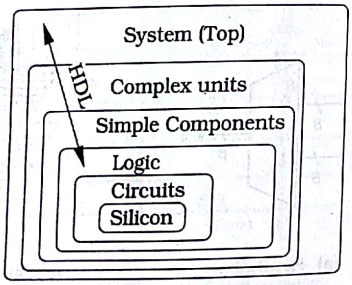


Figure 11.10 System-level hierarchy

11.2 Binary Decoders

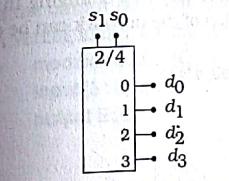
A binary n/m row decoder accepts an n-bit control word and activates one of the m-output lines while the other $(m - 1)$ lines are not affected. An **active-high** decoder sets a 1 on the selected line and keeps the others at 0. An **active-low** decoder is just the opposite, with the selected line reset to 0 while the remaining lines are at 1.

A 2/4 active-high decoder symbol and function table is shown in Figure 11.11(a). The 2-bit select word s_1s_0 activates the line corresponding to its specified decimal value 0, 1, 2, 3. The function table yields the equations

$$\begin{aligned} d_0 &= \overline{s_1} \cdot \overline{s_0} = \overline{s_1 + s_0} \\ d_1 &= \overline{s_1} \cdot s_0 = \overline{s_1 + \overline{s_0}} \\ d_2 &= s_1 \cdot \overline{s_0} = \overline{\overline{s_1} + s_0} \\ d_3 &= s_1 \cdot s_0 = \overline{\overline{s_1} + \overline{s_0}} \end{aligned} \tag{11.9}$$

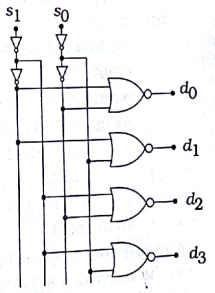
A straightforward NOR-gate implementation is shown in Figure 11.11(b). This gives the basis for the structural description

```
module decode_4 (d0, d1, d2, d3, s0, s1) ;
input s0, s1 ;
output d0, d1, d2, d3 ;
nor   (d3, ~s0, ~s1) ,
      (d2, ~s0, s1) ,
```



s_1	s_0	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(a) Symbol and table



(b) NOR2 implementation

Figure 11.11 An active-high 2/4 decoder

```
        (d1, s0, ~s1) ,
        (d0, s0, s1) ;
    endmodule
```

where we have absorbed the NOT drivers into the notation using the ~ operator.

An equivalent architectural description using **case** keywords can be written as

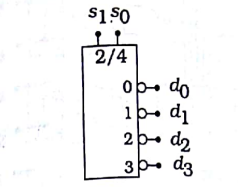
```
module dec_4 (d0, d1, d2, d3, sel ) ;
input [1 : 0] sel ;
output d0, d1, d2, d3 ;
case (sel)
   0 : d0 = 1, d1 = 0 , d2 = 0, d3 = 0 ;
   1 : d0 = 0, d1 = 1 , d2 = 0, d3 = 0 ;
   2 : d0 = 0, d1 = 0 , d2 = 1, d3 = 0 ;
   3 : d0 = 0, d1 = 0 , d2 = 0, d3 = 1 ;
endmodule
```

which explicitly lists each possibility depending on the decimal value of sel. Another approach would be to use **assign** procedures. This represents an abstract high-level description of the operation that contains no structural information. While one can understand the operation of the unit, it must be translated into a lower level description before it can be built. This provides another example of equivalent hierarchical views.

An active-low decoder is shown in Figure 11.12. In this case, the selected output is driven low while the others remain at logic 1 values. The design is achieved by simply replacing the NOR2 gates with NAND2 gates and complementing the inputs to each gate. The HDL code can be written by modifying the active-high listings by changing the logic. The gate-level structural Verilog description of the network can be constructed in the form

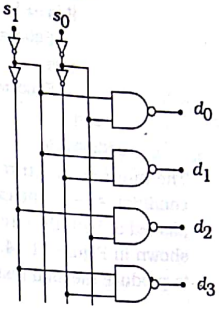
```
module dec_lo (d0, d1, d2, d3, s0, s1) ;
input s0, s1 ;
output d0, d1, d2, d3 ;
nand (d0, ~s0, ~s1) ,
     (d1, ~s0, s1) ,
     (d2, s0, ~s1) ,
     (d3, s0, s1) ;
endmodule
```

by inspection. These simple examples clearly show how large components can be described at various levels. The one that is most used in practice depends upon the problem and its level in the design hierarchy. In general, no single solution will be optimal for all situations.



s_1 s_0	d_0	d_1	d_2	d_3
0 0	0	1	1	1
0 1	1	0	1	1
1 0	1	1	0	1
1 1	1	1	1	0

(a) Symbol and table



(b) NAND2 implementation

Figure 11.12 Active low 2/4 decoder

11.3 Equality Detectors and Comparators

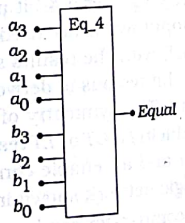
An equality detector compares two n-bit words and produces an output that is 1 if the inputs are equal on a bit-by-bit basis. A simple 4-bit circuit is shown in Figure 11.13. This uses the equality (XNOR) relation

$$\overline{a_i \oplus b_i} = 1 \qquad (11.10)$$

iff $a_i = b_i$ as a means to compare the inputs. If every XNOR produces a 1, then the output AND gate gives *Equal* = 1; otherwise, *Equal* = 0.

A Verilog listing for the operation is

```
module equality (Equal , a, b ) ;
input [3 : 0] a, b ;
output Equal ;
```



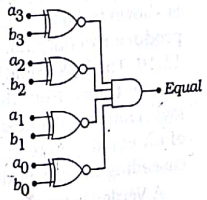


Figure 11.13 A 4-bit equality detector

```
always @ (a or b)
  begin
    if (a == b)
      Equal = 1 ;
    else
      Equal = 0 ;
  end
endmodule
```

The internal structure of the circuitry is hidden in the logical equality condition a == b. The extension to an arbitrary word size is easily accomplished at both the circuit and HDL level. An example is the 8-bit version shown in Figure 11.14. This uses two 4-bit circuits with ANDed outputs to produce the final result.

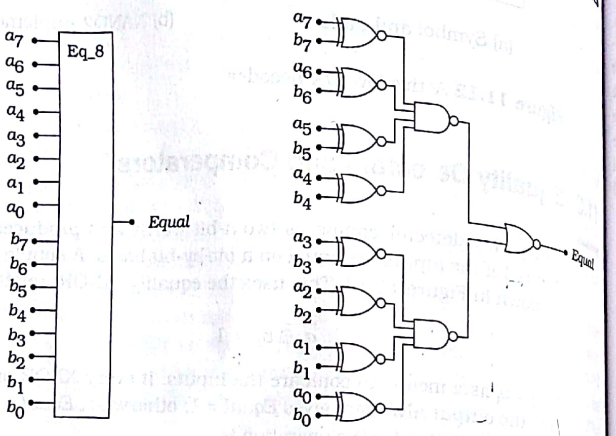


Figure 11.14 8-bit equality detector

Magnitude comparator circuits are used to compare two words a and b and determine if $a > b$ or $a < b$ is true; the equality condition $a = b$ may also be detected by the logic. The logic for a 4-bit magnitude comparator is shown in Figure 11.15. The input words are used on a bit-wise basis to produce two outputs, *GT* and *LT*, with the results summarized in Figure 11.16. The logic equations are a bit tedious to derive, but the signal paths can be traced from the diagram. The symmetry of the upper and lower logic chains is the basis for producing a *GT* or *LT* result. Optional features of an equality detection output and an enable control can be added by cascading the circuit into the logic network shown in Figure 11.17.

A Verilog listing for the 4-bit comparator can be constructed as follows.

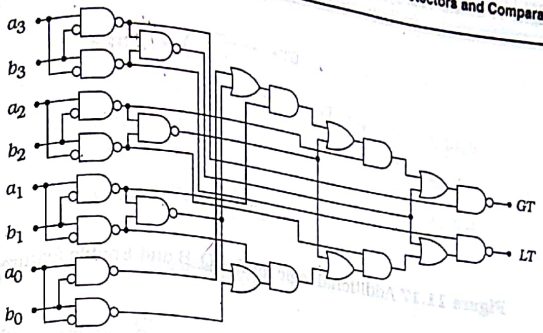


Figure 11.15 4-bit magnitude comparator logic

```
module comp_4 (GT, LT, a, b ) ;
input [ 3 : 0 ] a; b ;
output GT, LT ;
always @ (a or b)
  begin
    if (a > b)
      GT = 1 , LT = 0 ;
    elseif (a < b)
      GT = 0 , LT = 1 ;
    else
      GT = 0 , LT = 0 ;
  end
endmodule
```

The high-level description masks the internal structure completely, making it appropriate for architectural simulations. However, the logic and circuit implementations can be quite complicated .

The hierarchical design technique allows us to build an 8-bit comparator using two 4-bit circuits (Comp 4) and an interfacing network. The main circuit is shown in Figure 11.18. The lower Comp 4 block accepts the lower 4 bits of each word, while the upper block uses bits 4–7. The

Condition	GT	LT
$a > b$	1	0
$a < b$	0	1
$a = b$	0	0

Figure 11.16 Comparator output summary

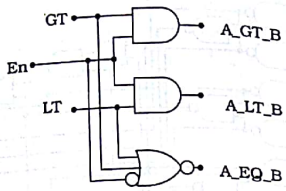


Figure 11.17 Additional logic for A_EQ_B and Enable features

interface block labeled Comp 8 includes an Enable input. The logic d
gram for the interfacing network is shown in Figure 11.19. The upp
inputs are the *GT* Comp 4 outputs, while the lower inputs are *LT* value
from the 4-bit comparison circuits. These are then compared to produ
the outputs. Including the AND gate and the NAND-NOT cascades (at th
A_GT_B and A_LT_B outputs) allows us to generate the equality sign
A_EQ_B that will be 1 if the words are equal. Note that A_GT_B and
A_LT_B are both zero when A_EQ_B = 1.

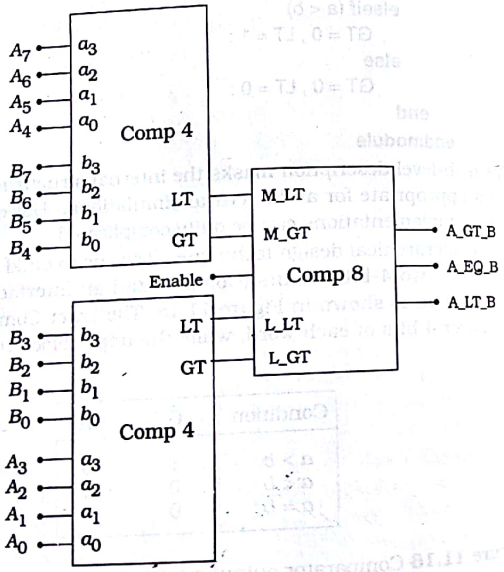


Figure 11.18 8-bit comparator system

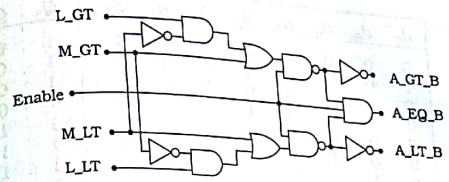


Figure 11.19 Comp 8 logic diagram

11.4 Priority Encoder

A priority encoder examines the input bits of an n-bit word and produces an output that indicates the position of the highest priority logic 1 bit. Consider an 8-bit word

$$d = d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 \tag{11.11}$$

and let us assign the highest priority to bit d_7, the second highest priority to d_6, and so on. The operation of a priority encoder is to detect the presence of 1's in d; if two or more bits are at a logic 1 value, then the input with the highest priority takes precedence. If we use d as the input to an 8-bit priority encoder, then the output word

$$Q = Q_3 Q_2 Q_1 Q_0 \tag{11.12}$$

is coded to indicate the highest priority bit. A function table for this scheme is provided in Figure 11.20. The bit Q_3 is equal to 1 if any input bit is a 1. The 3-bit word $Q_2 Q_1 Q_0$ is encoded to indicate the highest priority input bit. There is no formal logic symbol, so we will use the simple box shown in Figure 11.21 when the device is used in a system design.

The logic for the network is drawn in two parts. The first section in Figure 11.22 shows the input buffers and complement generators for each bit. The output logic for the $Q2$ and $Q3$ is simple and is given by the expressions

$$Q2 = (d_0 + d_1 + d_2 + d_3) \cdot \overline{(d_4 + d_5 + d_6 + d_7)} \tag{11.13}$$

$$Q3 = (d_0 + d_1 + d_2 + d_3) + \overline{(d_4 + d_5 + d_6 + d_7)}$$

as can be verified from the schematic. The $Q0$ and $Q1$ encoders use the buffered and complemented inputs as shown in the circuits of Figure 11.23. The logic equation for the $Q0$ circuit is

$$Q0 = \overline{d_7} \cdot [d_6 + \overline{d_5} \cdot (d_4 + \overline{d_3} \cdot [d_d + \overline{d_1} \cdot d_0])] \tag{11.14}$$

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	-	1	0	0	1
0	0	0	0	0	1	-	-	1	0	1	0
0	0	0	0	1	-	-	-	1	0	1	1
0	0	0	1	-	-	-	-	1	1	0	0
0	0	1	-	-	-	-	-	1	1	0	1
0	1	-	-	-	-	-	-	1	1	1	0
1	-	-	-	-	-	-	-	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0

d_7 has highest priority
d_0 has lowest priority

$Q_3 = 1$ when $d_i = 1$ for any $i = 0, \ldots, 7$

Figure 11.20 Function table for an 8-bit priority encoder

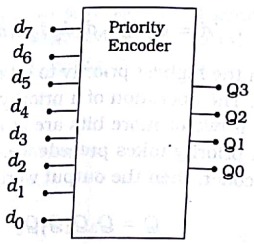


Figure 11.21 Symbol for priority encoder

while

$$Q1 = \overline{d_7} \cdot \overline{d_6} \cdot [d_5 + \overline{d_4} + \overline{d_3} \cdot \overline{d_2} \cdot (d_1 + d_0)] \quad (11.15)$$

gives the $Q1$ bit.

Even though the internal details of the circuit are complicated, the behavioral description is concerned only with the overall functional behavior. One implementation for the module is

```
module priority_8 (Q, Q3, d ) ;
input [ 7: 0 ] d ;
output Q3 ;
output [ 2: 0 ] Q ;
always @ (d)
  begin
    Q3 = 1 ;
    if ( A[7] ) Q = 7 ;
```

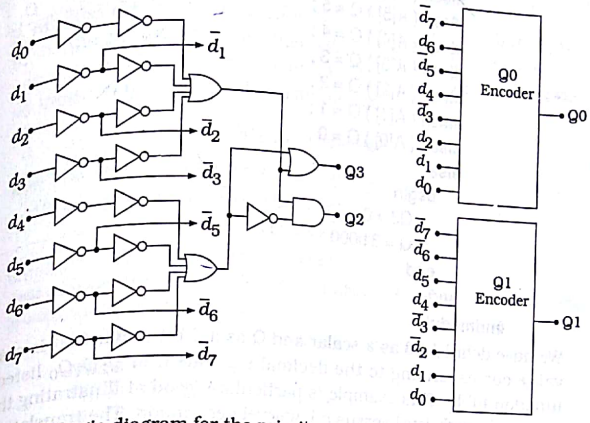
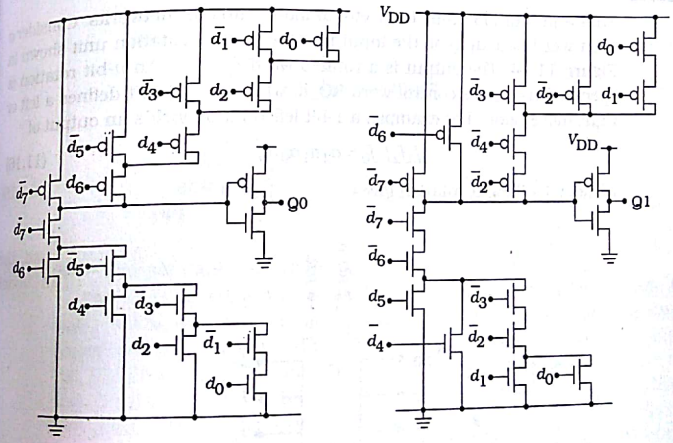



Figure 11.22 Logic diagram for the priority encoder



(a) Q0 circuit (b) Q1 circuit

Figure 11.23 $Q0$ and $Q1$ circuits for the 8-bit priority encoder

```
        elseif ( A[6] ) Q = 6 ;
        elseif ( A[5] ) Q = 5 ;
        elseif ( A[4] ) Q = 4 ;
        elseif ( A[3] ) Q = 3 ;
        elseif ( A[2] ) Q = 2 ;
        elseif ( A[1] ) Q = 1 ;
        elseif ( A[0] ) Q = 0 ;
        else
          begin
            Q3 = 0 ;
            Q = 3'b000 ;
          end
      end
    endmodule
```

We have defined Q3 as a scalar and Q as a 3-bit vector that is assigned a value corresponding to the decimal equivalent of $Q_2Q_1Q_0$ listed in the function table. This example is particularly good at illustrating the separation of a high-level versus a low-level description. The translation of the HDL to the circuit diagram is not a simple problem. Moreover, other equivalent circuits and logic algorithms can be constructed, each with different area and switching properties.

11.5 Shift and Rotation Operations

Shift and rotation units are useful in many different networks. Consider a 4-bit word $a_3a_2a_1a_0$ as the input into the general rotation unit shown in Figure 11.24. The output is a rotated word $f_3f_2f_1f_0$. An n-bit rotation is specified using the control word RO_n, while the L/R bit defines a left or right movement. For example, a 1-bit left rotation yields an output of

$$f_3f_2f_1f_0 = a_2a_1a_0a_3 \tag{11.16}$$

while a 1-bit right rotation gives

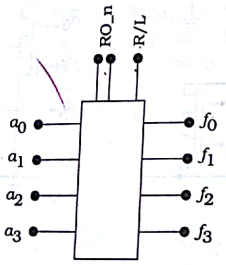


Figure 11.24 General rotator

$$f_3f_2f_1f_0 = a_0a_3a_2a_1 \tag{11.17}$$

A rotation exhibits wrap-around behavior where a bit that is pushed out of the word is added to the other side. A shift operation forces a 0 into the empty space. If we modify the unit to give a 1-bit shift left operation, then an input of $a_3a_2a_1a_0$ produces an output of

$$f_3f_2f_1f_0 = a_2a_1a_00 \tag{11.18}$$

with a similar behavior for a shift right operation.

Verilog provides bit-wise shift operators of

```
<<  // This is a shift left operation
>>  // This is a shift right operation
```

that can be used to specify vector shifts; both fill slots with 0s. These are shown in the example code

```
reg [7:0] a ;
reg [7:0] new_1 ;
reg [3:0] new_2 ;
reg [3:0] b ;
   new_1 = a >> b ; // This shifts the 7-bit word a by b-bits to the right
   new_2 = a << b ; // This shifts a by b-bits to the left
...
```

A rotation can be specified in a number of different ways. The simplest is a bit-by-bit assignment as in the clocked-behavior unit that is described by the listing

```
...
reg [3:0] ;
always @ (posedge clk)
  begin // This is a bit-by-bit rotate left
      a[0] <= a[3] ;
      a[1] <= a[0] ;
      a[2] <= a[1] ;
      a[3] <= a[2] ;
  end
...
```

More general rotations can be included by adding control bits.

There are different ways to implement rotations and shifts in VLSI circuits. While standard FF-based designs can be used, simpler networks based on FET switching properties yield highly regular designs. Consider the rotation of a 4-bit word. The switching network in Figure 11.25 uses four control bits Ror_0, Ror_1, Ror_2, Ror_3 to specify an n-bit rotation. The signals are created using combinational logic; only one of these is a 1 at any given time. Signal routing of the input bits $a_3a_2a_1a_0$ is accomplished by using nFETs as switches that give the desired connection to the output lines $f_3f_2f_1f_0$. A left-rotate array may be created by rewiring

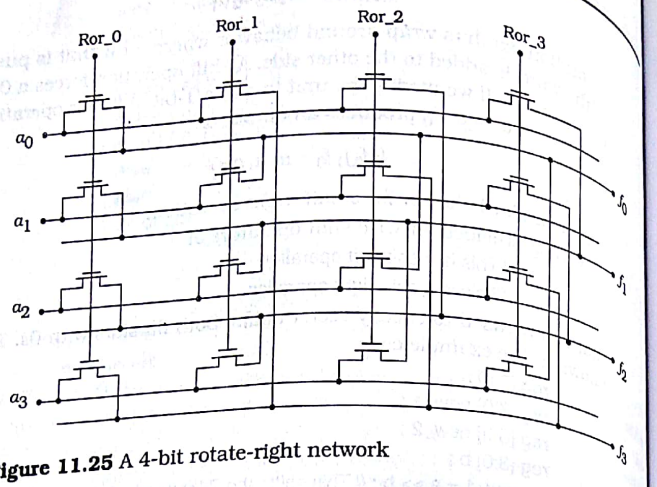


Figure 11.25 A 4-bit rotate-right network

the FETs to the configuration shown in Figure 11.26. The routing of both networks can be verified by simply tracing the input-output paths for each column of control FETs. The two may be combined into a single array that uses a control word Ro_n to specify the number of bits, and a separate control bit for left/right shifting.

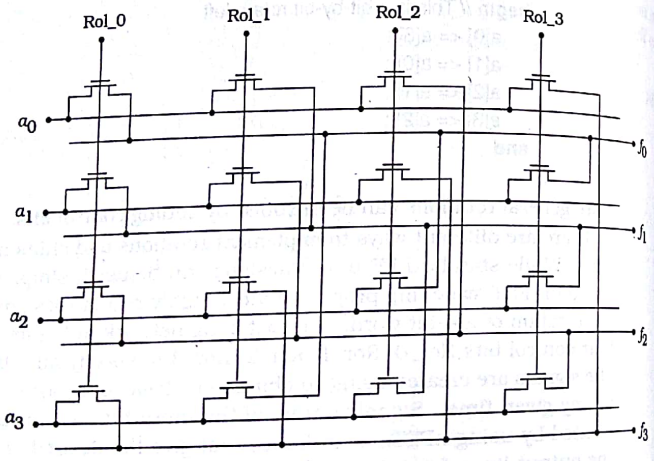
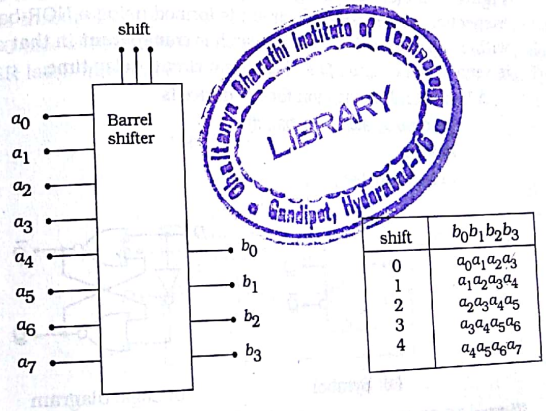


Figure 11.26 Left-rotate switching array

A related component is the **barrel shifter** that is specified as having an $m \times n$ architecture, with m the number of bits in the input word and n the number of output bits. Common situations are where we have $m = 2n$ and $m = n$. Figure 11.27 shows an 8×4 unit. The decimal value of the control word *shift* defines the output $f_3 f_2 f_1 f_0$ in terms of the input word $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$ as summarized in the table. This unit can be built using an nFET array with the result shown in Figure 11.28. As with the rotator designs, each column of transistors is controlled by a single-bit signal Sh_n where n = 0, 1, 2, 3, 4 in this design. These are produced by a combinational logic network from the 3-bit control word *shift*. Only one of the column signals is 1 at a time, so that the input-to-output paths are defined by the column of transistors that is active. A 4×4 network with wrap-around is just a rotator. Barrel shifters are useful in ALUs (arithmetic and logic units) for bit manipulations. The overall structure of the circuit itself provides a basis for designing integrated cross-bar switching networks and signal routers for applications in telecommunications and parallel processing.

The nFET array is extremely regular, which makes it relatively easy to layout at the CMOS physical design level. Library cells can be as simple as individual FETs, or as complicated as $p \times q$ sub-units that are used to create larger arrays. The main drawbacks of the nFET-only design are the threshold voltage drop problem (and the associated weak-1 transmission) and parasitic-limited switching times. Drivers can be added to speed up the circuits and restore the voltage swings. Alternately, transmission gates can be used as a 1-for-1 replacement of the FETs. Although the pFET area consumption is small, the routing complexity increases considerably.



shift	$b_0 b_1 b_2 b_3$
0	$a_0 a_1 a_2 a_3$
1	$a_1 a_2 a_3 a_4$
2	$a_2 a_3 a_4 a_5$
3	$a_3 a_4 a_5 a_6$
4	$a_4 a_5 a_6 a_7$

Figure 11.27 An 8×4 barrel shifter

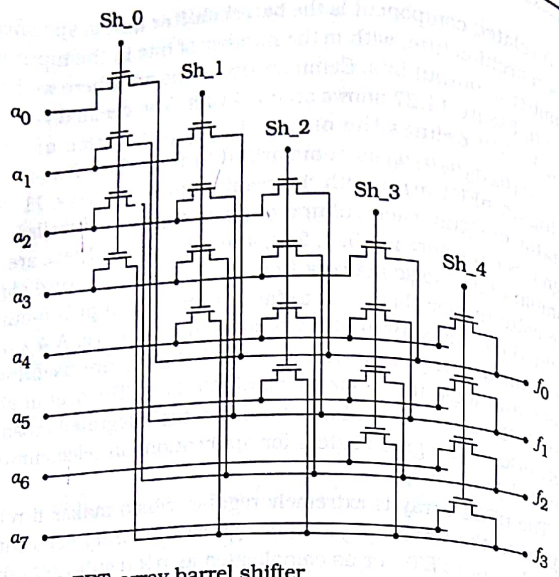


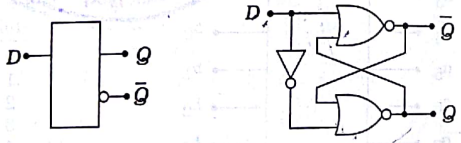
Figure 11.28 FET-array barrel shifter

11.6 Latches

A latch is a device that can receive and hold an input bit. A simple D-latch forms the basis for many designs. The symbol for the D-latch is shown in Figure 11.29(a), and a logic diagram is provided in Figure 11.29(b). By inspection we see that the circuit is formed using a NOR-based SR latch with complemented inputs. The latch is **transparent** in that a change in D is seen at the outputs Q and $\overline{Q}$ after a circuit delay time.

A behavioral description for the device is

```
module d_latch (q, q_bar, d) ;
input d ;
output q, q_bar ;
```



(a) Symbol (b) Logic diagram

Figure 11.29 D-latch


```
reg q, q_bar ;
always @ (d)
   begin
      # (t_d) q = d ;
      # (t_d) q_bar = ~d ;
   end
endmodule
```

We have used t_d for the time delay. The declaration models the action of the cross-coupled NOR circuit that can hold the input state. The equivalent structural description is

```
module d_latch_gates (q, q_bar, d ) ;
input d ;
output q, q_bar ;
wire not_d ;
not (not_d, d) ;
nor # (t_nor) g1 (q_bar, q, d),
    # (t_nor) g2 (q, q_bar, not_d);
endmodule
```

This provides a gate-by-gate guide for the device at the circuit and physical design level

Combinational Logic Designs

The CMOS circuit can be constructed using either the logic diagram or the structural description. A direct translation is shown in Figure 11.30. At the physical level, this can be created by instancing two NOR2 cells and one NOT cell, and then adding the interconnect wiring. Alternately, a custom layout would probably consume less area.

An enable control En can be added to the basic D-latch by routing the inputs through AND gates as illustrated in Figure 11.31. An enable bit of $En = 0$ blocks the inputs by forcing 0's to the AND outputs, which places the SR latch into a hold state. If $En = 1$, then the values of D and $\overline{D}$ are

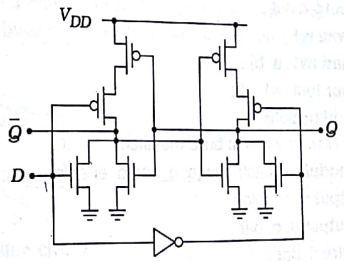
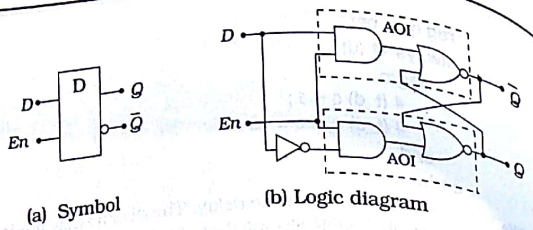


Figure 11.30 CMOS circuit for a D-latch



(a) Symbol (b) Logic diagram

Figure 11.31 Gated D-latch with *Enable* control

admitted to the NOR circuits. To include this control in the behavioral description, we rewrite the code as

```
module d_latch (q, q_bar, d , enable) ;
input d, enable ;
output q, q_bar ;
reg q, q_bar ;
always @ (d and enable)
   begin
      # ( t_d ) q = d ;
      # ( t_d ) q_bar = ~ d ;
   end
endmodule
```

A condition of enable = 1 is required to change the state.

The structural description could be based on a brute-force listing of the circuit. However, since CMOS allows for complex logic gates as primitives, it makes more sense to describe the complex gate and then instance it in the listing.

```
// First define the AOI module
module aoi_2_1 (out, a, b, c) ;
input a, b, c ;
output out ;
wire w1 ;
and (w1, a, b) ;
nor (out, w1, c) ;
endmodule
// Now use this to build the latch
module d_latch _aoi (q, q_bar, d, enable) ;
input d, enable ;
output q, q_bar ;
wire d_bar ;
not (d_bar, d) ;
```



```
aoi_2_1 (q_bar, d, enable, q) ;
aoi_2_1 (q, enable, d_bar, q_bar) ;
endmodule
```

The module name aoi_2_1 is interpreted as an AOI gate with 2 inputs to the AND, and 1 input to the OR. Although the notation is not standard, it is widely used in practice. Note that the order of the inputs must be preserved when the module is instanced. The CMOS circuit diagram for this implementation is shown in Figure 11.32, and consists of one NOT gate and two AOI circuits.

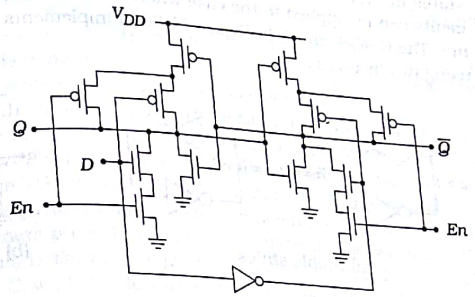


Figure 11.32 AOI CMOS gate for D-latch with Enable

CMOS VLSI Latch

Many static D-latches CMOS VLSI are constructed from inverters and TGs or pass FETs. This design is based on the characteristics of the simplest static storage configuration called a **bistable circuit**. A bistable circuit is one that can store (or, hold) either a logic 0 or a logic 1 indefinitely (or, at least as long as power is applied.)

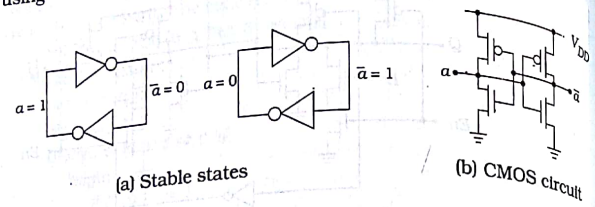
A basic bistable circuit consists of two inverters as shown in Figure 11.33(a). The gates are wired such that the output of one is the input to the other, forming a closed loop. Any closed loop with an even number of inverters gives a bistable circuit. If we use three inverters as in Figure

(a) Bistable circuit (b) Ring oscillator

Figure 11.33 Closed-loop inverter configurations

11.33(b), the resulting circuit is unstable and cannot hold a bit value. A closed loop with an odd number of inverters is often called a **ring oscillator** as the signal at any point oscillates in time.

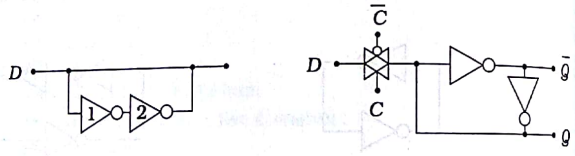
The storage mechanism of the bistable circuit is illustrated in Figure 11.34(a). If the left side is at a value $a = 1$, then tracing the signal path through the upper inverter shows that the right side has a value of $\bar{a} = 0$. If we continue and trace the signal to the left through the lower inverter, we obtain the starting point of $a = 1$. This shows that the state $a = 1$ is stable in that it can be maintained by the circuit itself. The same arguments can be applied to the case where $a = 0$, as shown in the same figure. The CMOS circuit in Figure 11.34(b) implements the bistable circuit using two inverters.



(a) Stable states (b) CMOS circuit

Figure 11.34 Operation of a bistable circuit

To create a D-latch, we must provide an entry node for the input bit. One simple idea is the **receiver circuit** shown in Figure 11.35(a). The value of D is held by the bistable circuit formed by the inverter pair. The circuit helps the line resist changes in D, making it useful as an input stage to a receiver module that is driven by an external line. The presence of the **feedback loop** from the output of Inverter 2 to the input of Inverter 1 provides the desired latching, but complicates the design. Inverter 1 needs to detect changes, but Inverter 2 cannot be so strong that it prohibits a change in state. In general, Inverter 1 can use relatively large FETs,



(a) Receiver circuit (b) Controlled loading

Figure 11.35 Adding an input node to the bistable circuit

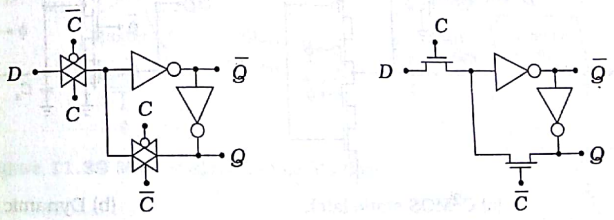
but Inverter 2 is purposely made weaker by using small transistors.

Adding a transmission gate at the input as shown in Figure 11.35(b) gives us the ability to control the loading. When $C = 0$ (so $\bar{C} = 1$), the TG acts as an open switch and the circuit holds the values of Q and $\bar{Q}$ at the output. When the control bit is set to $C = 1$ ($\bar{C} = 0$), the TG conducts and allows the input bit D to be transferred to the latching circuit. During this time the latch is transparent and the outputs go to $Q = D$ and $\bar{Q} = \bar{D}$. If C is reset to 0, then the state is held. The control bit C is thus equivalent to the enable signal *En* used previously. The inverter design constraints still apply.

While these circuits are easy to build in CMOS, they are relatively slow devices. This is because the bistable circuit tries to hold onto the stored values and resists changes as discussed for Figure 11.34(a) . If we force a change of the stored voltage, the feedback that exists from the output back to the input fights the transition. A solution to this problem is to add another switch that breaks the feedback loop during which a value is stored. The TG circuit shown in Figure 11.36(a) accomplishes this by using an oppositely phased switch between the inverters. In many cases, chip designers prefer to use nFETs in place of TGs because of the simpler wiring. This is shown in Figure 11.36(b); note that the input FET is controlled by C, while the feedback transistor is switched by the complement $\bar{C}$.

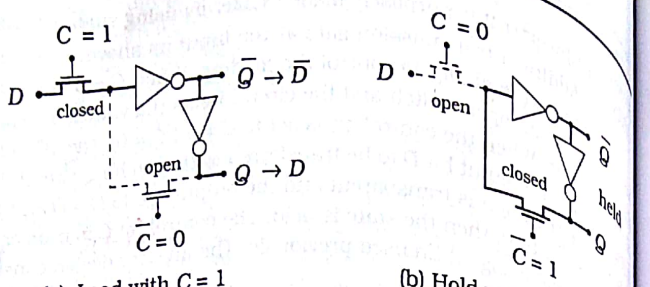
The operation of the nFET latch is summarized in Figure 11.37. A load of the input data bit D occurs when $C = 1$. As shown in Figure 11.37(a), this turns on the input FET and opens the feedback loop. The input thus sees a simple NOT chain, and loads very quickly. A control bit of $C = 0$ defines the hold state, and is illustrated in Figure 11.37(b). This turns off the input FET but the feedback loop is established to hold the values of Q and $\bar{Q}$. Since the logic equation for a TG is identical to that for an nFET, this description also applies to the TG-based circuit.

C^2MOS circuits provide the basis for another D-latch design style. The latch in Figure 11.38(a) uses a C^2MOS inverter as the input stage with the



(a) CMOS TG version (b) nFET pass gates

Figure 11.36 D-latch using oppositely phased switches

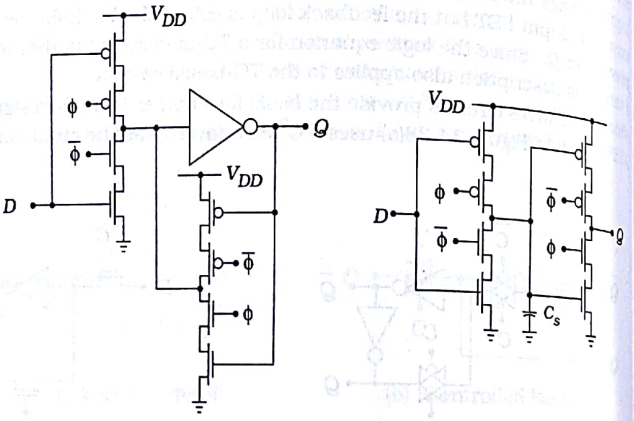


(a) Load with $C = 1$ (b) Hold with $C = 0$

Figure 11.37 Operation of the D-latch

clock ϕ controlling the loading. When $\phi = 0$, D is admitted into the circuit where it passes through the static inverter. This gives $Q = D$ after the characteristic delay time. When the clock makes a transition to $\phi = 1$, the first stage is driven into a Hi-Z state, while the feedback loop is closed by the oppositely phased inverter. This holds the output until the next clock cycle.

A variation that uses a pair of cascaded C^{2}MOS inverters is shown in Figure 11.38(b). This operation of this network is quite different in that it is a true dynamic circuit, i.e., it uses charge storage on capacitive nodes. The parasitic capacitor C_s between the two stages acts as the storage device for this circuit. With $\phi = 0$, D is admitted and the charge corresponding to $\bar{D}$ is stored on C_s. When the clock changes to $\phi = 1$, the out-



(a) C^{2}MOS static latch (b) Dynamic latch

Figure 11.38 C^{2}MOS-based D-latch circuits

put of the first stage is in a Hi-Z state which holds the charge on C_s. This clock phase activates the second stage which has an output of Q. The value of Q will be that of $\bar{D}$ delayed by the rise or fall time of the circuit. It is important to note that charge leakage limits the time that C_s can hold the state. Although introduced here as a latch, this circuit is often used as a time-delay element for synchronous networks.

11.7 D Flip-Flop

83448

A flip-flop differs from a latch in that it is non-transparent. The D-type flip-flop (DFF) is the most commonly used flip-flop in CMOS circuits. The basic DFF design is a master-slave configuration obtained by cascading two oppositely phased D-latches as in Figure 11.39. The clock signal ϕ controls the operation and provides synchronization. The master latch allows an input of D when $\phi = 0$ and M1 acts as a closed switch. During this time, nFETs M2 and M3 are open circuits. When the clock makes a transition to $\phi \rightarrow 1$, switches M2 and M3 close and effect the transfer of the bit to the slave. The input to the master is blocked since M1 is open with $\phi = 1$. The master-slave circuit acts as a **positive edge-triggered** device since the value of D during the positive clock edge defines the value of the bit that is transferred to the slave and available as Q.[2] Note that once the bit is latched into the slave at time t_1, it does not appear at the output until

$$t_1 + t_{FET} + t_{NOT} \tag{11.19}$$

where t_{label} is the rise or fall time of the specified element. The symbol for a positive edge-triggered DFF is shown in Figure 11.40(a). The 'triangle' denotes an edge-triggering input. Adding a bubble produces the symbol

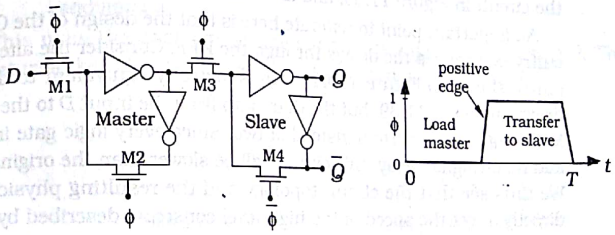
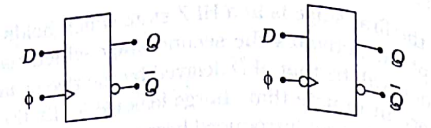


Figure 11.39 Master-slave D-type flip-flop

[2] In the strictest sense, there is a difference between a master-slave FF and a true positive edge-triggered circuit. However, in VLSI design, the terminology is used interchangeably.



(a) Positive edge-triggered DFF (b) Negative edge-triggered DFF

Figure 11.40 Edge-triggered DFF symbols

for a **negative edge-triggered** DFF in Figure 11.40(b). At the circuit level all that needs to be done is to interchange the ϕ and $\bar{\phi}$ signals.

A Verilog behavioral description of a positive edge-triggered DFF can be written in the following manner.

```
module positive_dff (q, q_bar, d, clk) ;
input d, clk ;
output q, q_bar ;
reg q, q_bar ;
always @ (posedge clk)
  begin
    q = d ;
    q_bar = ~ d ;
  end
endmodule
```

In a realistic application, a set of delay times would be needed. A negative edge-triggered module is obtained by modifying the **always** statement to

```
always @ (negedge clk)
```

This changes the stimulus in an obvious manner. A structural description is straightforward using the **nmos** and **not** primitive elements to model the circuit in Figure 11.39, and is left as an exercise.

An important point to reiterate here is that the design of the CMOS circuitry determines the delays through the DFF. Consider the alternate circuitry shown in Figure 11.41. This is logically equivalent to the circuit drawn in Figure 11.39, but the data path from the input *D* to the output *Q* is through four inverters instead of two. Since every logic gate introduces additional signal delay, this circuit will be slower than the original design. We thus see that the circuit topology and the resulting physical design directly affect the speed of the high-level construct described by the HDL listing. This type of consideration is one of the factors that distinguishes high-speed VLSI from other digital systems designs.

It is possible to add direct **clear** and **set** capabilities to the circuit by changing the gate functions. One approach is to use NAND2 logic. Consider the case in Figure 11.42 where one input is a control bit *s* and the

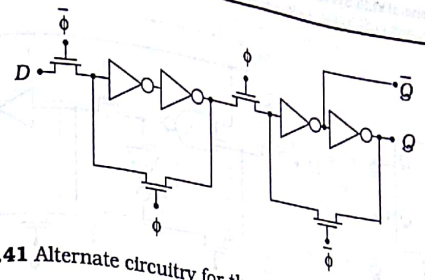
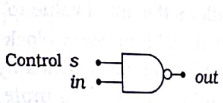


Figure 11.41 Alternate circuitry for the master-slave DFF

other is a data value *in*. When $s = 0$, the output is 1 regardless of the value of *in*. If $s = 1$, then $out = \overline{in}$ as shown. Replacing selected inverters with NAND2 gates yields DFFs with assert-low clear or set inputs, or both. Figure 11.43(a) provides the ability to clear (to 0) the contents of the latch using the *Clear* control. With *Clear* = 1, the NAND gates act as inverters and the circuit behaves as a normal DFF. When *Clear* = 0, the NAND in the slave forces an output of $Q = 0$. The output of the master is forced to a logic 1, which inverts to an output $Q = 0$. A Verilog listing that describes this device is

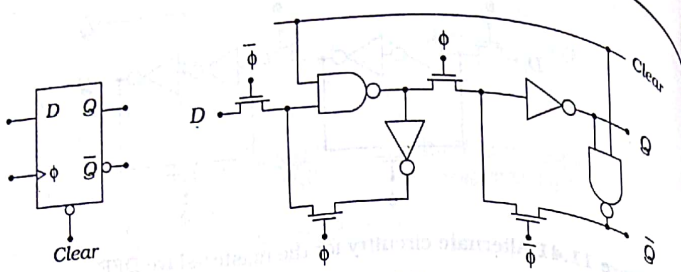
```
module dff_clear (q, d, clear, clk) ;
input d, clk, clear ;
output q ;
reg q ;
always @ (posedge clk )
      q = d ;
always @ ( clear )
   if ( clear )
      assign q = 0 ;
   else
      deassign q ;
endmodule
```

This uses the **deassign** statement that is executed if clear is 0. This returns the value of q to its value established in the q = d line.

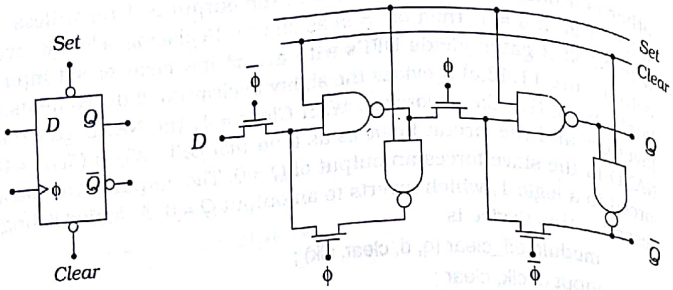


s	*in*	*out*
0	0	1
0	1	1
1	0	$\overline{in}$
1	1	$\overline{in}$

Figure 11.42 NAND2 used as a control element



(a) DFF with direct Clear



(b) DFF with direct Clear and Set

Figure 11.43 DFF circuits with assert-low Clear and Clear/Set controls

A DFF with both clear and set controls is shown in Figure 11.43(b). The set capabilities are achieved by substituting NAND2 gates for the other two inverters. A condition of *Set* = 1 gives normal operation, while *Set* = 0 forces the slave to an output of $Q = 1$. This is reinforced by the output of the master. Note that *Clear* and *Set* cannot both be 0 at the same time as this forces $Q = \overline{Q}$. It is common to find several variations of DFFs in a cell library including features such as input buffers, clock buffers, and inputs with combinational logic gates. For example, a toggle flip-flop (TFF) that changes states on every rising clock edge can be created by adding a feedback loop from $\overline{Q}$ to *D* as shown in Figure 11.44. The assert-low *Set* logic modification allows the initial value to be established as a 1.

A basic DFF loads a new data bit on every clock edge. Storage over an arbitrary number of clock cycles can be obtained by adding a control signal and associated logic to the circuit. A simple classical solution is shown in Figure 11.45(a). The control signal *Load* controls a 2:1 MUX. If the control signal is asserted with *Load* = 1, the MUX allows a new data value *D* to enter the DFF. A control bit with a value *Load* = 0 takes the

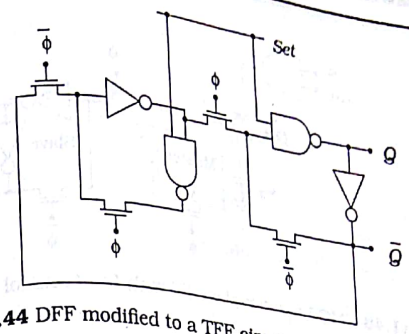


Figure 11.44 DFF modified to a TFF circuit using feedback

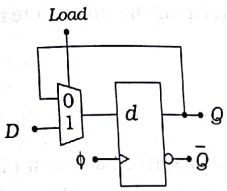
output Q and redirects it back to the input. This type of circuit is usually integrated as a single element with the simplified symbol shown in Figure 11.45(b). A simple Verilog description that includes the *Load* control is

```
module dff_load (q, q_bar, d, load, clk) ;
input d, clk, load ;
output q, q_bar ;
reg q, q_bar ;
always @ (posedge clk )
  begin
    if ( load ) q = d ;
    q_bar = ~ d ;
  end
endmodule
```

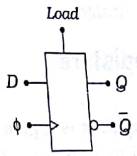
Another approach to designing a controlled-loading DFF is shown in Figure 11.46. This is the basic CMOS master-slave arrangement with a modified control signal

$$\overline{\phi} \cdot Load \tag{11.20}$$

applied to the input FET M1. This allows loading only when both *Load*



(a) Wiring diagram



(b) Symbol

Figure 11.45 D-type flip-flop with *Load* control

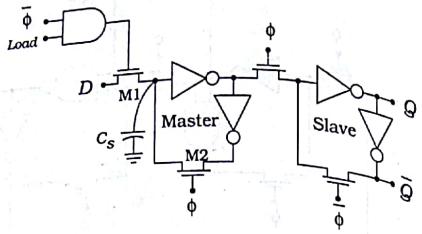


Figure 11.46 CMOS master-slave FF with *Load* control

and $\bar{\phi}$ are 1. The operation of the circuit is shown in Figure 11.47. A load condition is shown in Figure 11.47(a), and is identical to the operation of the original circuit. The value of D establishes the voltage V_{in} at the input to the master circuit of either 0 V or V_{max} corresponding to a logic 0 or a logic 1, respectively. Note that this voltage establishes the charge state on the storage capacitor C_s shown in the schematic.

There are two possible conditions for a data hold state when *Load* = 0. When $\phi = 1$, the master latch holds the value of the data bit using a clocked feedback loop, and transfers it to the slave and the outputs. This is shown in Figure 11.47(b). If, on the other hand, $\phi = 0$ and *Load* = 0 at the same time, then the circuit switches are in the states shown in Figure 11.47(c). The master feedback loop is open. Storage is achieved by holding the charge on C_s, making this a quasi-dynamic circuit that is subject to charge leakage effects. Note that the slave feedback loop is closed during this time. This establishes the voltage V_a. When the clock returns to $\phi = 1$, the circuit of Figure 11.47(b) is again valid. This has two effects. First, the master feedback loop closes and establishes static hold capabilities. Second, the slave voltage V_a reinforces the value of the voltage on C_s. It is important to remember that the charge leakage may place a lower limit on the usable clock frequency. In particular, idling the clock may cause the circuit to exhibit long delays when the clock is restarted. This may have significant ramifications in developing methods to test a chip that uses the design.

11.8 Registers

A **register** is a general term that describes a group of circuits that are used to store a word as a unit entity. A 1-bit register is simply a single flip-flop, whereas an n-bit register loads and holds an n-bit word. Registers are important components in VLSI design. They allow us to design sequential circuits and state machines that form the basis of modern digital systems. The transition from single-bit logic to word-handling units is a critical step in the design hierarchy.

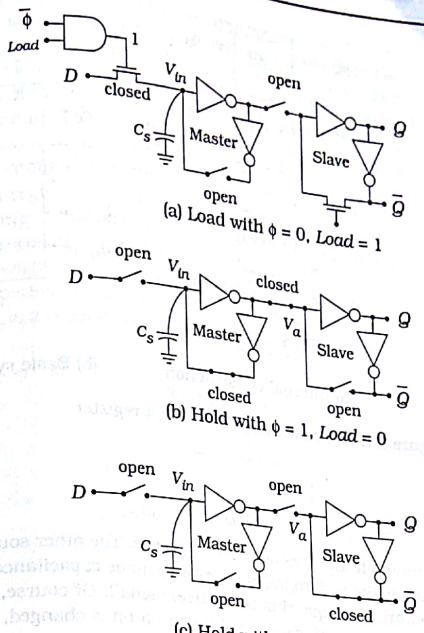
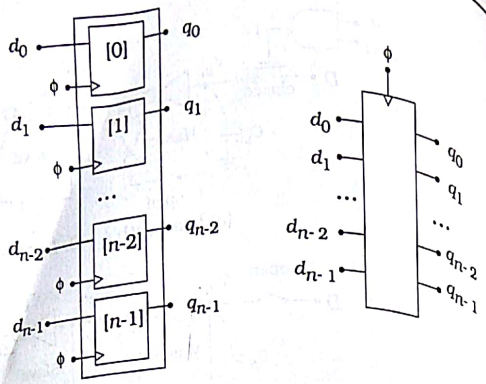


Figure 11.47 Operation of the CMOS DFF with load control

An n-bit positive edge-triggered register can be built by paralleling n single-bit DFFs as shown in Figure 11.48(a). The register symbol shown in Figure 11.48(b) masks the details of the individual circuits, but provides the basic operation: n-bit input words are loaded into the register on a rising clock edge. The outputs are valid after a circuit-induced delay time. Hold capabilities can be added as discussed in the previous section.

Since DFF's are intrinsically clock-controlled, they affect the dynamic power dissipation of the network. The DFF operation described previously in Figure 11.47 shows two sources of dynamic power. One is due to the four clocking FETs needed to control the operation of the master-slave arrangement. These load the clock drivers and increase the dynamic power



(a) Internal construction (b) Basic symbol

Figure 11.48 Construction of an n-bit register

$$P_{dyn} = C_L V_{DD}^2 f \qquad (11.2)$$

by adding to C_L. This occurs every cycle. The other source of power dissipation is the recharging of the storage node capacitance C_s to compensate for charge leakage; this is relatively small. Of course, the inverters also dissipate power if the value of the input bit is changed, but this cannot be avoided; logic operations always require energy.

A purely static multi-output port solution is shown in Figure 11.49. This circuit is the basic two-inverter storage circuit with access transistors and an output driver. The write-enable signal WE controls the loading, while the two read-enable signals (RE_a and RE_b) are used to demultiplex the output to Qa or Qb. Since this uses single-ended inputs, the design of the inverters is important. Inverter 1 should be sensitive to the input and have reasonable drive strength for quick response. Relatively large transistors would be appropriate for this and inverter 3.

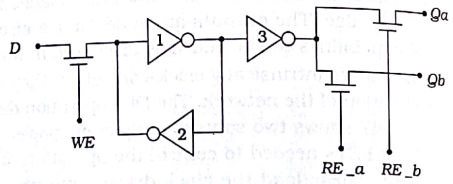


Figure 11.49 One-bit static multiport register circuit

Inverter 2 provides the feedback that latches the data bit. Since it will resist changes at the input, it is usually designed to be much weaker with small transistors. These considerations are identical to those discussed in the context of the latch in Figure 11.35(a). Clocking is not included in this primitive circuit, and must be added to the input if needed. A simple solution is to add a clock-controlled nFET in series with the input transistor, or create a composite control signal $WE \cdot \phi$ for the existing circuit.

An n-bit register using this circuit is shown in Figure 11.50. This provides one input and two outputs for every bit. Additional ports can be added if necessary. Since there is no clock power dissipation associated with this design, it is useful in situations where the contents may be held for a long period of time. In particular, it can be used to build a **register file**, which is a collection of word-size storage registers.

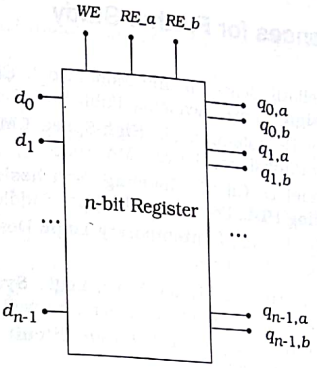


Figure 11.50 An n-bit static multiport register

11.9 The Role of Synthesis

The logic components in this chapter were chosen to illustrate techniques of translating a high-level HDL description down to a logic or circuit schematic. In each case, the large-scale function could be implemented using different logic networks or CMOS design styles. Since every realization of a given function will result in different values for the switching speed and silicon real estate, the role of the designer is complicated by the need to make the correct choice. CAD toolsuites allow different designs to be built and characterized for comparison. This is particularly important for designing critical datapaths or complex logic networks.

(b) Select a CMOS design technique for the DFFs and use it to construct the circuit.

(c) Now write a Verilog description of the shift register using nmos and pmos primitives.

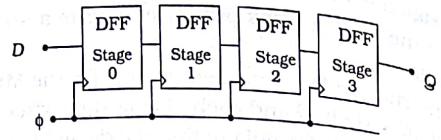


Figure P11.1

[11.10] Construct a Verilog module for an 8-bit register that loads word on rising clock edges iff the control bit *En* is 1. You may use any level description.

[11.11] Write a structural description of the register circuit shown in Figure 11.49.

Arithmetic Circuits in CMOS VLSI

12

Arithmetic functions such as addition and multiplication have a special significance in VLSI designs. Many applications require these basic operations, but good silicon implementations have been a challenge since the early days of digital chip building. In this chapter we will examine binary adders in detail, and extend the discussion to include multipliers.

12.1 Bit Adder Circuits

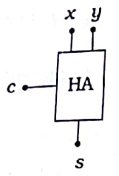
Consider two binary digits x and y. The binary sum is denoted by $x + y$ such that

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+0 &= 1 \\ 1+1 &= 10 \end{aligned} \qquad (12.1)$$

where the result in the last line is a binary 10 (i.e., 2 in base-10). This simple example illustrates the problem with addition. If we take two base-r numbers with digits 0, 1, ... , (r-1), then the sum of two numbers can be out of the range of the digit set itself. This, of course, is the origin of the concept of a carry-out. In the binary sum 1+1, the result 10 is viewed as a 0 with a 1 shifted to the left to give a "carry-out of 1."

A **half-adder** circuit has 2 inputs (x and y) and 2 outputs (the sum s and the carry-out c) and is described by the table provided in Figure 12.1. The outputs are given by the basic equations

$$\begin{aligned} s &= x \oplus y \\ c &= x \cdot y \end{aligned} \qquad (12.2)$$



x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 12.1 Half-adder symbol and operation

which are taken directly from the table. A high-level Verilog behavioral description of the cell can be written as

```
module half_adder (sum, c_out, x, y ) ;
input x, y ;
output sum, c _out ;
assign { c _out , sum } = x + y ;
endmodule
```

This defines x and y as single-bit quantities, and then uses the concatenation operator { } to obtain a 2-bit result. This operator "connects" binary segments in the order listed to create a single result. Alternately, we may construct the gate-level network shown in Figure 12.2. This is described by the structural model

```
module half_adder_gate (sum, c_out, x, y ) ;
input x, y ;
output sum, c _out ;
and (c_out, x, y ) ;
xor (sum, x, y ) ;
endmodule
```

using primitive gate instances. Two more possibilities are shown in Figure 12.3. The half-adder in Figure 12.3(a) uses NAND2 gates, while the alternate in Figure 12.3(b) is a NOR-based design. Preference might be given to the NAND design since it avoids series pFET chains, but a half-adder is simple enough so that the difference is not a major factor.

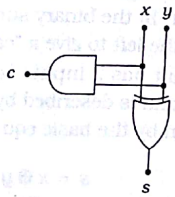
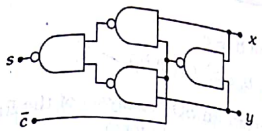


Figure 12.2 Half-adder logic diagram



(a) NAND2 logic (b) NOR-based network

Figure 12.3 Alternate half-adder logic networks

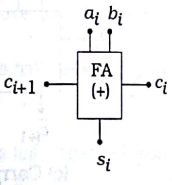
A more complicated problem is adding n-bit binary words. Consider two 4-bit numbers $a = a_3a_2a_1a_0$ and $b = b_3b_2b_1b_0$. Adding gives

$$\begin{array}{r} a_3a_2a_1a_0 \\ +\ b_3b_2b_1b_0 \\ \hline c_4\ s_3s_2s_1s_0 \end{array} \qquad (12.3)$$

where $s = s_3s_2s_1s_0$ is the 4-bit result and c_4 is the carry-out bit. To design an adder for the binary words, we break the problem down to bit level adders on a column-by-column basis. In the standard carry algorithm, each of the i-th columns ($i = 0, 1, 2, 3$) operates according to the **full-adder** equation

$$\begin{array}{r} c_i \\ a_i \\ +\ b_i \\ \hline c_{i+1}\ s_i \end{array} \qquad (12.4)$$

where c_i is the carry-in bit from the (i-1)-st column, and c_{i+1} is the carry-out bit for the column. The operation is described by the full-adder table given in Figure 12.4 along with a simple schematic symbol. The most common expressions for the network are



a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Figure 12.4 Full-adder symbol and function table

$$s_i = a_i \oplus b_i \oplus c_i$$
$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i) \quad (12.5)$$

which can be derived directly from an SOP analysis of the function table. The carry-out bit can be written in alternate form

$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i + b_i) \quad (12.6)$$

if desired.

A particularly compact circuit implementation is obtained using dual-rail complementary pass-transistor logic (CPL). The basic building block is the CPL 2-input array that has the generic form illustrated in Figure 12.5(a). The sum circuit is shown in Figure 12.5(b); the output of the first XOR/XNOR gate is the pair

$$a_i \oplus b_i \quad \text{and} \quad (\overline{a_i \oplus b_i}) \quad (12.7)$$

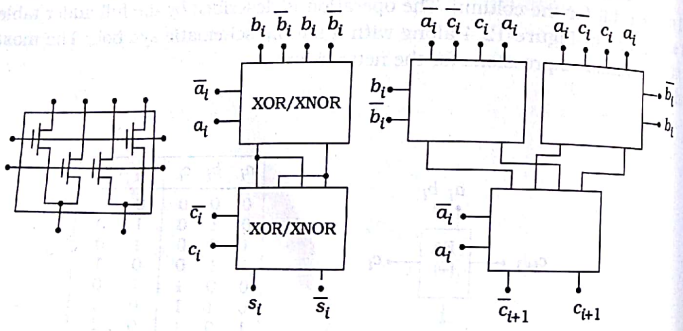
The second gate produces the sum by

$$s_n = (\overline{a_i \oplus b_i}) \cdot c_i + (a_i \oplus b_i) \cdot \bar{c}_i \quad (12.8)$$

The carry circuit in Figure 12.5(c) employs the 2-input array as a combinational logic element. For example, the top array on the left-hand side produces outputs of

$$\bar{a}_i \cdot b_i + \bar{b}_i \cdot \bar{c}_i$$
$$\bar{b}_i \cdot c_i + a_i \cdot b_i \quad (12.9)$$

while the upper right circuit gives



(a) 2-input array (b) Sum circuit (c) Carry circuit

Figure 12.5 CPL full-adder design

$$\bar{a}_i \cdot \bar{b}_i + b_i \cdot \bar{c}_i$$
$$b_i \cdot c_i + a_i \cdot \bar{b}_i \quad (12.10)$$

The last gate uses these to produce c_{i+1} and $\bar{c}_{i+1}$. Although this is a simple-looking solution, it must be remembered that CPL is a dual-rail technique that requires complementary variable pairs such as $(a_i, \bar{a}_i)$ at every stage. Also, because the threshold voltage loss drops the value of a logic 1 voltage as it passes through an nFET, restoring buffers or latching circuits are needed at the output. CPL is thus a somewhat specialized solution to implementing the CMOS full-adder. We note in passing that a CPL half-adder is easy to construct since it requires only the XOR/XNOR and AND/NAND functions.

A behavioral description of the full-adder is obtained by a simple modification of the half-adder model to the form

```
module full_adder (sum, c_out, a , b , c_in) ;
input a, b, c_in ;
output sum, c _out ;
assign { c _out, sum } = a + b + c_in ;
endmodule
```

All variables are scalars (single bits) and the concatenation operator creates the two outputs. Structural modeling can be based on the gate-level network shown in Figure 12.6(a). This is a straightforward one-to-one translation of the equation set. At this level, the module takes the form

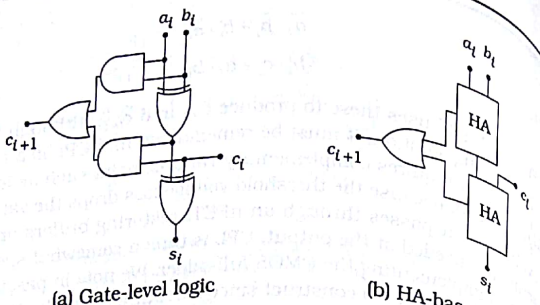
```
module full_adder_gate (sum, c_out, a, b, c_in ) ;
input a, b, c_in ;
output sum, c _out ;
wire w1, w2, w3 ;
xor (w1, a, b) ,
    (sum, w1, c_in) ;
and (w2, a, b) ,
    (w3 , w1, c_in) ;
or (c_out, w2, w3) ;
endmodule
```

where we have used slightly different variable identifiers to make the code more readable. A full-adder can also be built from two HA modules as shown in Figure 12.6(b). Using instances of the module defined by the listing

```
module half_adder_gate (sum, c_out, x, y) ;
...
```

gives

```
module full_adder_HA (sum, c_out , a, b, c_in) ;
input a, b, c_in ;
output sum, c _out ;
wire wa, wb, wc ;
```



(a) Gate-level logic (b) HA-based design

Figure 12.6 Full-adder logic networks

```
half_adder_gate (wa, wb, a, b) ;
half_adder_gate (sum, wc, wa, c_in) ;
or (c_out, wb, wc) ;
endmodule
```

as the description.

Owing to the importance of the full adder, several implementations have been developed over the years. An AOI algorithm for static CMOS logic circuits can be obtained by writing the carry-out bit using equation (12.6). This allows us to write

$$\bar{s}_i = (a_i + b_i + c_i) \cdot \bar{c}_{i+1} + (a_i \cdot b_i \cdot c_i) \tag{12.11}$$

so that both c_{i+1} and $\bar{s}_i$ are in SOP form. Moreover, $\bar{s}_i$ uses $\bar{c}_{i+1}$ so that we can design an AOI gate for $\bar{c}_{i+1}$ and use the output to feed another AOI gate for $\bar{s}_i$. Figure 12.7 shows the construction of the two OAOI networks.

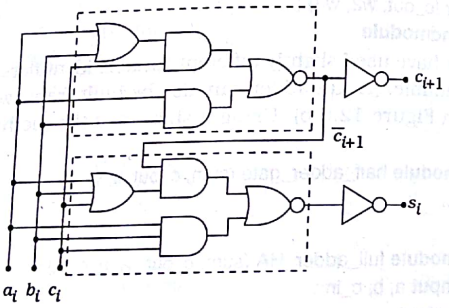


Figure 12.7 AOI full-adder logic

The upper circuit produces $\bar{c}_{i+1}$ and the lower one gives s_i after inversion. It is a straightforward exercise to design both OAOI circuits using series-parallel CMOS gates. Note, however, that equation (12.11) contains four OR operations, which indicates that the bottom AOI gate will have 4 series-connected pFETs. This may induce an unacceptably long delay in a word adder arrangement.

To find a more efficient circuit, consider the nFET array for the carry-out circuit as implied by the AOI logic diagram. Using standard construction gives the nFET circuit in Figure 12.8(a). We see that there are two main pull-down paths corresponding to the terms

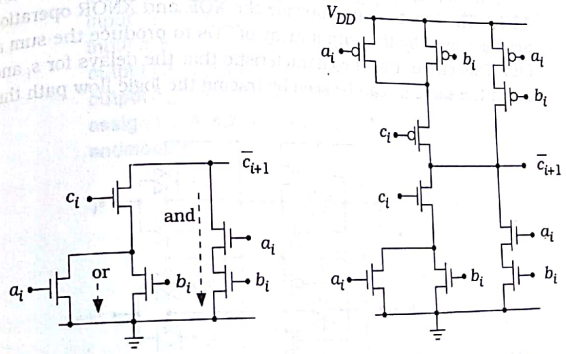
$$\begin{gathered} a_i \cdot b_i \\ c_i \cdot (a_i + b_i) \end{gathered} \tag{12.12}$$

If either of these evaluates to 1, then the output is pulled to 0 (ground). The AND term is important when a_i and b_i are both 1; the OR term gives a pull down if either $a_i = 1$ or $b_i = 1$ while $c_i = 1$. This leads us to construct a pFET mirror circuit to yield the total gate shown in Figure 12.8(b). The series pFETs give a pull up to 1 (V_{DD}) if $a_i = b_i = 0$, which is the opposite of the nFET pull-down condition. If only one is 0, then the output is determined by the value of c_i.

To complete the building of a mirror CMOS full-adder, let us write the sum bit $\bar{s}_i$ as a simple OR gate such that

$$\bar{s}_i = A + B \tag{12.13}$$

where



(a) Standard nFET logic (b) Mirror circuit

Figure 12.8 Evolution of carry-out circuit

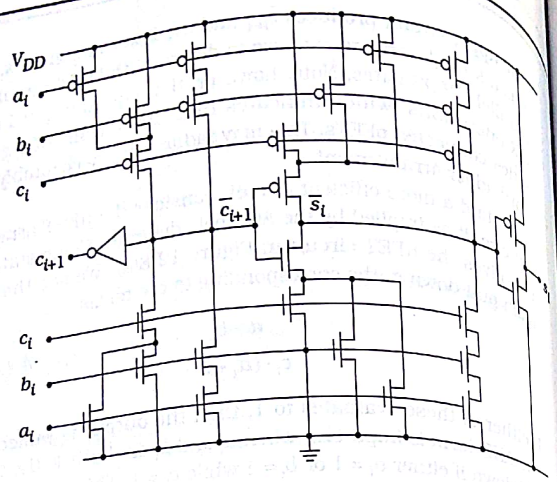


Figure 12.9 Mirror AOI CMOS full-adder

$$A = (a_i + b_i + c_i) \cdot \overline{c}_{i+1}$$
$$B = (a_i \cdot b_i \cdot c_i) \qquad (12.1$$

This has the same characteristics as the carry-out circuit, and allows us to construct the complete full adder shown in Figure 12.9. This is faster than a series-parallel realization, and facilitates the layout because of the mirrored FET arrays.

A full-adder based on transmission gates (TGs) is shown in Figure 12.10. The input circuits provide the XOR and XNOR operations which are then used by the output array of TGs to produce the sum and carry bits. This circuit has the characteristic that the delays for s_i and c_{i+1} are about the same as can be seen by tracing the logic flow path through the

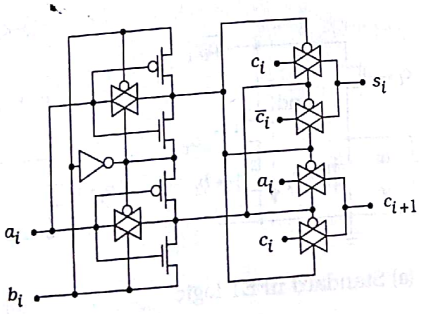


Figure 12.10 Transmission-gate full-adder circuit

upper and lower sections. If the input bits are applied simultaneously, then both the sum and carry-out bits will be valid at about the same time. This is distinctly different from the AOI circuit in which the carry-out bit is produced first and then used in calculating the sum.

12.2 Ripple-Carry Adders

Now that we have a basis for adding single bits, let us extend the problem to adding binary words. In general, adding two n-bit words yields an n-bit sum and a carry-out bit c_n that can either be used as the carry-in to another higher order adder, or act as an overflow flag. A general symbol is shown in Figure 12.11. We will use $n = 4$ in our initial discussions.

Ripple-carry adders are based on the addition equation

$$\begin{array}{r} c_3\, c_2\, c_1\, c_0 \\ +\ a_3 a_2 a_1 a_0 \\ +\ b_3 b_2 b_1 b_0 \\ \hline c_4\, s_3 s_2 s_1 s_0 \end{array} \qquad (12.15)$$

where c_i represents the carry-in bit from the previous column. We will keep the 0-th carry-in bit c_0 for generality. Note that by including the carry-in word this is really adding three binary words. An n-bit ripple-carry adder requires n full-adders with the carry-out bit c_{i+1} used as in the carry-in bit to the next column. This is shown in Figure 12.12 for the case of 4-bit words.

A high-level model can be constructed using Verilog vectors as illustrated by the following code.

```
module four_bit_adder (sum, c_4, a, b, c_0) ;
input [ 3 : 0 ] a, b ;
input c_0 ;
output [ 3 : 0 ] sum ;
output c _4 ;
assign { c _4, sum } = a + b + c_0 ;
endmodule
```

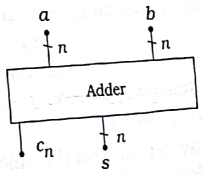


Figure 12.11 An n-bit adder

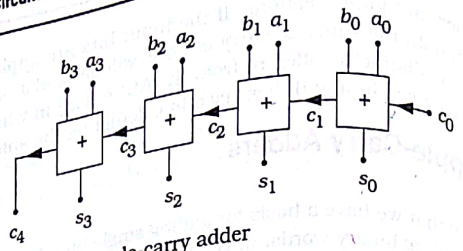


Figure 12.12 A 4-bit ripple-carry adder

This uses concatenation to create a 5-bit output that contains both sum and the carry-out bit c_4. Another approach to modeling this is to use four full-adder modules that are wired together as shown in the drawing.

```
module FA_modules (sum, c_4, a, b, c_0) ;
input [3:0] a, b ;
input c_0 ;
output [3:0] sum ;
output c_4 ;
wire c_1, c_2, c_3 ;
/* The single-bit FA modules instanced below have the syntax
    full_adder (sum, c_out, a, b, c_in) */
full_adder fa0 (sum[0], c_1, a[0], b[0], c_0 ) ;
full_adder fa1 (sum[1], c_2, a[1], b[1], c_1 ) ;
full_adder fa2 (sum[2], c_3, a[2], b[2], c_2 ) ;
full_adder fa3 (sum[3], c_4, a[3], b[3], c_3 ) ;
endmodule
```

This example uses the notation

```
sum[i], a[i], and b[i]
```

to define the i-th bit of a vector. This is straightforward to understand. If we define a quantity such as

```
input [7:0] Q ;
```

with Q = 10001110, then Q[0] = 0, Q[1] = 1, Q[2] = 1, and so on. As always, it is assumed that instanced modules such as

```
full_adder (sum, c_out, a, b, c_in) ;
```

are defined elsewhere in the listing. They may be written at any level from a behavioral description down to a gate-level structural listing.

The ripple-carry adder construction provides for easy connections of neighboring circuits. It is this feature, however, that makes the design slow. Since the output of any full-adder is not valid until the incoming carry bit is valid, the left-most circuit is the last to react. The word result is not valid until this occurs.

The overall delay depends on the characteristics of the full-adder circuits. Different CMOS implementations will produce different worst-case delay paths. For our purposes, let us assume that the AOI mirror CMOS

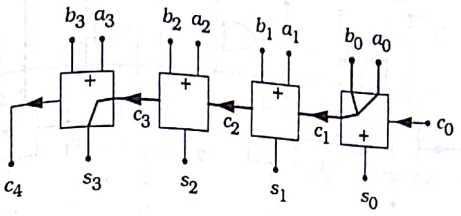


Figure 12.13 Worst-case delay through the 4-bit ripple adder

full-adder in Figure 12.9 is used in the 4-bit network. Since the carry-out is required to calculate the sum, the carry delay from c_i to c_{i+1} is minimized. Figure 12.13 shows the longest delay path for the adder where the carry bits are transferred through every stage; it is assumed that all inputs are valid at the same time. Let us start by summing the individual delays to get the total delay t_{4b} as

$$t_{4b} = t_{d3} + t_{d2} + t_{d1} + t_{d0} \tag{12.16}$$

where t_{di} is the worst-case delay through the i-th stage. The contributions for each stage can be evaluated. For the 0-th bit, $t_{d0} = t_d\,(a_0,\ b_0 \rightarrow c_1)$, which is the time for the inputs to produce the carry-out bit. The delay through sections 1 and 2 is the same, and is from the carry-in to the carry-out: $t_{d1} = t_{d2} = t_d\,(c_{in} \rightarrow c_{out})$. Finally, the delay in the last stage 3 in this design is the time needed to produce the output sum bit s_3, which we write as $t_{d3} = t_d\,(c_{in} \rightarrow s_3)$. Thus, the total delay is

$$t_{4b} = t_d(c_{in} \rightarrow s_3) + 2t_d(c_{in} \rightarrow c_{out}) + t_d(a_0, b_0 \rightarrow c_1) \tag{12.17}$$

If we extend this to an n-bit ripple-carry adder, then the worst-case delay is

$$t_{n-bit} = t_d(c_{in} \rightarrow s_{n-1}) + (n-2)t_d(c_{in} \rightarrow c_{out}) + t_d(a_0, b_0 \rightarrow c_1) \tag{12.18}$$

which shows that the delay is of order n. Symbolically, we express this as

$$\text{delay} \sim O(n) \tag{12.19}$$

The ripple structure is therefore not a good choice for large word sizes.

Before progressing into more advanced adder designs, let us recall that a 2's complement subtractor can be built by adding XOR gates and an *add_sub* control bit as shown in Figure 12.14. When *add_sub* = 0, the XORs pass the b_i bits and the output is the sum $(a+b)$. A control bit of *add_sub* = 1 changes the XORs into inverters, and the complemented values $\overline{b_i}$ enter the full adders; *add_sub* = 1 also acts as a carry-in of $c_0 = 1$. These operations combine to give the 2's complement algorithm for the

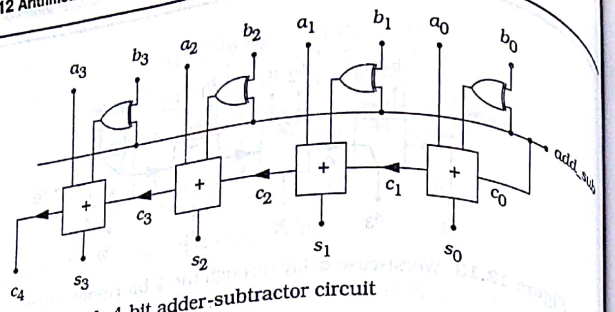


Figure 12.14 4-bit adder-subtractor circuit

difference (a - b). This technique is also applicable in a limited manner to other adder networks.

12.3 Carry Look-Ahead Adders

Carry look-ahead (CLA) adders are designed to overcome the latency introduced by the rippling effect of the carry bits. The CLA algorithm is based on the origin of the carry-out bit in the equation

$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i) \tag{12.20}$$

for the cases that give $c_{i+1} = 1$. Since either term may cause this output, we treat the two separately. First, if $a_i \cdot b_i = 1$, then $c_{i+1} = 1$. We call

$$g_i = a_i \cdot b_i \tag{12.21}$$

the **generate** term, since the inputs are viewed as "generating" the carry-out bit. Note that if $g_i = 1$, then we must have $a_i = b_i = 1$. The second term represents the case where an input carry $c_i = 1$ may be "propagated" through the full-adder. This will happen if the **propagate** term

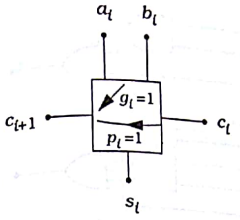
$$p_i = a_i \oplus b_i \tag{12.22}$$

is equal to 1; if $p_i = 1$ then $g_i = 0$ since the XOR operation produces a 1 iff the inputs are not equal. Figure 12.15 shows the behavior of the generate and propagate terms. With these definitions, the equation for the carry-out bit is

$$c_{i+1} = g_i + p_i \cdot c_i \tag{12.23}$$

The main idea of the CLA is to first calculate the values of p_i and g_i for every bit, then use them to find the carry bits c_{i+1}. Once these are found, the sum bits are given by

$$s_i = p_i \oplus c_i \tag{12.24}$$



	g_i	p_i
	$a_i \; b_i$	$a_i \oplus b_i$
$a_i = b_i = 0$	0	0
$a_i = b_i = 1$	1	0
$a_i \neq b_i$	0	1

$c_{i+1} = a_i \; b_i + c_i \cdot (a_i \oplus b_i)$

Figure 12.15 Basis of the carry look-ahead algorithm

for every i. This avoids the need to ripple the carry bits serially down the chain.

Let us analyze the 4-bit CLA equations. With c_0 assumed known, we have

$$c_1 = g_0 + p_0 \cdot c_0 \tag{12.25}$$

The expressions for c_2, c_3, and c_4 have the same form with

$$\begin{aligned} c_2 &= g_1 + p_1 \cdot c_1 \\ c_3 &= g_2 + p_2 \cdot c_2 \\ c_4 &= g_3 + p_3 \cdot c_3 \end{aligned} \tag{12.26}$$

These can be expressed using primitive generate and propagate terms by noting that c_i can be substituted into c_{i+1} in succession. The first reduction is obtained by substituting c_1 into the c_2 equation to arrive at

$$c_2 = g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0) \tag{12.27}$$

Expanding,

$$c_2 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \; c_0 \tag{12.28}$$

Similarly, substituting c_2 into c_3 gives

$$\begin{aligned} c_3 &= g_2 + p_2 \cdot (g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0) \\ &= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0 \end{aligned} \tag{12.29}$$

Finally, the carry-out bit is

$$\begin{aligned} c_4 &= g_3 + p_3 \cdot (g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0) \\ &= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0 \end{aligned} \tag{12.30}$$

These equations show that every carry bit can be found from the generate and propagate terms. Moreover, the algorithm yields nested SOP expressions. The logic diagram for the 4-bit network is shown in Figure 12.16

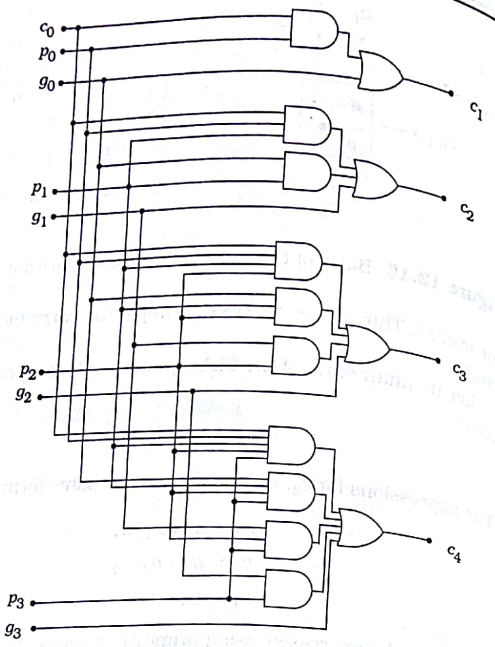


Figure 12.16 Logic network for 4-bit CLA carry bits

using the expanded expressions. Note the structured nature of the gate arrangement. Once the carry-out bits have been calculated, the sums are found using the simple XOR in equation (12.24). The complete adder circuit is shown in Figure 12.17 where the "CLA Network" box represents the carry bit logic in Figure 12.16. This illustrates a marked departure from the ripple-carry design.

The high-level abstract Verilog description of a 4-bit adder can be used to describe any adder, including the CLA-based design. However, we can rewrite the behavioral code to better illustrate the internal algorithm in an explicit manner. The **assign**-based RTL module below illustrates this idea.

```
module CLA_4b (sum, c_4, a, b, c_0 ) ;
input [ 3 : 0 ] a, b ;
input c_0 ;
output [ 3 : 0 ] sum ;
output c _4 ;
wire p0, p1, p2, p3, g0, g1, g2, g3 ;
wire c1, c2, c3, c4 ;
```

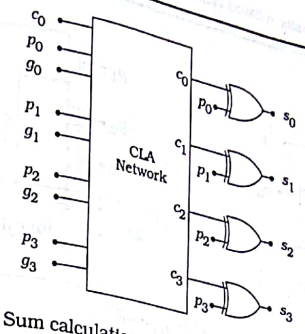



Figure 12.17 Sum calculation using the CLA network

```
assign
    p0 = a[0] ^ b[0] ,
    p1 = a[1] ^ b[1] ,
    p2 = a[2] ^ b[2] ,
    p3 = a[3] ^ b[3] ,
    g0 = a[0] & b[0] ,
    g1 = a[1] & b[1] ,
    g2 = a[2] & b[2] ,
    g3 = a[3] & b[3] ;
assign
    c1 = g0 | ( p0 & c_0 ) ,
    c2 = g1 | ( p1 & g0 ) | ( p1 & p0 & c_0 ) ,
    c3 = g2 | ( p2 & g1 ) | ( p2 & p1 & g0 ) | ( p2 & p1 & p0 & c_0 ) ,
    c4 = g3 | ( p3 & g2) | ( p3 & p2 & g1 ) | ( p3 & p2 & p1 & g0 )
       | ( p3 & p2 & p1 & p0 & c_0 ) ;
assign
    sum [0] = p0 ^ c_0 ,
    sum [1] = p1 ^ c1 ,
    sum [2] = p2 ^ c2 ,
    sum [3] = p3 ^ c3 ,
    c_4 = c4 ;
endmodule
```

Adding delay times on each statement provides the final information needed for a simulation. The repetitive nature of the CLA equations can be implemented in a more efficient coding style by using the Verilog **for** procedure.

To translate the CLA algorithms into circuits, we use the logic construction techniques developed in Chapter 2 to create the nFET arrays shown in Figure 12.18. Note that each carry-out circuit $\bar{c}_i$ forms the basis

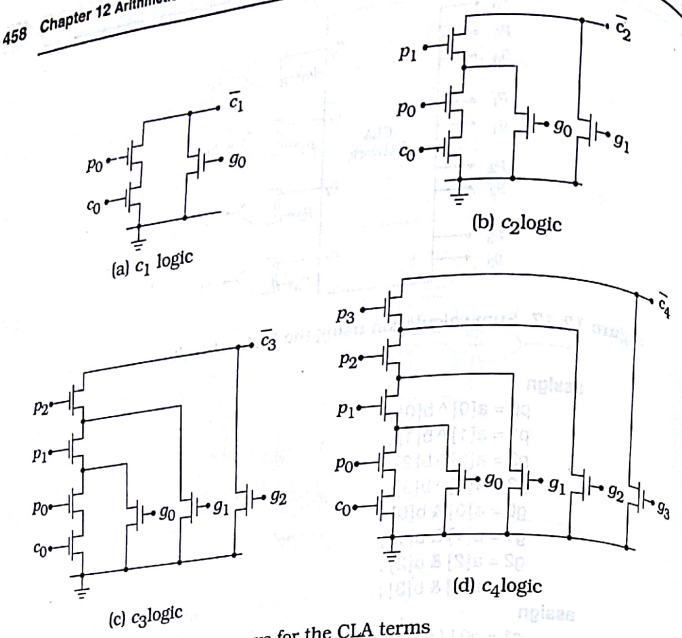


Figure 12.18 nFET logic arrays for the CLA terms

for the next higher term $\bar{c}_{i+1}$. This is due to the nesting property of the algorithm.

Once the nFET logic is designed, it can be used in a variety of circuits. Figure 12.19 shows three possibilities. The structure in Figure 12.19(a) represents standard complementary structuring where we create a pFET array using bubble pushing to obtain the series-parallel pFET array. The static pseudo-nMOS approach in Figure 12.19(b) could be used, but we would have to be concerned about device ratios to insure that the output low voltage V_{OL} is sufficiently small without using excessively large nFETs. This is avoided if we opt for a dynamic logic family as in Figure 12.19(c). This, however, introduces timing problems and gives outputs that are only valid for a short period of time. Obviously, the selection of the circuit family involves considering many factors.

Let us examine the possibility of using full complementary static circuits. The $\bar{c}_1$ circuit in Figure 12.20(a) shows the complete nFET/pFET arrays with series-parallel structuring. This, however, has a form similar to the carry-out circuit analyzed earlier in Figure 12.8, so that we may create the mirror-equivalent logic gate shown in Figure 12.20(b). The pro-

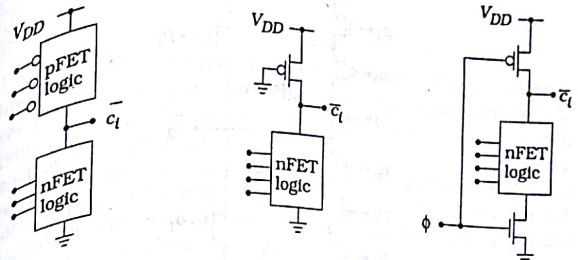


Figure 12.19 Possible uses of the nFET logic arrays in Figure 12.18

cess can be continued for the remaining bits. For example, Figure 12.21 shows the mirror circuit for $\bar{c}_2$. Note the symmetry in the arrays. This feature allows for a more structured layout at the physical design level.

Another approach is to use multiple-output domino logic (MODL) as a basis. This is possible because the nesting of the carry bits from one bit to the next gives the ANDing relationship needed to implement MODL. To see this analytically, recall that we had

$$\begin{aligned} c_1 &= g_0 + p_0 \cdot c_0 \\ c_2 &= g_1 + p_1 \cdot c_1 \\ &= g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0) \end{aligned} \tag{12.31}$$

We can use c_1 as one output, and c_2 as another output with the two related by the AND operation. Since this type of relationship is valid for c_3 and c_4, only a single MODL gate is needed to produce all four carry bits.

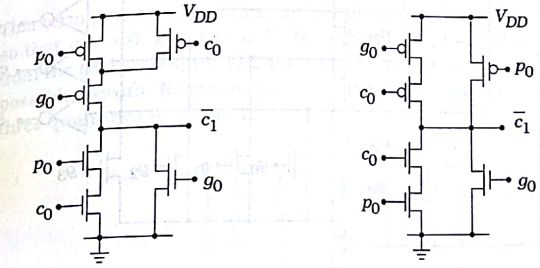


Figure 12.20 Static CLA mirror circuit

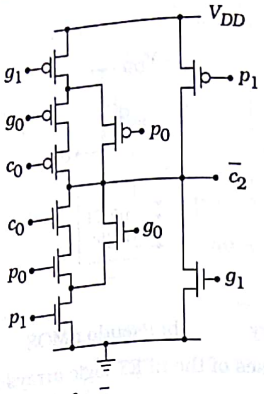


Figure 12.21 Static mirror circuit for $\bar{c}_2$

Moreover, MODL is a non-inverting logic family with the inverters built into the structure. Figure 12.22 shows the 4-bit MODL carry circuit where the logic array provides separate outputs for each carry bit. A precharge pFET has been added for every internal node. When the circuit is in precharge with the clock at $\phi = 0$, all of the outputs are driven to 0. The logic network accepts the inputs during the evaluation phase ($\phi = 0$). If the c_1 network allows for a discharge of the internal node and produces $c_1 =$ 1, then one of the conditions

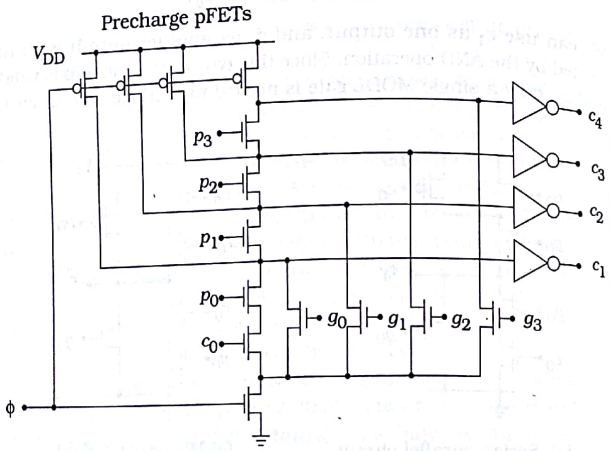


Figure 12.22 MODL carry circuit

$$g_0 = 1, \text{ or}$$
$$p_0 \cdot c_0 = 1 \tag{12.32}$$

holds. If $p_0 \cdot c_0 = 1$ then a value of $p_1 = 1$ will drive c_2 to 1. Alternately, the carry-out may be generated with $g_1 = 1$. This type of interdependence continues upward in the logic network to produce c_3 and c_4. The ability to use a single logic gate to produce the four carry-out bits is very attractive. The layout will have the complexity of a single c_4 mirror gate with fewer transistors. However, it must be remembered that MODL is a dynamic circuit technique, and is subject to the usual limitations: clocking is mandatory, the output is subject to charge leakage and charge sharing, and the series-connected nFET chains will give long discharge times unless large FETs are used.

12.3.1 Manchester Carry Chains

The Manchester carry scheme is a particularly elegant approach to dealing with CLA bits. It is based on building a switch-logic network for the basic equation

$$c_{i+1} = g_i + p_i \cdot c_i \tag{12.33}$$

that can be cascaded to feed to successively stages.

Consider a full adder with inputs a_i, b_i, and c_i. We will use the generate and propagate expressions

$$g_i = a_i \cdot b_i$$
$$p_i = a_i \oplus b_i \tag{12.34}$$

to introduce the **carry-kill** bit k_i such that

$$k_i = \overline{a_i + b_i}$$
$$= \bar{a}_i \cdot \bar{b}_i \tag{12.35}$$

This term gets its name from the fact that if $k_i = 1$, then $p_i = 0$ and $g_i = 0$, so that $c_{i+1} = 0$; $k_i = 1$ thus "kills" the carry-out bit. This can be verified from the table in Figure 12.23 that shows the values of p_i, g_i, and k_i for all possible inputs. Note that for a given input set (a_i, b_i), only one of the three quantities is a logic 1

a_i	b_i	p_i	g_i	k_i
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	0	1	0

Figure 12.23 Propagate, generate, and carry-kill values

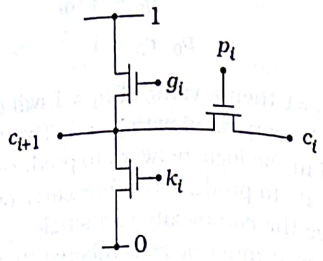
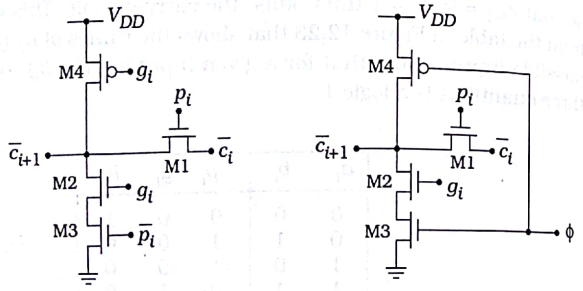


Figure 12.24 Switching network for the carry-out equation

The Manchester carry scheme is based on this behavior. Since only one of the three quantities p_i, g_i, and k_i can be a 1, we can construct the switch-level circuit using FETs shown in Figure 12.24. The topology has been chosen such that only one FET is a closed switch at a time. The operation can be understood by examining each possibility. First, if we have $(a_i, b_i) = (0, 0)$, then $k_i = 1$ and $c_{i+1} = 0$. If $a_i \neq b_i$, then $p_i = 1$ and the input bit c_i is propagated through the circuit to give $c_{i+1} = c_i$. Finally, an input of $(a_i, b_i) = (1, 1)$ indicates that a carry-out has been generated by a term $g_i = 1$, so $c_{i+1} = 1$. At the circuit level, it is important to note that using only nFETs induces a threshold voltage drop on a logic 1 transmission through the transistor.

Several different Manchester carry circuits can be built. Two are shown in Figure 12.25. The static logic gate in Figure 12.25(a) uses $\bar{c}_i$ as an input. First, suppose that $p_i = 0$. This opens M1 and blocks the input $\bar{c}_i$ from propagating through, but also turns on nFET M3. If $g_i = 0$, pFET M4 is on and pulls the output to $\bar{c}_{i+1} = 1$. If $g_i = 1$, then both nFETs M2 and M3 are on while M4 is off, giving an output of $\bar{c}_{i+1} = 0$. The case where $p_i = 1$ is more complicated. The generate term g_i must be 0, so pFET M4 is on while the nFET chain acts as an open circuit since M3 is off. The output is



(a) Static circuit (b) Dynamic circuit

Figure 12.25 Manchester circuit styles

then controlled by $\bar{c}_i$. If $\bar{c}_i = 1$, then this is transmitted to the output and supported by the pFET connection to the power supply so that $\bar{c}_{i+1} = 1$. However, if $\bar{c}_i = 0$, then the circuit reduces to a pseudo-nMOS inverter made up of M4 and M1, with $p_i = 1$ at the input.[1] To obtain a low output of $\bar{c}_{i+1} = 0$ we must choose the nFET/pFET size ratio to be large enough to give a low output voltage.

A dynamic circuit is shown in Figure 12.25(b). The logic is similar to the static design except that the evaluation nFET M3 replaces a logic transistor. During the precharge ($\phi = 0$), the output node is brought to a logic 1 voltage. Evaluation takes place when the clock switches to $\phi = 1$. A carry propagation occurs if $p_i = 1$, while the node discharges to 0 if $g_i = 1$. This circuit can be used to build the Manchester carry chain shown in Figure 12.26. Every stage undergoes precharge when $\phi = 0$. The carry bits are available during the evaluation time with the longest time delay for c_4.

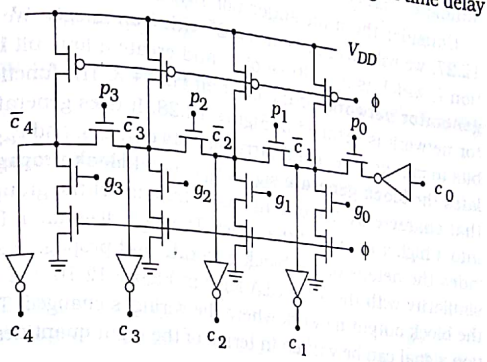


Figure 12.26 Dynamic Manchester carry chain

12.3.2 Extension to Wide Adders

The carry look-ahead equations can be extended to adders wider than 4-bits, but one must be careful of hardware delays due to the increased gate count through the longest delay path. For example, if we use a brute-force approach for an 8-bit design, then the carry-out bit c_8 would have a term of the form

$$p_7 \cdot p_6 \cdot p_5 \cdot p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0 \tag{12.36}$$

that would have to be dealt with. Various techniques have been published to obtain more efficient CLA networks for wide adders. Consider the addi-

[1] This can be seen by grounding the input $\bar{c}_i$ and covering up the M2-M3 transistors.

tion of two n-bit words. Work by von Neumann and others has shown that the longest carry chain has an average length of[2]

$$\log_2(n) \tag{12.37}$$

For example, the average carry chain in an 8-bit adder is

$$\log_2(8) = \log_2(2^3) = 3 \tag{12.38}$$

while a 32-bit adder has an average length

$$\log_2(32) = \log_2(2^5) = 5 \tag{12.39}$$

This implies that the length of the carry circuits does not have to span the entire length of the word, but can be broken up into smaller segments. Multilevel CLA networks are based on this philosophy.

Consider the n-bit adder portrayed at architectural level in Figure 12.27; we will assume that $n = 2^k$ with k an integer. We select a bit position i, which is a multiple of 4, and create a four-bit **lookahead carry generator network** for the bit from i to $i + 3$. The function of the generator network is detailed in Figure 12.28. It uses generate and propagate bits to produce the usual carry-out bits c_{i+1}, c_{i+2}, and c_{i+3}, but also calculates the **block generate** signal $g_{[i,i+3]}$ and **block propagate** signal $p_{[i,i+3]}$ that characterize the overall characteristics of the group and can be fed into a higher section of the adder. The logic diagram in Figure 12.29 provides the details of the block generate and propagate signals. Note the similarity with the 4-bit CLA logic in Figure 12.16; the difference lies in the block output network where the wiring is changed. The block generation signal can be written in terms of the input quantities as

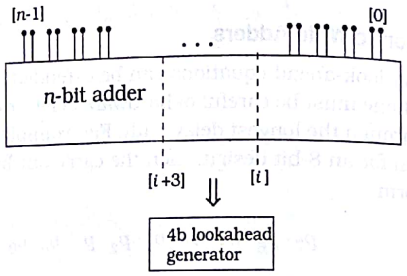


Figure 12.27 An n-bit adder network

[2] See Reference [4].

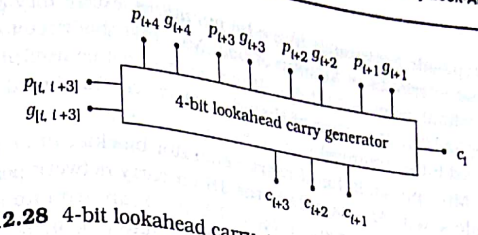


Figure 12.28 4-bit lookahead carry generator signals

$$g_{[i,i+3]} = g_{i+3} + p_{i+3} \cdot g_{i+2} + p_{i+3} \cdot p_{i+2} \cdot g_{i+1} + p_{i+3} \cdot p_{i+2} \cdot p_{i+1} \cdot g_i \tag{12.40}$$

and is taken out of the gate labeled **or1** in the diagram. The block propagation is

$$p_{[i,i+3]} = p_{i+3} \cdot p_{i+2} \cdot p_{i+1} \cdot p_i \tag{12.41}$$

which is the output of gate **and1** in the diagram. The block generate and

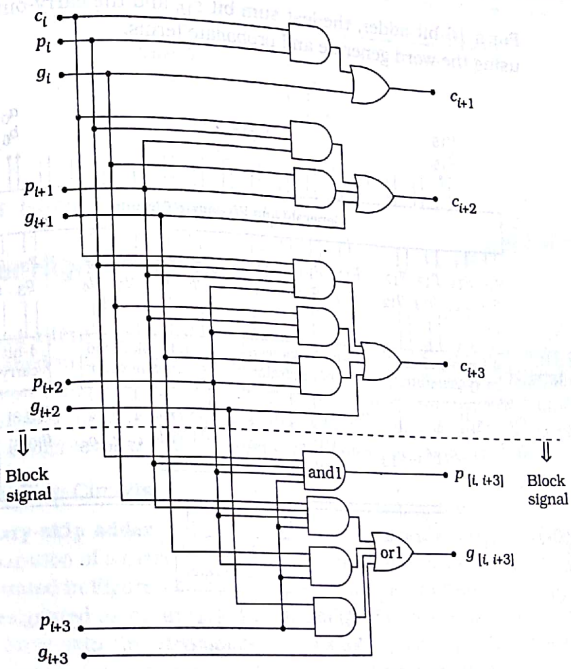


Figure 12.29 Block lookahead generator logic

propagate are similar to the bit quantities, except they provide the overall characteristics of a group of bits. Note that this circuit does not compute the final carry-out bit c_{i+4}. It may or may not be needed, depending upon the overall structure of the adder network. Additional logic can be provided if it is required.

Multiple lookahead carry generator blocks can be used to design a wide adder. An example is the 16-bit carry network portrayed in Figure 12.30. The inputs $a_{15} \ldots a_0$ and $b_{15} \ldots b_0$ are fed into the generate and propagate network that produces the values $(p_{15}, g_{15}), \ldots, (p_0, g_0)$ for use in the CLA blocks. The CLA subsystem is usually described in levels. At Level 1, four 4-bit lookahead carry generator networks are used to provide the carry-out bits c_{i+3}, c_{i+2}, c_{i+1}, and the block generate and propagate terms $g_{[i,i+3]}$ and $p_{[i,i+3]}$ for $i = 0, 4, 8, 12$. The block terms are then sent to the single Level 2 4-bit lookahead carry network. The Level 2 block produces carry-out bits c_4, c_8, c_{12}, and the word generate and propagate terms $g_{[0,15]}$ and $p_{[0,15]}$. At this point, all of the carry bits except c_{15} have been calculated for use in the sum equation

$$s_i = p_i \oplus c_i \tag{12.42}$$

For a 16-bit adder, the last sum bit s_{15} and the carry-out can be found using the word generate and propagate terms.

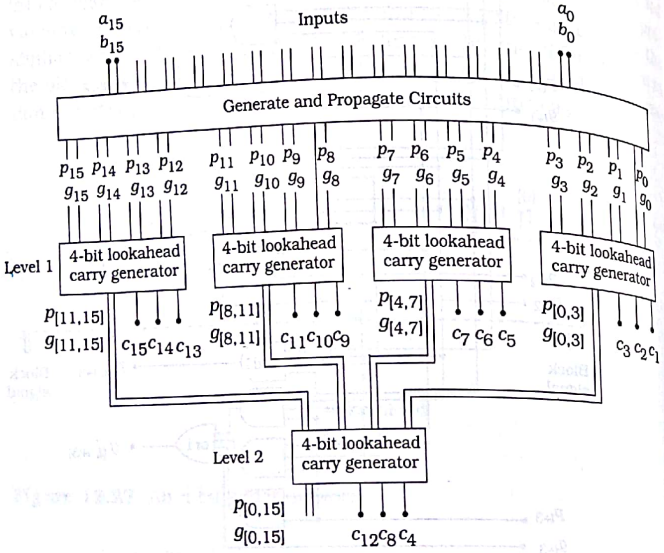


Figure 12.30 Multilevel CLA block scheme for a 16-bit adder

A 64-bit adder can be obtained by adding another level of lookahead carry blocks to the 16-bit network. The scheme is shown in Figure 12.31. Four 16-bit blocks are used to produce four sets of group generate and propagate terms. These are then fed into the Level 3 block that provides the final carry-out bits. It is important to note that each block produces carry-out bits for use in the sum calculations. The carry-out bits are available at times that vary with the level where the circuitry is. Level 1 bits are available first, Level 2 bits second, and Level 3 bits are the final ones out of the network. There is no *a priori* reason for using 4-bit lookahead carry generator circuits; smaller or larger widths are acceptable.

We have examined the basic concepts involved in CLA structures here. The interested reader is directed to Reference [2] for a more detailed discussion.

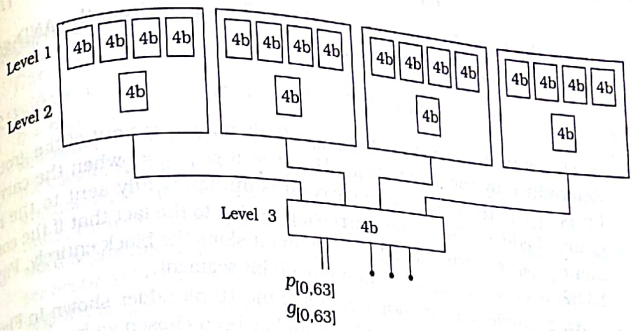


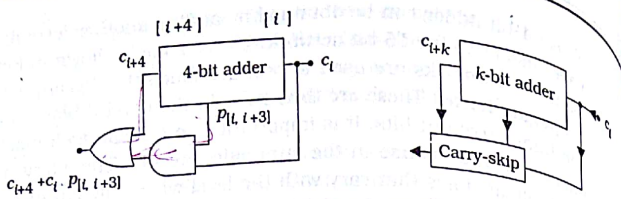
Figure 12.31 64-bit CLA adder architecture

12.4 Other High-Speed Adders

Several alternate approaches to designing fast word adders have been published in the literature. All have the objective of decreasing the computation time, and each has different trade-offs. This section examines a few of these designs illustrating the variations that one has in translating from a high-level architectural description down to the circuit level.

12.4.1 Carry-Skip Circuits

A **carry-skip adder** is designed to speed up a wide adder by aiding the propagation of a carry bit around a portion of the entire adder. The idea is illustrated in Figure 12.32(a) for the case of a 4-bit adder. The carry-in bit is designated as c_i, and the adder itself produces a carry-out bit of c_{i+4}. The carry-skip circuitry consists of two logic gates. The AND gate accepts the carry-in bit and compares it to the group propagate signal



(a) Carry-skip logic (b) Generalization

Figure 12.32 Carry-skip circuitry

$$P_{[i,\,i+3]} = P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot P_i \tag{12.43}$$

using the individual propagate values. The output from the AND gate is ORed with c_{i+4} to produce a stage output of

$$carry = c_{i+4} + P_{[i,\,i+3]} \cdot c_i \tag{12.44}$$

as shown in the drawing. If $P_{[i,i+3]} = 0$, then the carry-out of the group is determined by the value of c_{i+4}. However, if $P_{[i,i+3]} = 1$ when the carry-in bit is $c_i = 1$, then the group carry-in is automatically sent to the next group of adders. The name "carry-skip" is due to the fact that if the condition $P_{[i,i+3]} \cdot c_i$ is true, then the carry-in bit skips the block entirely. Figure 12.32(b) shows the generalization to a k-bit segment.

An example of carry-skip circuits is the 16-bit adder shown in Figure 12.33. The size of the carry-skip group has been chosen as $k = 4$ for every segment. The worst-case delay through this circuit is when $c_0 = 0$ and the 0-th bit adder produces a carry-out bit of $c_1 = 1$. If ripple adders are used, then the worst-case situation is where this bit emerges as $c_4 = 1$, and then skips the next segment groups [7,4] and [11,8] and enters the final block where it ripples through to the output as $c_{16} = 1$.

The size k of a carry-skip block affects the overall speed of the scheme. It has been shown that the optimal block size for an n-bit adder that minimizes the delay can be estimated as

$$k = \sqrt{\frac{n}{2}} \tag{12.45}$$

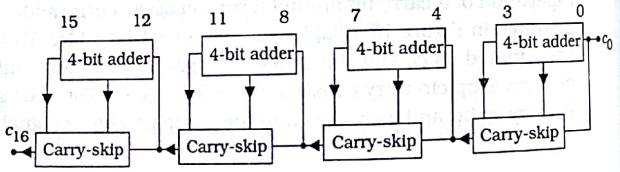


Figure 12.33 A 16-bit adder using carry-skip circuits

For $n = 16$, the block size would be $k \approx 3$. Alternately, a variable k-value can be used. The carry-skip circuits can be nested to create multilevel networks. Figure 12.34 shows an example of a 2-level carry-skip adder.

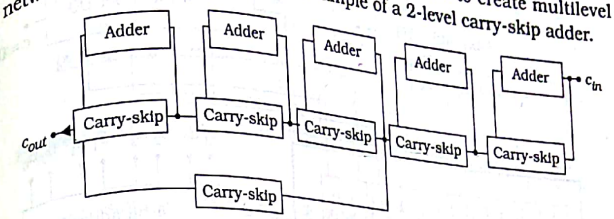


Figure 12.34 A 2-level carry-skip adder

12.4.2 Carry-Select Adders

Carry-select adders use multiple narrow adders to create fast wide adders. Consider the addition of two n-bit numbers with $a = a_{n-1} \ldots a_0$ and $b = b_{n-1} \ldots b_0$. At the bit level, the adder delay increases from the least significant 0-th position upward, with the $(n-1)$-th requiring the most complex logic. A carry-select adder breaks the addition problem into smaller groups. For example, we can split the n-bit problem into two $(n/2)$-bit sections, then give special attention to the higher order group that adds the word segments $a_{n-1} \ldots a_{n/2}$ and $b_{n-1} \ldots b_{n/2}$. The carry delay will then center around the carry-out bit $c_{n/2}$ produced by the sum of lower order word segments $a_{(n/2)-1} \ldots a_0$ and $b_{(n/2)-1} \ldots b_0$. We know that there are only two possibilities for the carry bit:

$$c_{n/2} = 0 \text{ or } c_{n/2} = 1 \tag{12.46}$$

A carry-select adder provides two separate adders for the upper words, one for each possibility. A MUX is then used to select the valid result.

As a concrete example, consider an 8-bit adder that is split into two 4-bit groups. The lower-order bits $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ are fed into the 4-bit adder L to produce the sum bits $s_3s_2s_1s_0$ and a carry-out bit c_4 as shown in Figure 12.35. The higher order bits $a_7a_6a_5a_4$ and $b_7b_6b_5b_4$ are used as inputs to two 4-bit adders. Adder U0 calculates the sum with a carry-in of $c = 0$, while U1 does the same only it has a carry-in value of $c = 1$. Both sets of results are used as inputs to an array of 2:1 MUXes. The carry bit c_4 from the adder L is used as the MUX select signal. If $c_4 = 0$, then the results of U0 are sent to the output, while a value of $c_4 = 1$ selects the results of U1 for $s_7s_6s_5s_4$. The carry-out bit c_8 is also selected by the MUX array.

This design speeds up addition of the word by allowing the upper and lower portions of the sum to be calculated simultaneously. The price paid is that it requires an additional word adder, a set of multiplexors, and the

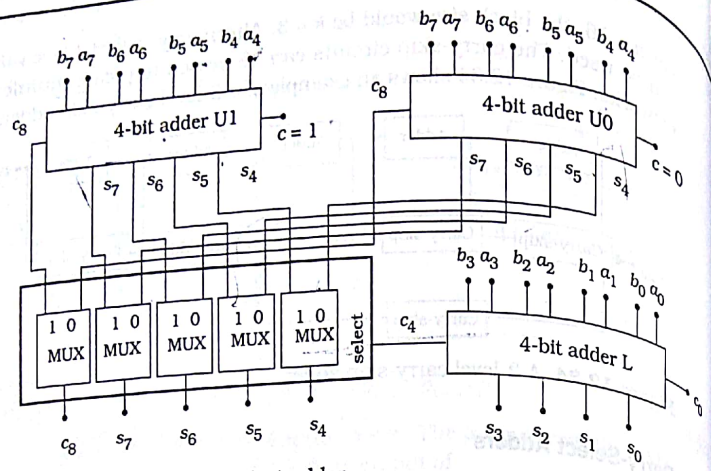
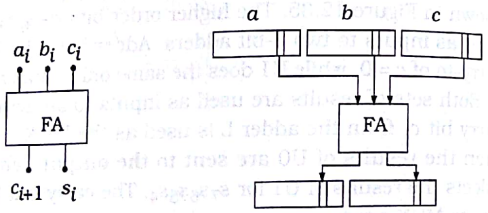


Figure 12.35 8-bit carry-select adder

associated interconnect wiring. The design becomes viable if speed is more important than area consumption. Carry-select adders can be cascated using multiple levels, but the hardware costs increase accordingly.

12.4.3 Carry-Save Adders

Carry-save adders are based on the idea that a full-adder really has three inputs and produces two outputs as shown in Figure 12.36(a). While we usually associate the third input with a carry-in, it could equally well be used as a "regular" value. In Figure 12.36(b), the FA is used as a 3-2 reduction network where it starts with bits from 3 words, adds them, and then has an output that is 2-bits wide. We can build an n-bit carry-save adder by using n separate adders as in Figure 12.37. The name "carry-save" arises from the fact that we save the carry-out word instead of using it immediately to calculate a final sum.



(a) Symbol (b) 3-to-2 reduction

Figure 12.36 Basis of a carry-save adder

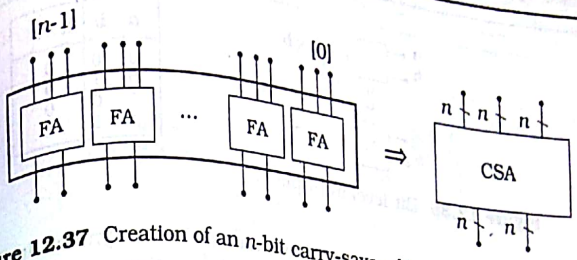


Figure 12.37 Creation of an n-bit carry-save adder

Carry-save adders (CSAs) are useful in situations where we need to add more than two numbers. Since the design automatically avoids the delay in the carry-out bits, a CSA chain may be faster than using standard adders or cycling with a clocked synchronous network.

An example is the 7-to-2 reduction scheme shown in Figure 12.38. This starts with 7 n-bit words a, b, ..., g and uses five CSA units to reduce it down to two words. If we want a final sum, then a normal CPA (carry-propagate adder) can be used at the bottom of the chain to add the two values together.

12.5 Multipliers

Binary multiplication is based on the basic operations

$$0\times0 = 0\,,\quad 0\times1 = 0\,,\quad 1\times0 = 0\,,\quad 1\times1 = 1 \tag{12.47}$$

If we multiply two bits a and b, then we see that the logical AND operation

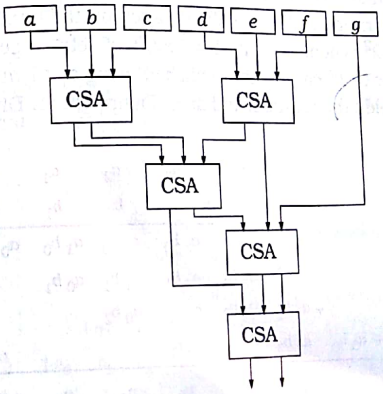
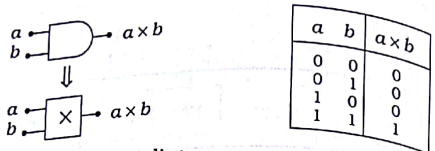


Figure 12.38 A 7-to-2 reduction using carry-save adders



a	b	a × b
0	0	0
0	1	0
1	0	0
1	1	1

Figure 12.39 Bit-level multiplier

produces the same result as summarized by the symbols in Figure 12.39. Bit-level multiplication is thus a trivial operation.

The complexity arises when we multiply n-bit words. Let us specify a word length of $n = 4$ to illustrate the main ideas. With input values given by $a = a_3a_2a_1a_0$ and $b = b_3b_2b_1b_0$, the product $a \times b$ is given by the 8-bit ($2n$) result

$$p = p_7p_6p_5p_4p_3p_2p_1p_0 \tag{12.48}$$

as shown in Figure 12.40. Each bit b_i multiplies the multiplicand a on a bit-by-bit basis. The product term from least significant bit b_0 is aligned to the multiplicand, while the next term (due to b_1) is shifted one column left. The array builds until every bit of the multiplier is used. The product bits p_i are obtained by summing each of the i-th columns, accounting for a carry from the (i-1)-th column. A simple expression is

$$p_i = \sum_{i=j+k} a_j b_k + c_{i-1} \tag{12.49}$$

where $c_{i-1} = 0$ for $(i - 1) \le 0$.

A special case worth remembering is multiplication by 2. For an 8-bit word, this corresponds to an input of $b = 00000010$, which is equivalent to performing shift left (shl) operation on the multiplicand. Multiplication by $4 = 2^2$ is achieved with $b = 00000100$, etc. In general, multiplication by a factor of 2^m can be accomplished using a shl_m operation on a register that holds the multiplicand as in Figure 12.41. Division by 2^k is obtained

$$\begin{array}{rcccccccl}
 & & & a_3 & a_2 & a_1 & a_0 & & \text{multiplicand} \\
 & & \times & b_3 & b_2 & b_1 & b_0 & & \text{multiplier} \\
\hline
 & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & & \\
 & & + & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & \\
 & & + a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & \\
 & + a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 & \text{product}
\end{array}$$

Figure 12.40 Multiplication of two 4-bit words

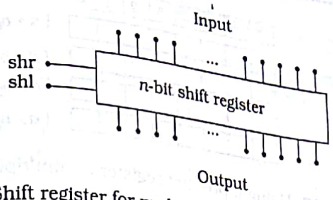


Figure 12.41 Shift register for multiplication or division by a factor of 2

with a shift right (shr_k) command. Bit overflow should be monitored to insure that the results are valid.

A high-level Verilog description of the 4 × 4-bit multiplier can be written as

```
module mult_4 (product, a , b) ;
input [ 3 : 0 ] a, b ;
output [ 7 : 0 ] product ;
assign #(t_delay) product = a * b ;
endmodule
```

for use in initial architectural simulations. The word size can be adjusted as needed using the vector specification. However, the critical time delay parameter t_delay depends upon the implementation, and accurate values are necessary to establish a sound design.

The details of the multiplication procedure yield specific techniques for calculating the product with binary switching networks. One way to view the process is that the bit b_i multiplies the entire word a; each term ($a \times b_i$) has a base-10 weighting of 2^i. This is shown in Figure 12.42. Starting from the first line $(a \times b_0) \times 2^0$ and working down, each factor of 2^i represents a shift in position. A simple view of this process is to use a product register as shown in Figure 12.43. Each term ($a \times b_i$) occupies a shifted position such that the product is obtained by adding the terms in each column.

$$\begin{array}{rcccccccl}
 & & & a_3 & a_2 & a_1 & a_0 & & \text{multiplicand} \\
 & & \times & b_3 & b_2 & b_1 & b_0 & & \text{multiplier} \\
\hline
 & & & (a_3 & a_2 & a_1 & a_0) \times b_0 & & (a \times b_0)\,2^0 \\
 & & (a_3 & a_2 & a_1 & a_0) \times b_1 & & & (a \times b_1)\,2^1 \\
 & (a_3 & a_2 & a_1 & a_0) \times b_2 & & & & (a \times b_2)\,2^2 \\
+ & (a_3 & a_2 & a_1 & a_0) \times b_3 & & & & (a \times b_3)\,2^3 \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 & \text{product}
\end{array}$$

Figure 12.42 Alternate view of multiplication process

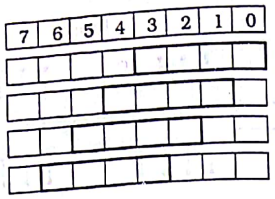


Figure 12.43 Using a product register for multiplication

A practical implementation is based on the sequence illustrated in Figure 12.44. The left side of the register allows for parallel loading of a 4-bit word. The product is created by successive addition and shift-right operations as shown. Note that the carry-out bits are tracked by shifting them into the left register. For the multiplication of two n-bit words, the algorithm for the product can be expressed as[3]

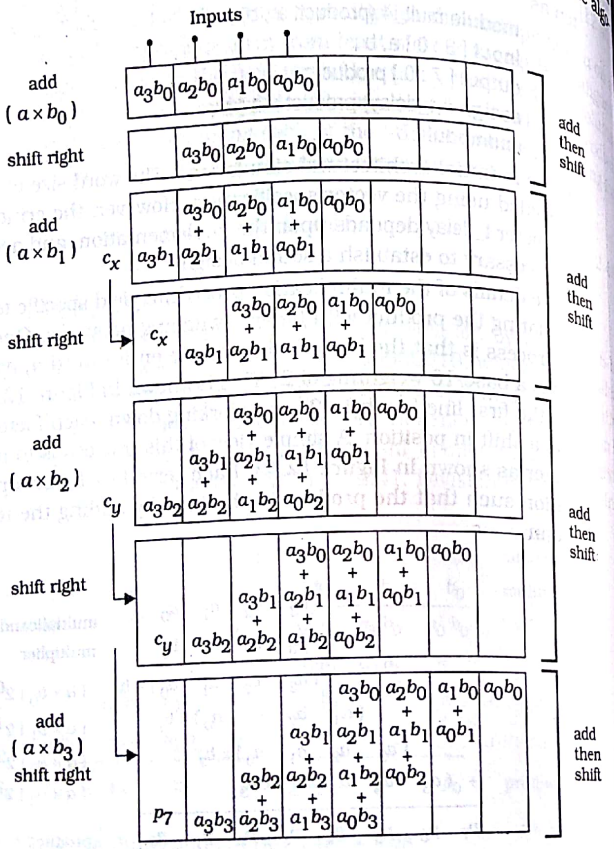


Figure 12.44 Shift-right multiplication sequence

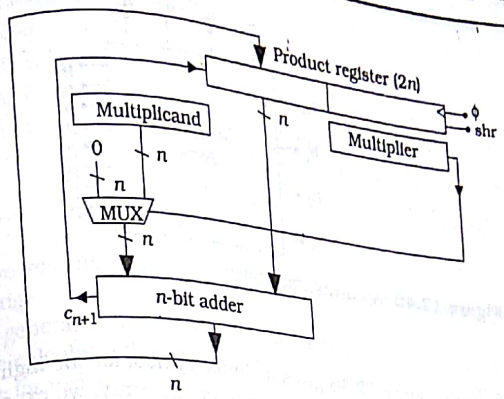


Figure 12.45 Register-based multiplier network

$$p_{i+1} = (p_i + a2^n b_i)2^{-1} \tag{12.50}$$

with $p_n = p$ the final answer such that $p_0 = 0$. The factor $(p_i + a\,b_i\,2^n)$ gives the addition while 2^{-1} accounts for a right shift. The factor of 2^n multiplying a is used to compensate for the 2^{-n} introduced by the right shift at the end of the calculation. The algorithm can be used to create the register-based hardwire multiplier network shown in Figure 12.45, which can be built using standard VLSI cells and sequential design. Note that the multiplier bits b_i are used to control a 2:1 multiplexor. If $b_i = 0$, an n-bit zero word is sent to the adder, while $b_i = 1$ directs the multiplicand a to the input. The Booth algorithm can be added to the network, as can several other improvements. Also, we note in passing that a left-shift algorithm can be derived, leading to a different hardware implementation.

This type of design can be coded in Verilog using **assign** statements and shift operators. Accurate delay times are needed to accurately simulate the system. These in turn depend upon the characteristics of the VLSI cells in the library. The complexity of multiplier units is reflected in the length of the HDL code and the time needed to design an efficient network.

12.5.1 Array Multipliers

An array multiplier accepts the multiplier and multiplicand and uses a array of cells to calculate the bit products $a_j \cdot b_k$ individually in a paral

[3] See Reference [2].

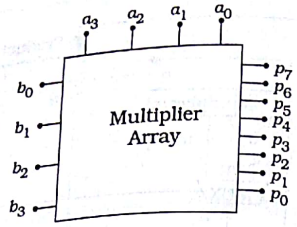


Figure 12.46 An array multiplier

manner. Figure 12.46 gives a simple symbol for the high-level view. To determine the properties needed for the array, we expand the view to show the structure of the multiplication procedure in Figure 12.47. Each block requires that we first calculate the bit product $a_j \cdot b_k$ and then add it to other contributions in column $i = (j + k)$. This produces the sum

$$p_i = \sum_{i=j+k} a_j b_k + c_{i-1} \tag{12.51}$$

for each product bit. An equivalent description of the operation is obtained by writing the base-10 values

$$A = \sum_{j=0}^{n-1} a_j 2^j \qquad B = \sum_{k=0}^{n-1} b_k 2^k \tag{12.52}$$

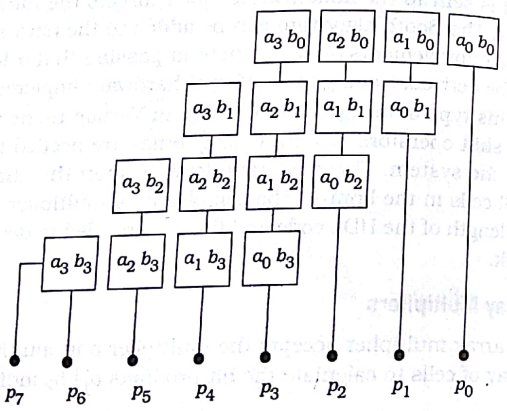


Figure 12.47 Modularized view of the multiplication sequence

and then forming the product

$$P = AB = \left(\sum_{j=0}^{n-1} a_j 2^j\right)\left(\sum_{k=0}^{n-1} b_k 2^k\right) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_j b_k 2^{j+k} \tag{12.53}$$

Then we see that the terms $a_j \cdot b_k$ provide the bit value and 2^{j+k} the weighting.

The general structure of a 4 × 4 array is shown in Figure 12.48. This scheme calculates the bit products $a_j \cdot b_k$ using AND gates. The product bits are formed using adders in each column. The adders are arranged in a carry-save chain as can be seen by noting that the carry-out bits are fed to the next available adder in the column to the left. The array multiplier accepts all of the input bits simultaneously. The longest delay in the calculation of the product bits depends on the speed of the adders. The

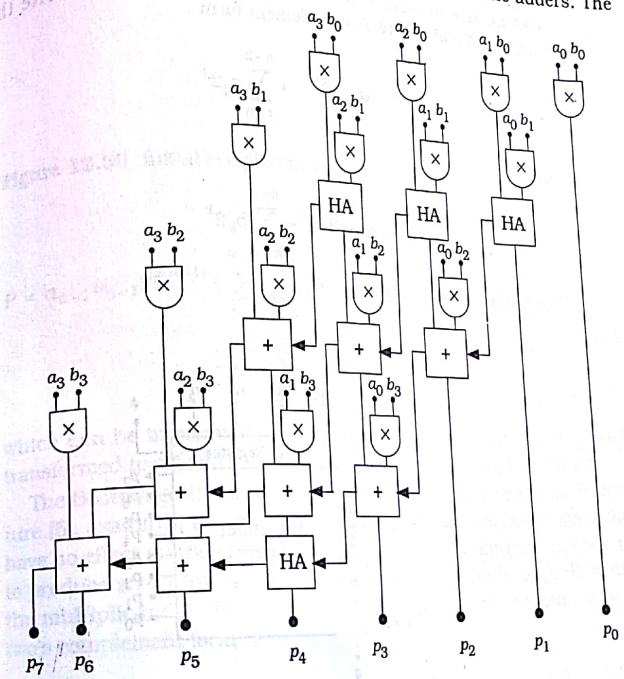


Figure 12.48 Details for a 4× 4 array multiplier

carry-chain in p_7 that originates from the carry bits from the p_1 column and propagates through the p_2 - p_6 quantities would be an obvious problem. Input registers may be added to synchronize the dataflow as shown in Figure 12.49. An output register may also be used if necessary. In general, an array multiplier for n-bit words requires n (n-2) full-adders, n half-adders, and n^2 AND gates. The gate count allows an estimate of the required area based on the library entries.

For layout purposes, it is useful to see if the cells can be arranged to give a more rectangular overall shape. An initial plan is obtained by using a regular interconnect pattern for the input bits, and then placing the units themselves in the order of the dataflow. The array structure starts to evolve as illustrated by the first-cut patterning in Figure 12.50. The actual placement can be adjusted to accommodate interconnect wiring and the different cell sizes.

12.5.2 Other Multipliers

Many multiplier algorithms and circuits have been published in the literature. The Baugh-Wooley multiplier is based on two's complement numbers for use in signed arithmetic. For this case, we write the input numbers A and B in two's complement form

$$A = -a_{n-1}2^{n-1} + \sum_{j=0}^{n-2} a_j 2^j \tag{12.54}$$

and

$$B = -b_{n-1}2^{n-1} + \sum_{k=0}^{n-2} b_k 2^k \tag{12.55}$$

The product is then given as

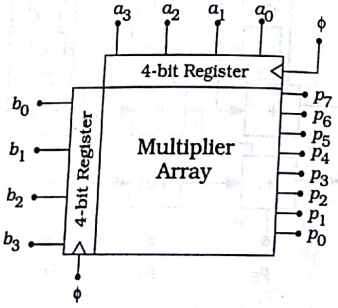


Figure 12.49 Clocked input registers

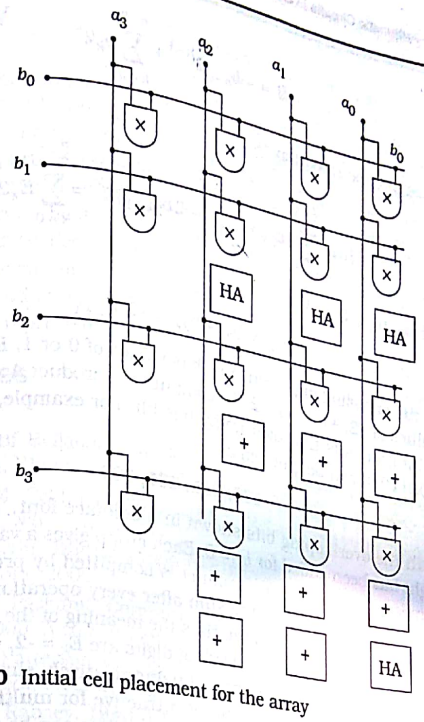


Figure 12.50 Initial cell placement for the array

$$p = a_{n-1}b_{n-1}2^{2(n-1)} + \sum_{j=0}^{n-2}\sum_{k=0}^{n-2} b_k a_j 2^{j+k} + -a_{n-1}\sum_{k=0}^{n-2} b_k 2^{k+n-1} - b_{n-1}\sum_{j=0}^{n-2} a_j 2^{j+n-1} \tag{12.56}$$

which can be implemented using adders and subtractors. This can be transformed to an adder-only scheme by using bit complements [1].

The Booth algorithm, familiar from studies in basic computer architecture [5], examines the multiplier word B and searches for 0's since these have no effect on the sum. This may be used to encode groups of bits in B to produce a control digit that specifies the operation to be performed on the multiplicand A. To see the basis of the technique, we start with the two's complement form

$$B = -b_{n-1}2^{n-1} + \sum_{k=0}^{n-2} b_k 2^k \tag{12.57}$$

This may be rewritten as

$$B = \sum_{k=0}^{\frac{n}{2}-1} [b_{2k} + b_{2k-1} - 2b_{2k+1}]2^k = \sum_{k=0}^{\frac{n}{2}-1} E_k 2^k \tag{12.58}$$

where $b_{-1} = 0$ and

$$E_k = b_{2k} + b_{2k-1} - 2b_{2k+1} \tag{12.59}$$

is the encoding digit. Since b_k has values of 0 or 1, E_k can have decimal values of +2, +1, 0, -1, -2. To compute the product $A \times B$, we divide B into 3-bit segments that overlap by one bit. For example, the 8-bit word B = 10011010 can be grouped as

$$\mathbf{100}, \mathbf{011}, \mathbf{101}, \mathbf{100} \tag{12.60}$$

with the overlapping bits shown in a boldface font. The last zero on the right has been added for $b_{-1} = 0$. Each group gives a value of E_k that determines an operation. The product is computed by providing a dual-word size register that holds the sum after every operation is completed. The table in Figure 12.51 summarizes the meaning of the encoded values. For the example shown, the encoding digits are E_k = -2, +2, -1, -2. The VLSI circuit can thus be constructed using relatively simple logic along with standard adder cells, making it attractive for multiplying large words. Another adder circuit called the Wallace tree can be used to improve the network by using carry-save adders for the sum.

b_{2k+1}	b_{2k}	b_{2k-1}	E_k	Effect on sum
0	0	0	0	add 0
0	0	1	+1	add *A*
0	1	0	+1	add *A*
0	1	1	+2	shift A left, add
1	0	0	-2	take two's (A), shift left, add
1	0	1	-1	add two's (*A*)
1	1	0	-1	add two's (*A*)
1	1	1	0	add 0

Figure 12.51 Summary of Booth encoded digit operations

12.6 Summary

Arithmetic circuits are created by using binary algorithms to suggest structures that fit well into the VLSI principles of regular layout, repetition of cells, and fast circuits. In this chapter we have examined some of the more important issues associated with issues of implementation. Only the basics have been presented. High-radix algorithms, floating-point numbers, and a host of other topics await the interested reader who is willing to pursue deeper studies.

Arithmetic circuits will continue to be of primary importance as microprocessors and other VLSI circuits evolve to even higher levels of performance. This represents a fascinating field for future research endeavors.

12.7 References

[1] Abdellatif Bellaouar and Mohamed I. Elmasry, **Low-Power Digital VLSI Design**, Kluwer Academic Publishers, Norwell, MA, 1995.

[2] James M. Feldman and Charles T. Retter, **Computer Architecture**, McGraw-Hill, New York, 1994.

[3] Ken Martin, **Digital Integrated Circuit Design**, Oxford University Press, New York, 2000.

[4] Behrooz Parhami, **Computer Arithmetic**, Oxford University Press, New York, 2000. A comprehensive, in-depth treatment of the subject.

[5] David A. Patterson and John L. Hennessy, **Computer Organization & Design**, 2nd ed., Morgan-Kaufmann Publishers, San Francisco, 1998.

[6] Jan M. Rabaey, **Digital Integrated Circuits**, Prentice Hall, Upper Saddle River, NJ, 1996.

[7] Bruce Shriver and Bennett Smith, **The Anatomy of a High-Performance Microprocessor**, IEEE Computer Society Press, Los Alamitos, CA, 1998.

[8] William Stallings, **Computer Organization and Architecture**, 4th ed., Prentice Hall, Upper Saddle River, NJ, 1996.

[9] John P. Uyemura, **CMOS Logic Circuit Design**, Kluwer Academic Publishers, Norwell, MA, 1999.

[10] Neil H.E. Weste and Kamran Eshraghian, **Principles of CMOS VLSI Design**, 2nd ed., Addison-Wesley, Reading, MA, 1993.

[11] Wayne Wolf, **Modern VLSI Design**, 2nd ed., Prentice Hall PTR, Upper Saddle River, NJ, 1998.

12.8 Problems

[12.1] Design a half-adder that has inputs *a* and *b* using pseudo-nMOS. Then construct the gate-level Verilog description using **nmos** and other primitives that are needed.

Memories and Programmable Logic 13

Memories are indispensable in modern digital systems. They provide for short- and long-term storage of binary variables and words. The VLSI aspects of CMOS memories are interesting because they are designed using a cell library and exhibit repetitive layout geometries. This chapter discusses the design of semiconductor memory arrays and concludes with an introduction to more general programmable logic structures.

13.1 The Static RAM

The acronym **RAM** stands for **random-access memory**, and implies a memory array that allows access to any bit (or group of bits) as needed. In practice, however, the meaning of "RAM" has evolved to imply a memory with both read and write capabilities to distinguish it from a read-only memory (ROM) array.

Static random-access memory (**SRAM**) cells use a simple bistable circuit to hold a data bit. A static RAM cell can hold the stored data bit so long as the power is applied to the circuit. SRAMs have three operational modes. When the cell is in a **hold** state, the value of the bit is stored in the cell for future usage. During a **write** operation, a logic 0 or 1 is fed to the cell for storage. The value of the stored bit is transmitted to the outside world during a **read** operation.

Figure 13.1 shows the general circuit scheme. A pair of cross-coupled inverters provides the storage, while two **access transistors** MAL and MAR provide read and write operations. The access transistors are controlled by the **word line** signal *WL* that defines the operational modes. When $WL = 0$, both access FETs are off and the cell is isolated. This defines the hold condition. To perform a read or write operation, the word line is brought up to a value of $WL = 1$. This turns on the access transis-

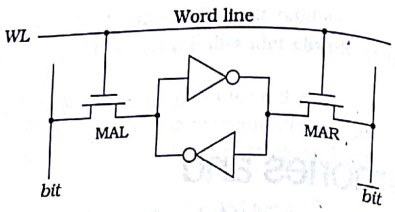
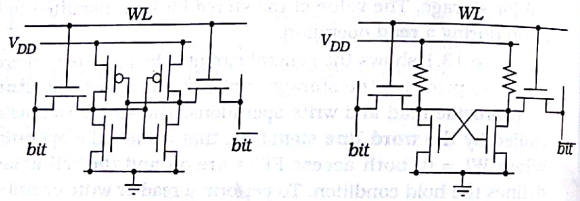


Figure 13.1 General SRAM cell

tors connecting the dual-rail data lines *bit* and $\overline{bit}$ to the outside circuitry; these are often called the **bit** and **bit-bar** lines, respectively. A write operation is performed by placing voltages on the *bit* and $\overline{bit}$ lines, which then act as inputs. Dual-rail logic helps increase the writing speed. For a read operation, the bit and bit-bar lines act as outputs and are fed into a **sense amplifier** that determines the stored state. The distinction between read and write operations is obtained by circuitry outside the cell array.

Two types of CMOS cells are dominant in practice. The circuit in Figure 13.2(a) is called the 6-transistor (6T) design and uses standard CMOS inverters. The 4-transistor (4T) uses resistors as load devices in an nMOS circuit as in Figure 13.2(b). The resistors are made using an undoped poly layer that resides above the silicon (transistor) level. This can yield a smaller cell area and allow higher packing density, but requires that an additional polysilicon layer and masking step be added to the process. The electrical characteristics of the two are quite different since the 4T cell uses a very large (typically greater than about 1 GΩ) passive pull-up resistor. We will concentrate on the dominant 6T design here.

The basic circuit level design issues revolve around choosing the values of the transistor aspect ratios to insure that the cell can hold a state while still allowing it to be changed during a write operation without excessive delay. Figure 13.3 shows the main parameters. A symmetric design is assumed such that β_A is the device transconductance of both



(a) 6T cell (b) 4T cell with poly resistors

Figure 13.2 CMOS SRAM circuits

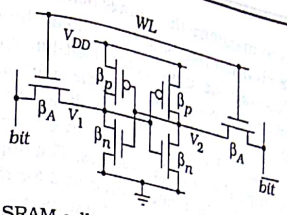


Figure 13.3 6T SRAM cell design parameters

access FETs, while the storage cell itself uses nFETs and pFETs with sizes described by β_n and β_p, respectively.

Stability of the hold state depends upon the functionality of the cross-coupled inverter cell. The inverter ratio (β_n/β_p) establishes the midpoint voltage V_M of each NOT gate, which in turn sets the characteristics of the feedback. This is usually described by a curve known as a **butterfly plot** that is obtained by forcing an input on one of the internal nodes and plotting the response on the other side, then performing the same operation to the other side. The superposed plots give the butterfly shape as in Figure 13.4. The **static noise margin**, labeled as SNM in the drawing, is the separation between the curves along a 45° slope and has units of volts. Its value indicates the level of immunity that the cell has to unwanted voltage changes due to coupled electromagnetic signals that are collectively called **noise**. A reasonable noise margin is needed for robust storage. The 6T cell design gives higher SNM values than the resistor-load 4T design, making it more attractive in noisy high-density environments.[1] Although the values of β_n and β_p can be adjusted to create different butterfly characteristics, the storage FETs are commonly chosen to have the smallest possible

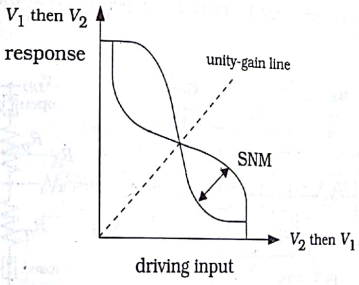


Figure 13.4 Butterfly plot

[1] The problem of electrical noise is discussed in Chapter 14 in the context of interconnect analysis.

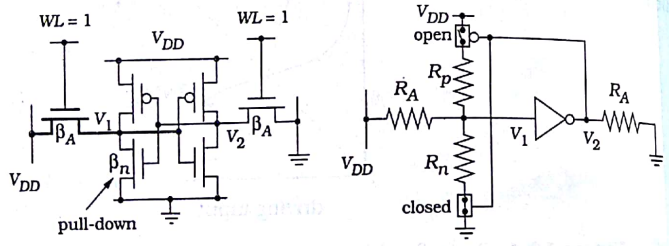
aspect ratios to maximize the storage density of an SRAM array.

The write characteristics of the cell can be understood from Figure 13.5(a). In this case, we apply a logic 1 bit-line voltage of V_{DD} to the left-bit line that feeds the access FET, while the right side (bit-bar line) is simultaneously placed at 0 V for a logic 0 voltage. The worst-case condition is where initially $V_1 = 0$ V and $V_2 = V_{DD}$ since both the bit and bit-bar voltages must change the internal voltages. The important design parameter is (β_A / β_n) with published values around 2 for the 6T cell. The reasoning behind this statement can be seen from the resistor model of the circuit shown in Figure 13.5(b). The input voltage V_{DD} is responsible for increasing V_1 to a logic 1 level. However, the nFET switch (at the bottom of R_n) is closed and pulls V_1 to 0 V and the feedback loop with the other inverter tries to hold this value. Selecting $\beta_A > \beta_n$ implies that $R_A < R_n$, which allows the access FET to be more effective in increasing V_1 to the level needed to switch the stored state. If cell area is the overriding factor, then (β_A / β_n) may be chosen to have a value closer to 1. Note that since both FETs are *n*-channel devices, the design ratio reduces to the ratio of aspect ratios

$$\frac{(W/L)_{nA}}{(W/L)_n} \tag{13.1}$$

in the layout; in the literature, this is sometimes called simply *the* β-ratio.

SRAM cell layout is driven by the desire to simultaneously minimize the cell area while providing port locations that will allow for high-density arrays. Figure 13.6 shows an approach to cell design that uses perpendicular lines in Metal1 and Metal2 to form the power supplies (VDD and VSS) and the bit, bit_bar lines. The storage cell is contained in the central part of the cell. The n+ regions of nFETs are extended beyond the inverter circuits to form the access transistors with the word lines running vertically. Allowing for 45° turns in the poly lines would help reduce the area.



(a) Write 1 operation (b) Resistor model

Figure 13.5 Writing to an SRAM

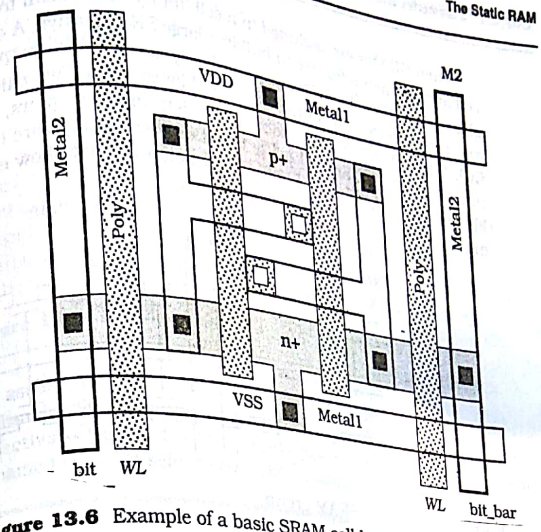


Figure 13.6 Example of a basic SRAM cell layout

Multiple-port SRAM cells provide cell access to more than one pair of bit/bit-bar lines. A 2-port cell is shown in Figure 13.7. The word line *WL_1* controls the read/write operations for the *bit_1* lines, while *WL_2* provides the same control for the *bit_2* lines. Additional logic must be added to avoid conflicts between the two ports. Multiport memories can simplify system wiring and layout, since different logic sections can share a memory block. At the system level, however, a method for tracking the contents of the memory and a priority access scheme must be developed to insure correct operation.

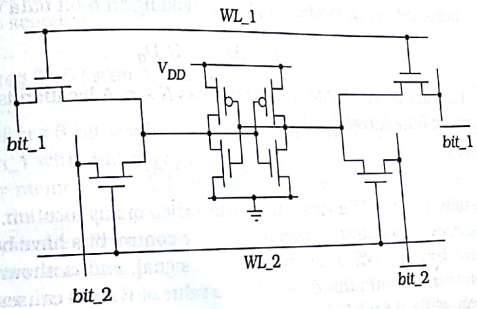


Figure 13.7 A 2-port CMOS SRAM cell

When SRAMs are included in a cell library, it is useful to create multi-cell arrangements for use in building large SRAM arrays. A 4-cell group is shown in Figure 13.8. The two word lines are denotes by RW0 and RW1, and respectively control the upper and lower pairs. Two pairs of bit lines (X0, Y0) and (X1, Y1) are used for the left and right pairs, respectively; note that X0 = $\overline{Y0}$ and X1 = $\overline{Y1}$. When multicell groups are included as a library entity there are usually support circuits that allow easy interfacing.

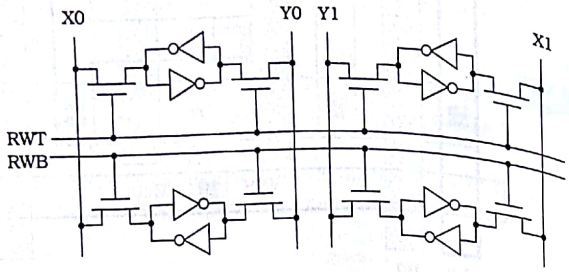


Figure 13.8 4-cell SRAM group

13.2 SRAM Arrays

Static RAM arrays are created by replicating the basic storage cell and adding the necessary peripheral circuitry. The objective is to obtain the highest storage density for a given cell layout; short access times are also important in the majority of applications.

The design of a complete SRAM provides an interesting and useful study of design hierarchies. Figure 13.9 shows the highest level view of a functional SRAM unit. At this level, an SRAM consists of N storage locations with each location capable of holding an n-bit data word

$$D_{n-1}D_{n-2}\ldots D_1D_0 \tag{13.2}$$

The size of the SRAM is designated as $N \times n$. A location is specified using an m-bit address word

$$A_{m-1}A_{m-2}\ldots A_1A_0 \tag{13.3}$$

such that $N = 2^m$ allows a unique selection of any location. This is used to specify read and write operations. Two control bits have been included in the drawing. WE is the **write-enable** signal, and is shown as an assert-low control; with this designation, a value of $WE = 0$ causes a write operation while $WE = 1$ indicates a read. The entire unit is under the control of

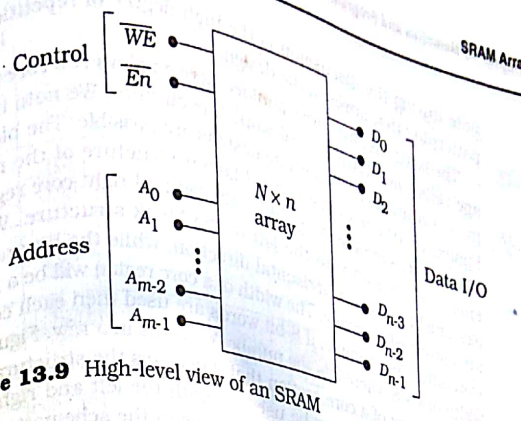


Figure 13.9 High-level view of an SRAM

the assert-low enable signal En. When $En = 1$, the read and write circuitry is disabled and the memory is in a hold state. A value of $En = 0$ is needed to activate the read/write operations. At the chip level, En would be renamed the **chip select** CS or **chip enable** CE.

Example 13.1

A 128K × 8 SRAM chip holds 128K 8-bit words for a total of 1 Mb of total storage. The address word must have a width of

$$m = \log_2(128K) = 17 \tag{13.4}$$

to select every 8-bit word location.

Verilog does not provide primitives for 2-dimensional memories. However, the **reg** data type can be used to write statements that describe SRAMs at the system level. An example is the 2KB storage unit sram_1 in the code segment

```
...
reg [7 : 0 ] sram_1 [0 : 2047]
...
```

This defines 8-bit words (i.e., 1 byte) using **reg** [7 : 0] that are identified by sram_1 with addresses from 0 to 2047. This can be modified for any word or memory size. The simplicity of the Verilog high-level description masks the complexity of the internal structure of the unit sram_1. To see the physical implementation of the memory, we will start with an architectural view and then progress downward and study some of the circuitry. This can be used to write Verilog models at lower hierarchical levels, which are useful in the verification of the architecture. One key point to

note during the discussion is the high degree of repetition and regula patterning that arises in the design.

The basic architecture examined here employs two core regions of stor age cells that share central word-line circuits. We note in passing that many variations and alternate designs are possible. The block diagram in Figure 13.10 shows the central layout structure of the memory array. Memory cells are tiled to produce the left and right core regions shown. A single cell is shown to the left of the block structure. Word lines are assumed to run in a horizontal direction, while the bit and bit-bar lines are patterned vertically. The width of a core region will be a multiple of the word size. For example, if 8-bit words are used then each core will have a width of $k \times 8$ where k is the number of words in a row. Figure 13.11 is an enlarged view of a core section that illustrates the structural detail of the cells. The structure can be used for both the left and right cores of the present example. The regular patterning in the schematic is maintained at the physical silicon level. Cell layout is based on finding a pattern that allows for proper placement and wiring with a high packing density.

The outputs of a centrally located active-high row decoder provide the word line signals to the storage cells. The address word specifies a particular row, which is then driven high. The access transistors of the selected row cells are turned on, permitting the read/write operations to take place. The location of the circuitry allows a single decoder to be used for both the left and right memory cores. A library-based static decoder circuit can be instanced directly into the design. The row decoder outputs are fed into row driver circuits that are used to drive the word lines of the arrays. Drivers are needed because of the large capacitive load presented by the long interconnects and the access transistors connected to each

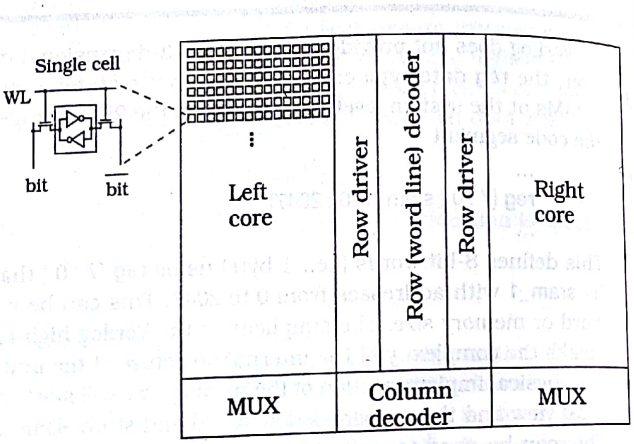


Figure 13.10 Central SRAM block architecture

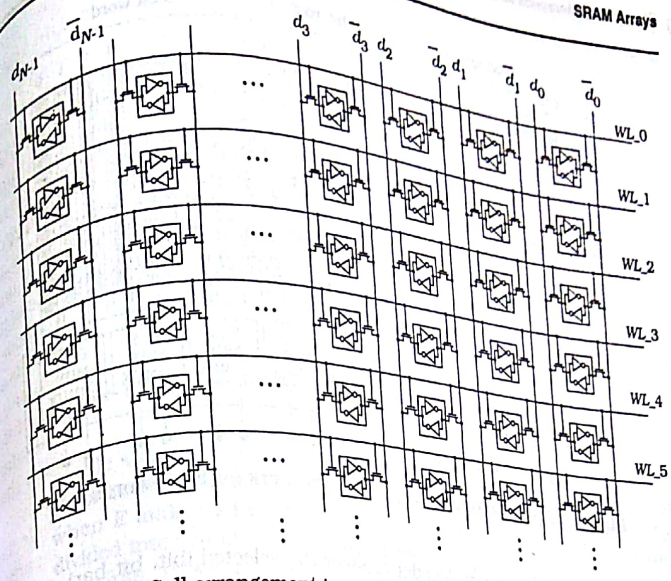


Figure 13.11 Cell arrangement in a core region

word line. A basic row driver design is shown in Figure 13.12. The output from the decoder is designated as Dec_out. The first pFET acts as a pull-up device, while the second pFET reinforces the output using feedback around the NAND2 gate. A sized inverter chain is used to provide the drive capability for the word line.

It is seen from the array in Figure 13.11 that the input/output bit and bit-bar data lines of the cells form the columns of the memory matrix. The data flow is thus visualized to be vertical for both read and write operations. Once a word line is driven high by the row decoder, every cell in the row is accessible. To choose a particular k-bit word in the row, we must add the group of column decoder circuits that select a particular set of columns in the matrix. The MUX sections shown in Figure 13.10 are c

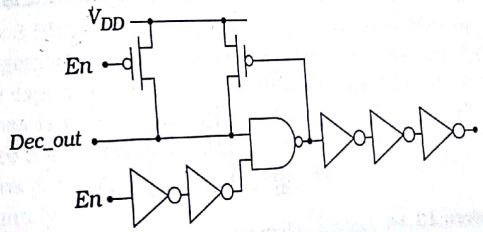


Figure 13.12 Row driver circuit

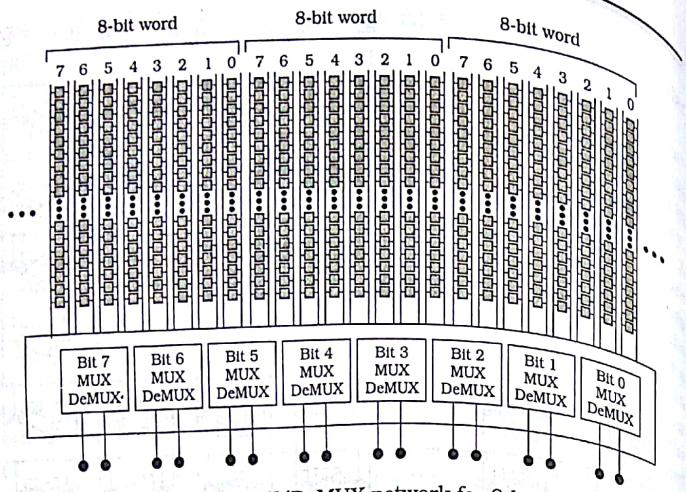


Figure 13.13 Column MUX/DeMUX network for 8-bit words

trolled by the column decoder to steer the selected (bit, bit-bar) groups. The overall structure of the column-select network is shown in Figure 13.13 for the case of 8-bit words. Each MUX/DeMUX block is connected to the appropriate data line of each word; for example, the bit_0 and bit-bar_0 lines of every word are wired to the Bit 0 MUX/DeMUX block. A read operation requires an output from the cells, so the circuits act as multiplexors. For a write operation, the DeMUX mode must be used to steer a data word into the proper columns. Column drivers are used to feed the MUXes; a simple feedback-oriented logic 1 driver design is shown in Figure 13.14. When In =1, the output is a 0 which turns on the pull-up pFET; the pFET is wired to help maintain the high input voltage.

To clarify the addressing scheme, let us examine the simplest case where the m-bit address word $A = A_{m-1} \ldots A_2 A_1 A_0$ is divided into row and column groups with x rows and y columns such that $x + y = m$. In the block diagram of Figure 13.15, the address is fed into an address latching register that allows it to stabilize. An address latching circuit is shown in Figure 13.16. It consists of a basic D-type latch that is controlled by the

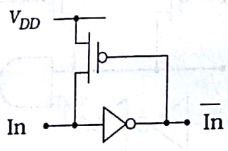


Figure 13.14 Logic 1 column driver

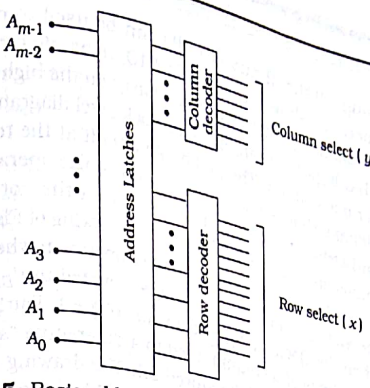


Figure 13.15 Basic addressing scheme

enable signal E that is derived from En and other control signals to synchronize the system. This circuit latches onto an input address bit Add_in when E makes a transition from 0 to 1. The outputs A and $\overline{A}$ are then divided into column and row segments and used as inputs to the decoder networks. This allows us access to any group of words in the cell matrix.

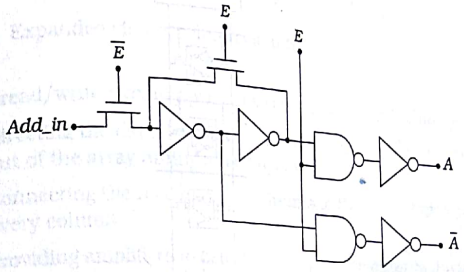


Figure 13.16 Address latch circuit

Example 13.2

The 128K × 8 SRAM chip requires a 17-bit address word. If we use a dual core arrangement with one word per word line, then we need 64K word lines. If we expand each word line to 64 bits (= 8 words) then the number of word lines is reduced to 8K. The 17-bit address $A_{16} \ldots A_0$ can thus be divided into a 4-bit column address group of $A_{16}A_{15}A_{14}A_{13}$ and a 13-bit row address group of $A_{12} \ldots A_0$. Other array sizes divide the address word proportionately.

Although static library circuits can be used to construct the entire SRAM network shown in Figure 13.10, dynamic circuits provide faster read operations by employing a precharge on the high-capacitance bit and bit-bar I/O (input/output) lines. A block-level diagram is shown in Figure 13.17 for one column. The precharge circuit at the top is controlled by a clock signal ϕ that is used to synchronize the operation and data flow. Read and write operations are indicated at the bottom of the column. More details are shown in the expanded drawing of Figure 13.18. The precharge circuits are active during a read operation when $\phi = 0$; during this time, the voltage on every data line is elevated to V_{DD}. Evaluation takes place when the clock changes to a value of $\phi = 1$. During this time, the bit and bit-bar lines of a column a fed to a differential "sense" amplifier that determines the value of the stored bit. The drawing also shows the column MUX circuits. Each word is selected by a control signal; Col_0 is used as an example in the figure. When *Col*_0 = 1, the nFETs are active and the entire group of bit and bit-bar lines are connected to the read/write circuit blocks. A separate column select signal is used for every word group.

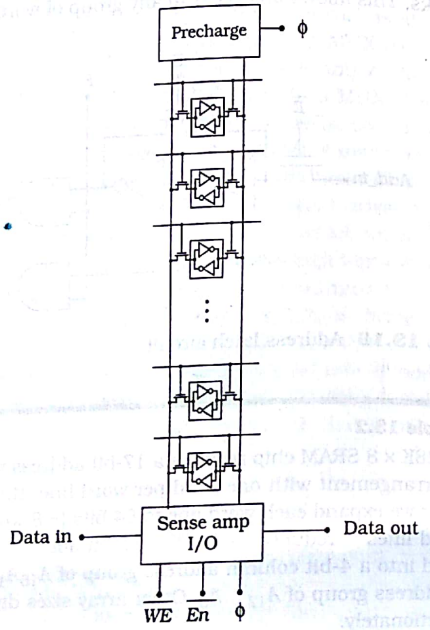


Figure 13.17 Precharge and I/O circuits for a single column

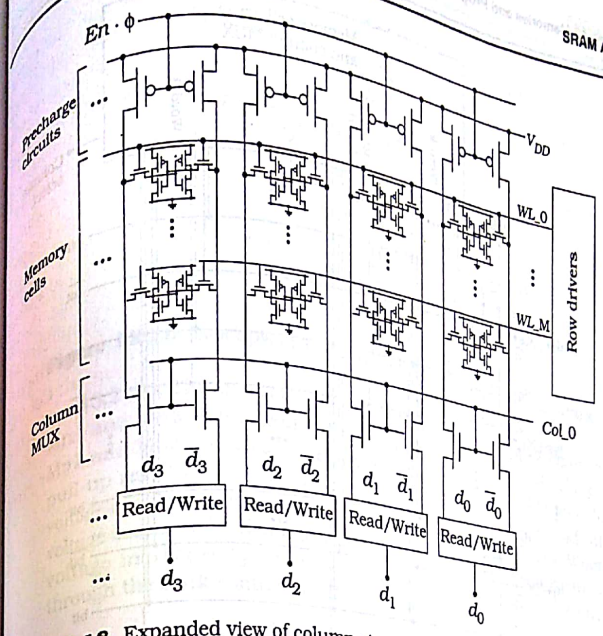


Figure 13.18 Expanded view of column circuitry

The read/write circuitry performs several functions including

- Directing the data flow into the array during a write operation, or out of the array during a read operation.
- Connecting the read and write circuits to the bit and bit-bar lines of every column.
- Providing amplifiers to detect and amplify the outputs during a read operation.

An example of the write circuitry is shown in Figure 13.19 for an 8-bit word design. The input bits $d_7, d_6, ..., d_1, d_0$ are inverted and buffered to provide complementary pairs (d_i, $\overline{d}_i$). When the write enable control bit has a value *WE* =1, the nFETs act as closed switches connecting the data pairs to the bit and bit-bar columns. As shown in the schematic, every bit pair is fed to the appropriate locations that define the 8-bit word column groups. The column multiplexor circuits (not shown explicitly in the drawing) determine which column receives the input word.

Additional circuitry is required to detect the stored bit values during a read operation. The block-level circuit for one bit shown in Figure 13.20 is based on the use of differential amplifiers (denoted by triangular symbols) with + and – inputs. An identical circuit is required for each output bit. A

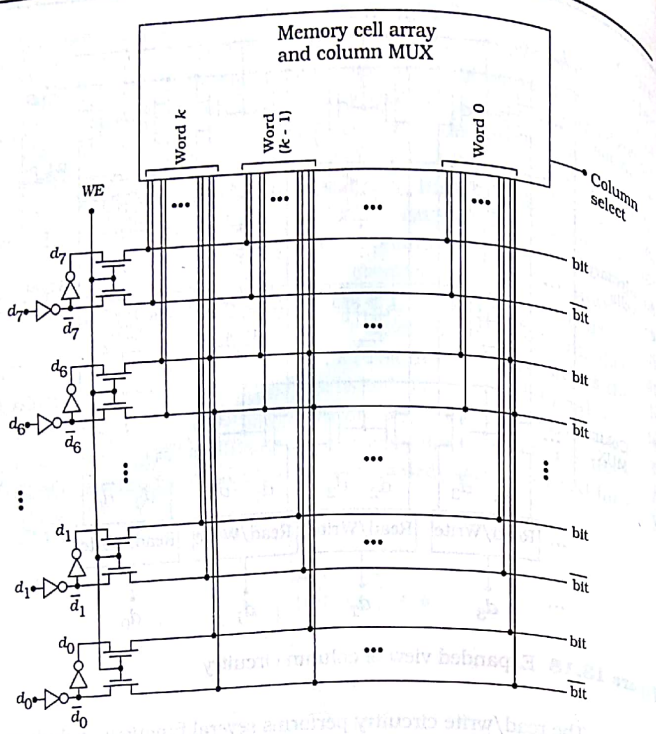


Figure 13.19 Write circuitry example

differential amplifier produces an output that depends upon the difference voltage

$$v_d = (v^+ - v^-) \tag{13.5}$$

between the input voltages v^+ and v^-. The output voltage of the amp is

$$v_{out} = Av_d = A(v^+ - v^-) \tag{13.6}$$

where $A > 1$ is the voltage gain of the amplifier. When used in an SRAM, the inputs are the bit and bit-bar signals from the storage cells. The circuit shown in the drawing uses a two-level sensing scheme. The first level consists of a pair of differential amplifiers that are fed oppositely phased inputs. The outputs are then combined to a single differential amplifier that outputs the result to a data latch. To make these compatible with the dynamic column precharge circuitry, the sense amps themselves are controlled by the clock signal ϕ.

The transistor-level details for a differential amplifier are shown in Figure 13.21(a). This is a standard design that is based on two input nFETs Mn1 and Mn2 that accept complementary inputs d and $\overline{d}$. The pFETs Mp1 and Mp2 are used as **active load** devices and act like non-linear pull-up resistors. The difference signal $(d\text{-}\overline{d})$ due to the bit and bit-bar voltages controls the currents I_{D1} and I_{D2} flowing in the nFETs. When the voltage associated with d is large, I_{D1} increases; similarly, increasing the $\overline{d}$ voltage increases I_{D2}. The total currents are limited by the current through the clock-controlled nFET Mn such that

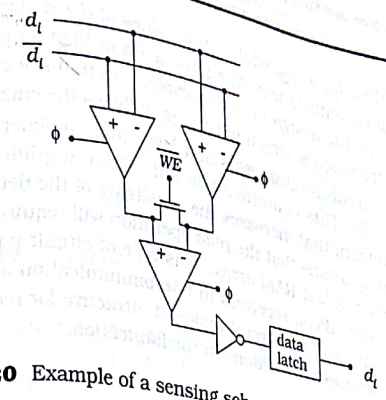
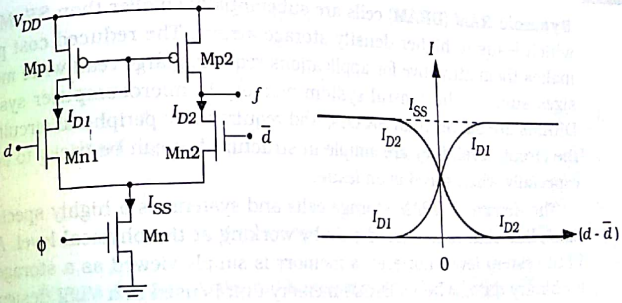


Figure 13.20 Example of a sensing scheme for the read operation

$$I_{SS} = I_{D1} + I_{D2} \tag{13.7}$$

which is valid when $\phi = 1$ in this design. Analyzing the circuit yields the current flow characteristics illustrated in Figure 13.21(b), which portrays the currents as a function of $(d - \overline{d})$.

When the inputs are the same with $d = \overline{d}$, $(d - \overline{d}) = 0$ and $I_{D1} = I_{D2}$. During an SRAM read operation, one voltage will be higher than the other. If



(a) Circuit diagram (b) Current flow

Figure 13.21 Single-ended differential amplifier

$d > \overline{d}$ then $I_{D1} > I_{D2}$, while input values of $d < \overline{d}$ give $I_{D2} > I_{D1}$. The difference in currents is translated into a low or high output voltage. At the circuit level, the design problem revolves around selecting the aspect ratios of the transistors, which in turn establishes the small signal gain.

The circuit diagram for a first-level dual-amplifier pair is shown in Figure 13.22. This combines two individual amplifiers in a cross-driven arrangement that increases the sensitivity of the detection circuit. A high sensitivity means that the read operation will require less time, leading to the idea of a fast RAM array. This type of circuit has also been used for high-speed silicon receivers in telecommunication applications. The balanced nature of the circuit makes it attractive for reducing noise and the effects of process variations in the fabrication.

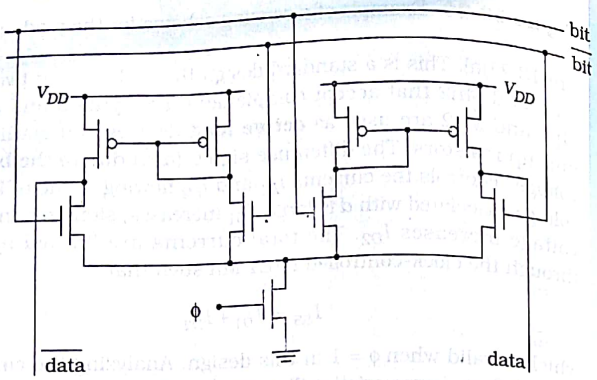


Figure 13.22 Dual-amplifier scheme for the sense amplifier network

13.3 Dynamic RAMs

Dynamic RAM (DRAM) cells are substantially smaller than SRAM cells, which leads to higher density storage arrays. The reduced cost per bit makes them attractive for applications requiring large read-write memory sizes such as the central system memory in microcomputer systems. DRAMs are slower than SRAMs, and require more peripheral circuitry. At the circuit level, they are simple in structure but can be tricky to design, especially when speed is an issue.

The design of DRAM storage cells and systems is a highly specialized discipline that is mastered only by working at the physical level. At the VLSI system level, however, a memory is simply viewed as a storage unit for binary data. When a DRAM memory unit is used in a VLSI design, it is usually instanced from a library entry that has been designed by a specialized group.[2] Owing to this observation, our discussion of DRAMs will

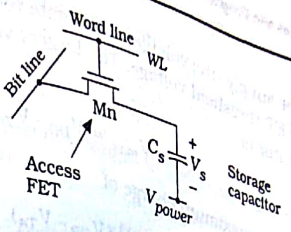


Figure 13.23 1T DRAM cell

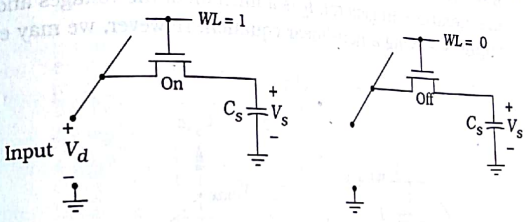
be limited to understanding the basics to see the operation and trade-offs.

A 1-transistor (1T) DRAM cell is shown in Figure 13.23. It consists of a single access nFET Mn and a storage capacitor C_s. The cell is controlled by the word line signal WL and a single bit line provides the I/O path to the cell. The bottom of the capacitor is connected to one of the power supply rails, and is denoted as V_{power} in the drawing; either V_{DD} or V_{SS} may be used. The storage mechanism is based on the concept of temporary charge retention on the capacitor. A voltage V_s across the capacitor corresponds to a stored charge Q_s of

$$Q_s = C_s V_s \tag{13.8}$$

With $V_s = 0$ V, $Q_s = 0$ and the charge state is a logic 0. Conversely, a large value of V_s gives a large Q_s, which is defined to be a logic 1 charge state.

The write operation is shown in Figure 13.24(a) for the case where $V_{power} = V_{SS} = 0$ V. Applying V_{DD} to the nFET gate turns on the access transistor and allows access to the storage capacitor. The input data voltage V_d controls the current to/from C_s. A logic 0 data voltage $V_d = 0$ V results in a voltage $V_s = 0$ V across the capacitor, corresponding to a charge state of $Q_s = 0$. If we apply a logic 1 data voltage $V_d = V_{DD}$ equal to



(a) Write operation (b) Hold

Figure 13.24 Write and hold operations in a DRAM cell

[2] SRAMs are often viewed in the same manner.

the power supply, the voltage on the gate reduces the transmitted signal by an nFET threshold voltage. The largest voltage that can be passed to the capacitor is

$$V_s = V_{max} = V_{DD} - V_{Tn} \tag{13.9}$$

which gives a maximum charge of

$$Q_{max} = C_s(V_{DD} - V_{Tn}) \tag{13.10}$$

The hold state is achieved by turning off the access transistor with a word line signal of $WL = 0$. This is shown in Figure 13.24(b).

The dynamic aspect of the cell arises during a data hold time. As discussed in Chapter 9, a MOSFET that is biased into cutoff with $V_G < V_T$ still admits small leakage currents. The DRAM circuit problem is illustrated in Figure 13.25. A logic 1 voltage $V_s = V_{max}$ on the storage capacitors provides the electromotive force for the leakage current I_L flowing away from C_s. This can be described by

$$I_L = -\left(\frac{dQ_s}{dt}\right) \tag{13.11}$$

which shows that the current removes charge from the capacitor. Using equation (13.8) for Q_s gives the capacitor relation

$$I_L = -C_s\left(\frac{dV_s}{dt}\right) \tag{13.12}$$

so that V_s also drops. Assuming an initial voltage of $V_s = V_{max}$ gives the voltage decay illustrated in Figure 13.25. The minimum logic 1 voltage is denoted as V_1 in the drawing. The hold time t_h is defined as the longest period of time that the cell can maintain a voltage large enough to be interpreted as a logic 1; the hold time is also called the **retention time** in the literature. In general, I_L is a function of the voltages and finding $V_s(t)$ requires solving a non-linear equation. However, we may estimate by t_h

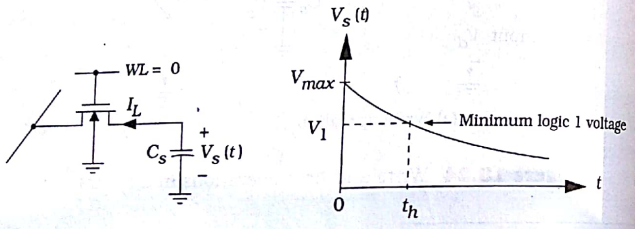


Figure 13.25 Charge leakage in a DRAM cell

assuming that I_L is a constant and writing

$$I_L \approx -C_s\left(\frac{\Delta V_s}{\Delta t}\right) \tag{13.13}$$

where ΔV_s and Δt represent changes in the variables. Rearranging gives the hold time equation

$$t_h = |\Delta t| \approx \left(\frac{C_s}{I_L}\right)(\Delta V_s) \tag{13.14}$$

as a first estimate. This shows that the hold time may be increased by using a large capacitance and minimizing the leakage current. As an example, if $I_L = 1$ nA, $C_s = 50$ fF, and $(\Delta V_s) = 1$ V, the hold time is

$$t_h = \left(\frac{50\times10^{-15}}{1\times10^{-9}}\right)(1) = 0.5\mu s \tag{13.15}$$

This illustrates the short hold time of a DRAM cell, and clearly justifies the use of the adjective "dynamic" for the circuit.

Memory units must be able to hold data so long as the power is applied. To overcome the charge leakage problem, DRAM arrays employ a **refresh operation** where the data is periodically read from every cell, amplified, and then rewritten. The procedure is listed in Figure 13.26. The cycle must be performed on every cell in the array with a minimum refresh frequency of about

$$f_{refresh} \approx \frac{1}{2t_h} \tag{13.16}$$

Refresh circuitry is included in the overhead logic that surrounds the cell array. The refresh cycle is designed to operate in the background and is therefore transparent to the user.

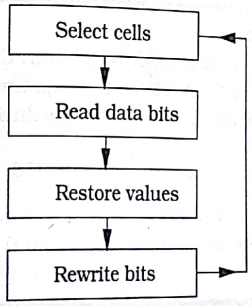


Figure 13.26 Refresh operation summary

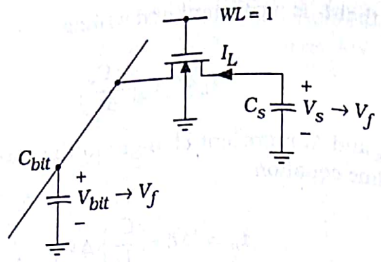


Figure 13.27 Read operation in a DRAM cell

A read operation is shown in Figure 13.27. The voltage V_s on the capacitor at the read time provides the voltage to move charge from C_s to the bit line capacitance C_{bit}, which sets up a charge sharing situation. C_{bit} includes the line capacitance and other parasitic contributions such as the input capacitance of the sense amplifier. The initial charge on the capacitor is

$$Q_s = C_s V_s \tag{13.17}$$

where $V_s = 0$ V for a logic 0, and $V_s > 0$ for a logic 1. Current flow from C_s to C_{bit} continues until the voltages are equal to the final voltage $V_f = V_{bit} = V_s$. The charge is redistributed according to

$$Q_s = C_s V_f + C_{bit} V_f \tag{13.18}$$

The initial and final values of Q_s must be equal by charge conservation, so

$$V_f = \left(\frac{C_s}{C_s + C_{bit}}\right) V_s \tag{13.19}$$

This shows that $V_f < V_s$ for a stored logic 1. In practice, V_f is usually reduced to a few tenths of a volt, so that the design of the sense amplifier becomes a critical factor.

Example 13.3

Suppose that we have a DRAM cell with C_s = 50 fF and a bit line capacitance of $C_{bit} = 8\ C_s$. Assuming a maximum voltage of $V_s = V_{max} = 2.5$ V on the storage capacitor, the final voltage during a logic 1 read operation is

$$V_f = \left(\frac{1}{9}\right)(2.5) = 278 \text{ mV} \tag{13.20}$$

A stored logic 0 would result in $V_f = 0$ V, so that the sense amplifier must be able to distinguish between 0 V and 0.28 V to determine the value of the stored bit.

Modern DRAM chips have surpassed the 1 Gb density by using novel capacitor structures that are possible with advanced semiconductor processing techniques. The 1T storage cell consists of a single transistor and a storage capacitor. High-density arrays are created by reducing the individual cell area A_{cell} to the smallest size possible. Peripheral circuits for addressing, refresh, and other operations must be added to make the chip functional and can easily consume more than 30% of the total chip area.

In standard MOS processing, the nFET must reside on the silicon wafer; since submicron line widths are standard, the FET area is relatively small. Decreasing the overall cell area usually revolves around the design of the storage capacitor. The value of C_s must be about 40 fF or larger. Using the parallel-plate capacitor formula indicates that we need a plate area A_p of

$$A_p = C_s\left(\frac{t_{ins}}{\varepsilon_{ins}}\right) \tag{13.21}$$

where t_{ins} and ε_{ins} are the insulator thickness and permittivity, respectively. Assuming a silicon dioxide layer that is 50 Å thick implies a plate area of

$$A_p = (40\times10^{-15})\left(\frac{50\times10^{-8}}{3.45\times10^{-13}}\right) = 5.8\times10^{-8} \text{ cm}^2 \tag{13.22}$$

which is 5.8 μm^2. This is much larger than can be used for large arrays. For example, a 64Mb DRAM usually requires a cell size of about 1.25 μm^2 to meet chip requirements. Much research has been devoted toward building storage capacitors that increase the plate area without increasing the cell surface area A_{cell} (also called the footprint size). There are two main structures in use: trench capacitors and stacked capacitors.

A storage cell that uses a trench capacitor is shown in Figure 13.28. The capacitor is created by using a reactive ion etch (RIE) process to create a deep trench in the silicon. The sides are oxidized to create a glass

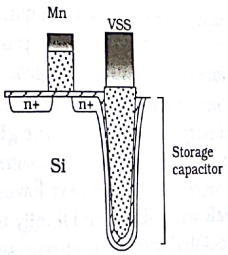


Figure 13.28 A DRAM cell using a trench capacitor

insulator, and then doped polysilicon is used to fill the trench and acts as the upper plate. The lower plate is created by an n+ implant along the entire wall area. The increase in plate area A_p is achieved by using the sidewalls of the trench without increasing the footprint area A_{cell}.

A stacked capacitor design places a polysilicon structure on top of the access transistor as portrayed schematically in Figure 13.29. Advanced 3-dimensional structures can be created to form the upper and lower plates. The plate area A_p depends upon the surface geometry created by the poly plates. Many interesting stacked capacitor designs have been published in the literature. In addition, surface corrugations and "bumps" have been added to further increase the value of C_s. The interested reader is directed to reference [8] for an overview of the subject.

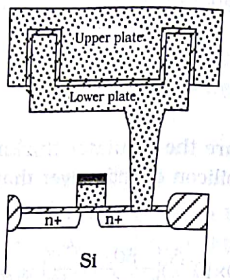


Figure 13.29 Visualization of stacked capacitor structure

13.3.2 Divided-Word Line Architectures

As the capacity of RAMs increases, new layout designs have been developed to reduce the effect of parasitic resistance and capacitance associated with long interconnect lines. Multiple storage cores are used to divide up the total storage area into separate **blocks** of cells; each block defines a specific address range. These can be further divided into **sub-blocks**, **sub-sub-blocks**, and so on, until the size of the cell array can be accessed without excessive delay. Figure 13.30 illustrates the concept. The advantage of dividing the storage array is that the word line routing can be divided into multiple paths as shown. This **divided-word line** (DWL) architecture simplifies the decoder circuits and speeds up access by distributing the loads over multiple stages.

The logic diagram in Figure 13.31 shows the wiring scheme. The outputs of the primary decoder define the **global word lines**. These are used in conjunction with the Block select signals to activate the various **block word lines**. Progressing to the next lowest level in the hierarchy takes us to the **sub-block word lines**, and finally to the **sub-sub-block** signals. The gates and associated interconnect parasitics can be designed to speed up the row selection process. A simple approach is to apply the techniques of

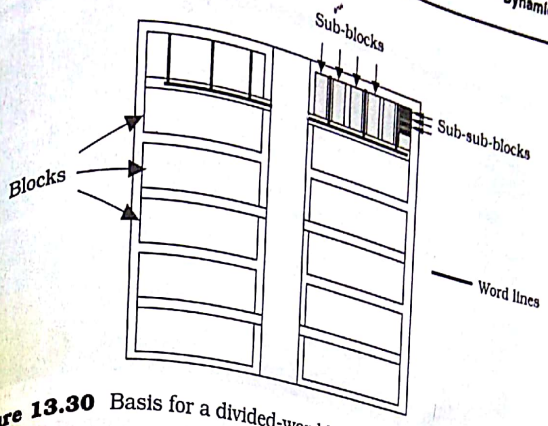


Figure 13.30 Basis for a divided-word line architecture RAM layout

Logical Effort to equalize the delay through each stage (logic gate with output load). This allows one to increase the speed without resorting to excessively large transistors. The technique can be applied to each level in the hierarchy.

Figure 13.32 is a micro-photograph of a portion of an SRAM that uses a multiple-block architecture. Cell block arrays can be identified by their regular layout patterning and are clearly seen in the bottom half. The upper section of the photograph contains decoders, drivers, and auxiliary circuitry including sense amplifiers.

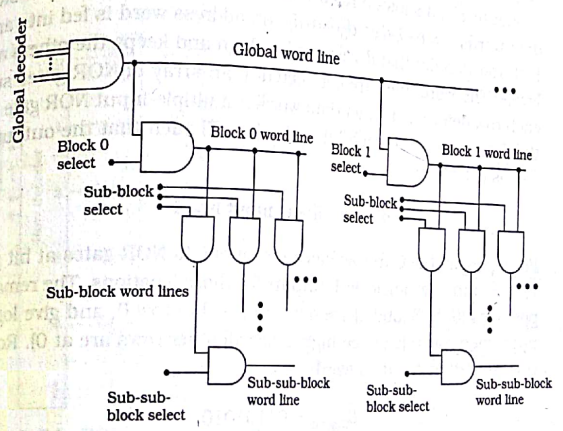


Figure 13.31 Logic for a DWL design

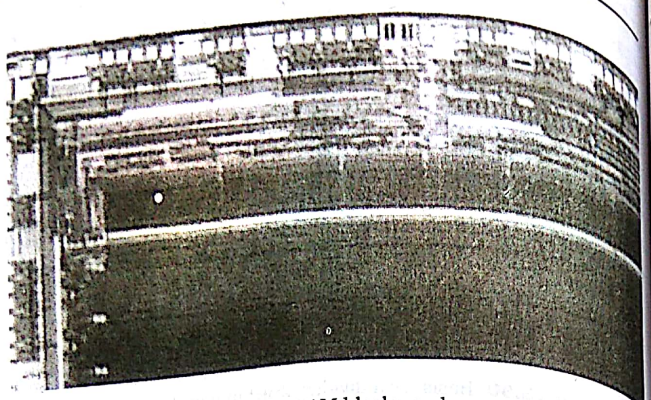


Figure 13.32 Photograph of SRAM blocks and support circuitry

13.4 ROM Arrays

Read-only memories (ROMs) are used for permanent bit storage. The structure of a ROM array is similar to that used for RAMs, but the individual bit cells are much simpler. The data stored in a basic read-only memory is created by the selective placement of FETs. Since this is accomplished in the physical design, the data cannot be altered once the chip is fabricated.

Figure 13.33 shows a ROM array that is uses NOR gates to store 8-bit data words $D = d_7d_6d_5d_4d_3d_2d_1d_0$. An address word is fed into an active-high row decoder that drives one line high and keeps the others at logic 0 levels. The word lines are connected to an array of NOR gates such that each row defines a distinct data word. A multiple-input NOR gate provides the data outputs for each bit d_i (i = 0,..., 7) such that the output of each gate is determined by

$$d_i = 0 \quad \text{if any input is a 1} \tag{13.23}$$

For example, the 0-th row has connections to NOR gates at bit positions 7, 4, 2, and 0, giving logic 1 outputs for those locations. The remaining bit positions (6, 5, 3, and 1) are not connected to row 0, and give logic 0 outputs when row 0 is driven high (since all other rows are at 0). Row 0 thus corresponds to the data word

$$D_{\text{Row 0}} = 01101010 \tag{13.24}$$

Every row is programmed in the same manner.

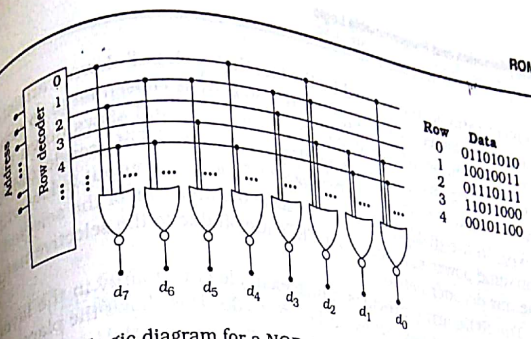


Figure 13.33 Logic diagram for a NOR-based ROM

The pseudo-nMOS implementation of the ROM is shown in Figure 13.34. Since only one pull-up pFET is required for each NOR gate, the task of programming centers on placement of nFETs that act as pull-down devices. A logic 0 output is obtained by providing a FET with its gate connected to the driving word line. This is understood by noting that when a pull-down transistor turns on, it provides a good connection to ground and pulls the output low. Pseudo-nMOS circuits are ratioed, so that the value of the output low voltage V_{OL} is determined by the nFET/pFET ratio $(\beta_n/\beta_p) > 1$ as discussed in Chapter 9. Selecting the nFET

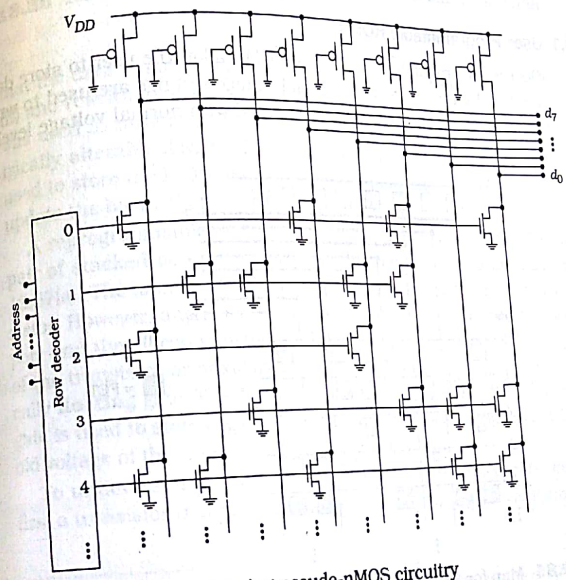


Figure 13.34 ROM array using pseudo-nMOS circuitry

aspect ratio is complicated by the fact that each pull-down transistor has an input capacitance of $C_G = C_{ox}WL$, so if W_n is chosen large to achieve a low V_{OL}, the word line capacitance increases and slows down the row decoder circuits. Another important characteristic of pseudo-nMOS circuits is that they dissipate DC power whenever an output is low with $V_{out} = V_{OL}$. In the ROM array, only the logic 0 output bits of the selected word consume power since all other FETs are off due to the selective nature of the row decoder network.

The ROM array provides a nice example of regularity in the layout. A simple FET map is provided in Figure 13.35. This shows the placement of the FETs relative to the input and output lines. Metal1 is used for the NOR gate connections running vertically (except for the VDD line), and the outputs are taken out on horizontal Metal2 lines. This results in the layout shown in Figure 13.36. Reprogramming is accomplished by adding or removing pull-down transistors.

Various approaches can be used to provide a library-based ROM with this design. One technique is to place a pull-down nFET at every intersection of a word line and a NOR output; a '0' is programmed into that location by connecting the word line to the FET gate with a poly contact. This is an example of a **mask-programmable** ROM where the stored data is defined by the poly-contact mask. Alternately, one can start with a blank nFET array and use a CAD tool to place transistors as needed.

13.4.1 User-Programmable ROMs

Electrically programmable ROMs (PROMs) allow the user to store data as required by the application. Special voltage settings are used to write to the cells. Read operations are performed with normal voltage levels, so

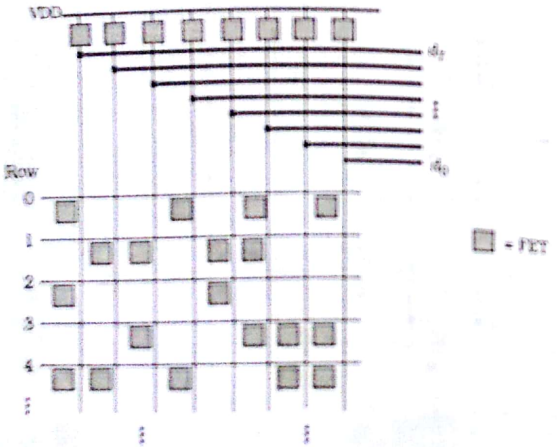


Figure 13.35 Map for ROM layout

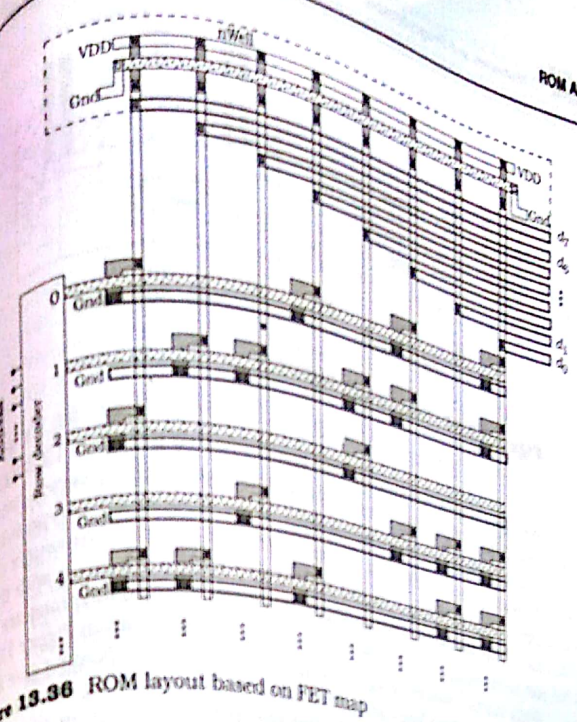


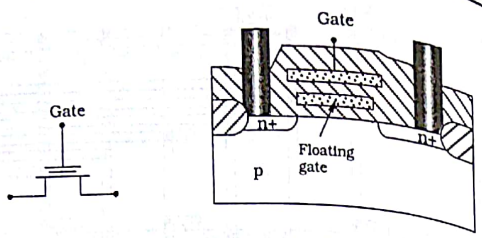
Figure 13.36 ROM layout based on FET map

that the data remains unchanged. Many ROM devices provide for erasure (E) and rewriting the contents of the array. Optical erasure using UV light was used in early EPROM designs, but these have been replaced by electrically alterable devices. Electrically-erasable EPROMs (E^2PROMs) are used to store the BIOS code in personal computers, and allow the user to update the board characteristics for new devices.[3]

A reprogrammable ROM array is built using special FETs that use a pair of stacked poly gates and has the circuit symbol shown in Figure 13.37(a). The topmost gate constitutes the usual gate terminal of the transistor. However, another poly gate layer is sandwiched in between the top poly and the silicon substrate. It is not electrically connected to any part of the transistor or auxiliary circuitry, and is therefore called an electrically **floating** gate. The details are shown in Figure 13.37(b). The floating gate is used to store negative electron charge, which increases the threshold voltage of the transistor above its normal value.

To understand the charge storage mechanism and effects, consider first a transistor that has zero charge on the floating gate. Applying a gate

[3] BIOS stands for Basic Input/Output System. The BIOS controls the booting procedure when a PC is powered-up and allows the operating system to be loaded into the system memory.

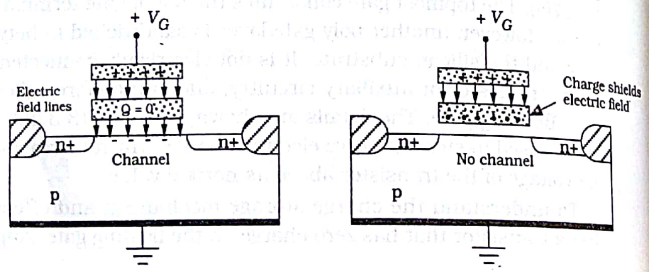


(a) Symbol (b) Structure

Figure 13.37 Floating-gate MOSFET

voltage creates the electric field lines indicated in Figure 13.38(a). Since the gate is floating, the structure acts like a pair of series-connected capacitors and field with field lines terminating on the p-type substrate. This creates the electron channel layer and allows drain-source current flow. A value of $Q = 0$ on the floating gate gives a transistor with the normal (low) threshold voltage V_{Tn}. If negatively charged electrons are stored on the floating gate, the field lines are altered as shown in Figure 13.38(b). With normal values of V_G, the negative charge on the floating gate shields the electric field lines and prevents them from reaching the silicon surface. No channel is formed, and the device remains in cutoff. If we increase the gate voltage to a high value $V_{Tn,H}$, then the FET goes active. However, we can design the transistor so that $V_{Tn,H} > V_{DD}$, which insures that it is always in cutoff when placed in a circuit.

The dual threshold voltage characteristics give rise to the EPROM scheme shown in Figure 13.39 for an 8-bit word; only the read circuitry is shown for simplicity. The nFETs with a low (normal) threshold voltage V_{Tn} are denoted by "L" while high threshold voltage devices with $V_{Tn,H} > V_{DD}$



(a) Normal V_{Tn} state (b) Increased $V_{Tn,H}$ state

Figure 13.38 Effect of charge storage on the floating gate

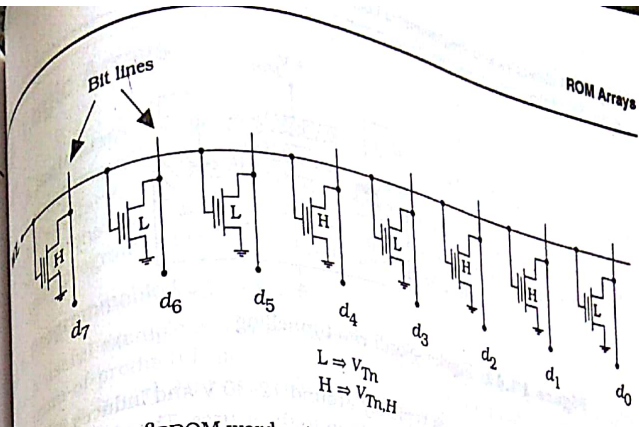


Figure 13.39 A E^2PROM word using floating-gate nFETs

are labeled as "H" in the drawing. The gates of the row are connected to the word line signal WL from a row decoder. When the word is accessed with $WL = 1$, a voltage of $V_{WL} = V_{DD}$ is applied to the logic transistors. Low (L) threshold voltage nFETs turn on and pull the output to ground (0 V). High (H) threshold voltage transistors, on the other hand, remain in cutoff and produce logic 1 output voltages using pull-up devices on the bit lines. For the example shown, the output word is

$$d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 = 10010110 \tag{13.25}$$

To allow the storage of arbitrary data words, a floating-gate FET must be wired as a pull-down device on every NOR gate in the array.

Now that we have seen how the floating-gate FETs are used in circuits, let us examine the programming technique. In the structure we have been studying, electronic charge is transferred to the gate using quantum mechanical tunnelling using **hot electrons**, which are highly energetic channel electrons. The conditions needed to induce the tunnelling are obtained by using a gate voltage $V_{G,prog}$ to create an electron channel, and a programming voltage applied to the drain as shown in Figure 13.40.

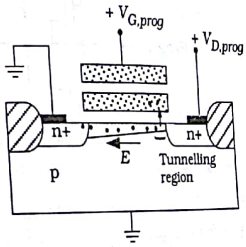


Figure 13.40 Programming a floating-gate FET

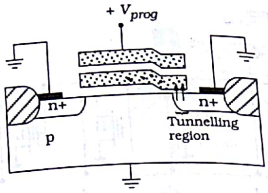


Figure 13.41 Fowler-Nordheim tunnelling

The value of $V_{D,prog}$ is typically around 12–30 V and induces a large electric field that points from the drain to the source. The strong electric field accelerates electrons to the drain side, where they can scatter and tunnel to the floating gate. Retention times are usually estimated to be on the order of 10–20 years in these structures.

Other floating-gate transistor designs use **Fowler-Nordheim emission** where the gate geometry is modified so that a portion of the floating gate extends over the n+ drain as shown in Figure 13.41. A large gate voltage is applied to create a large electric field between the substrate and gate, which enhances the tunnelling through the oxide. Both the drain and source are grounded during the programming operation. The cell circuit in Figure 13.42 accomplishes the write operation by pulsing the Program line while the word line is at $WL = 1$ and the bit and source lines are both grounded.

Erasure is accomplished by reversing the polarity of the applied voltages. General EPROM arrays allow bit erasure, while **flash EPROMs** are wired in a manner that erases large blocks of cells simultaneously. The latter is particularly useful for temporary storage of large data files such as those generated by digital photography.

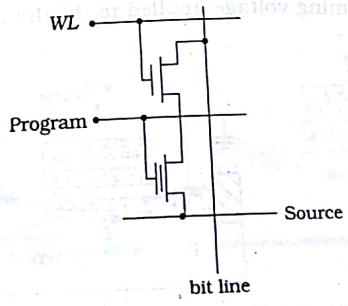


Figure 13.42 EEPROM cell with write line

13.5 Logic Arrays

A **logic array** is usually interpreted as being a structured unit that can be "programmed" to provide various functions and system tasks. They can be similar in structure to ROMs but are usually applied in more general situations. In most cases, the structure of the logic array is invariant. The user programs the array using a defined set of rules.

13.5.1 Programmable Logic Arrays

A useful example is a **programmable logic array** (PLA) for creating SOP (sum-of-product) logic expressions. Consider a group of four input variables a, b, c, d. An SOP function has the form

$$f = \sum_i m_i(a, b, c, d) \tag{13.26}$$

where $m_i(a, b, c, d)$ are minterms, i.e., terms that consist of the input variables or their complements ANDed together. In canonical form, a minterm uses every variable; the numerical value of the subscript i is the equivalent decimal value of the word. For the present case, the lowest order minterms are given by

$$m_0 = \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot\bar{d} \qquad m_1 = \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot d \qquad m_2 = \bar{a}\cdot\bar{b}\cdot c\cdot\bar{d}$$
$$m_3 = \bar{a}\cdot\bar{b}\cdot c\cdot d \qquad m_4 = \bar{a}\cdot b\cdot\bar{c}\cdot\bar{d} \qquad m_5 = \bar{a}\cdot b\cdot\bar{c}\cdot d \tag{13.27}$$

An SOP form is obtained by OR'ing the minterms such as

$$f = \bar{a}\cdot\bar{b}\cdot c\cdot\bar{d} + \bar{a}\cdot b\cdot\bar{c}\cdot d + a\cdot\bar{b}\cdot\bar{c}\cdot\bar{d} + a\cdot\bar{b}\cdot c\cdot d \tag{13.28}$$

This is equivalent to an AND-OR logic sequence, which forms the basis for the VLSI implementation. The general structure of an AND-OR PLA is shown in Figure 13.43. The inputs are fed to the AND-plane that calcu-

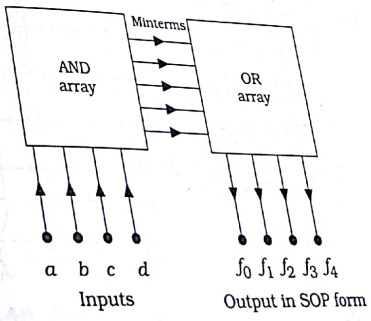


Figure 13.43 Structure of an AND-OR PLA

lates the needed minterms. These are then fed to the OR-plane, where they are OR'ed together. Several SOP functions $f_0, f_1, \ldots$, can be formed by adding different minterms together. The gate-level diagram for a PLA is shown in Figure 13.44. The AND plane has been chosen to provide five minterms by combining inputs. These feed into the OR plane where they are used to create the output functions shown. Each output can be determined by tracing the inputs. For example

$$f_x = m_0 + m_4 + m_5 \tag{13.29}$$

as can be easily verified by directly reading the connection. Another example is

$$f_y = m_3 + m_4 + m_5 + m_6 \tag{13.30}$$

It is important to remember that each minterm is in canonical form, i.e., every variable appears in either complemented or uncomplemented form. These can be read directly from a function table.

The most usable type of PLA circuit is one that can be easily programmed by placing one FET per connection. Pseudo-nMOS or dynamic logic styles provide the obvious solution, but both have sizing problems with series-connected transistors as needed in NAND gates. One solution is to create the AND-OR array in a NOR-based logic cascade. Figure 13.45(a) shows the desired logic flow. Using complemented inputs to a NOR gate produces the AND function in the schematic of Figure 13.45(b), this can be seen from evaluating

$$g = \overline{(\bar{a} + \bar{b})} = a \cdot b \tag{13.31}$$

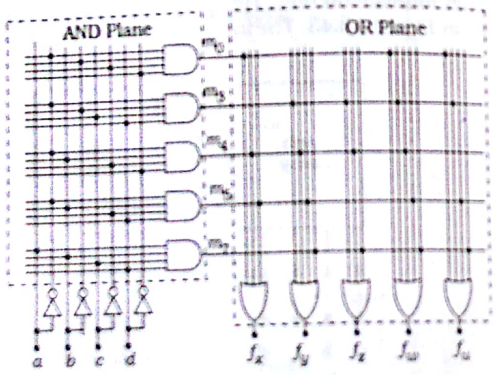
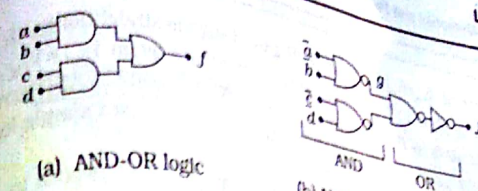


Figure 13.44 Logic gate diagram of the PLA



(a) AND-OR logic (b) NOR-based design

Figure 13.45 NOR-gate PLA logic

using a DeMorgan reduction. The OR operation is produced with a NOR-NOT cascade; alternately, the complement $\bar{f}$ may be used as the output. The basic structure for a dynamic cascade based on this equivalence is illustrated by the simple example in Figure 13.46. The AND plane consists of NOR gates, with the outputs fed to the NOR gates. The complemented outputs are

$$\bar{f}_1 = \overline{a \cdot b \cdot c + \bar{a} \cdot \bar{b} \cdot \bar{c}}$$
$$\bar{f}_2 = \overline{a \cdot b \cdot c} \tag{13.32}$$

Other functions can be created by expanding the gates in either (or both) plane(s). A single clock ϕ is used to synchronize the operation of both planes. These can be separated into two different signals where the input plane slightly leads the output plane. In this design, the size of the nFETs determines the high-to-low times of both gates.

A static pseudo-nMOS design is obtained by eliminating the n-channel evaluate FETs, and grounding the gates of the pFETs. Since pseudo-nMOS circuits are ratioed, the output-low voltage V_{OL} depends on the

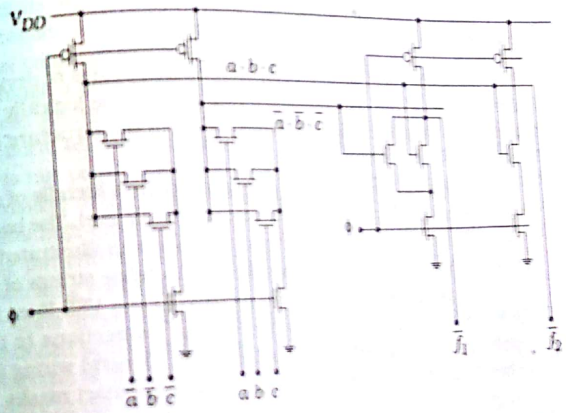


Figure 13.46 A dynamic CMOS PLA based on NOR gates

value of (β_n/β_p) but the NOR gate design is straightforward. Another problem is that DC power dissipation occurs when $V_{out} = V_{OL}$. An alternate approach to the PLA is to design a NOR-based OR-AND array that implements product of sum (POS) functions. This uses simple logic and is the subject of Problem [13.12].

The programmability feature of this approach is seen in Figure 13.47. Either logic plane can be represented as an array of NOR gates. The output for each gate is determined by the nFET logic block. The designer programs the array by placing and wiring FETs into each gate as needed. The physical design is characterized by regular FET patterns placed in between precharge-evaluate transistor pairs.

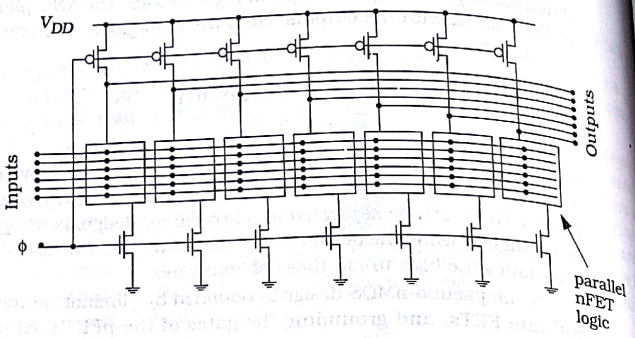


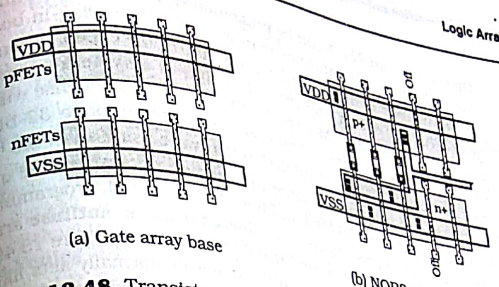
Figure 13.47 Generic NOR-based logic plane

13.5.2 Gate Arrays

The term **gate array** is used to describe an entire class of devices. It usually refers to a user-programmable chip that can be logically configured as needed. Programming techniques vary with the structure of the chip itself.

At the physical design level, a gate array consists literally of an array of logic gates that are wired at the interconnect mask level. The basis for this type of device is a "sea" of predefined transistors as illustrated in Figure 13.48(a). Common n+ or p+ regions are used to define strings of nFET and pFETs, respectively. The location of metal VDD and VSS lines is known, but in an unprogrammed array there are no connections to the power supply; these are added by the contact mask. General wiring is accomplished by adding features to both the metal and contact masks.

Logic gates can be constructed using any design style, but combinational static designs are the easiest and most common. The wiring for a



(a) Gate array base

(b) NOR3 gate wiring

Figure 13.48 Transistor arrangement in a gate array

NOR3 gate is shown in Figure 13.48(b). Power supply contacts have been added to the VDD and VSS lines. Complementary nFET/pFET pairs are wired using metal and poly contacts. Transistor wiring is attained with metal and active contacts to create three parallel nFETs and three series pFETs. Electrical isolation in this type of array is achieved using cutoff transistors. In the NOR3 gate, the logic pFETs are isolated from the other p-channel transistors by tying a gate to the power supply VDD; an nFET gate connected to VSS provides the same effect on the nFET string. More advanced technologies use oxide isolation to overcome high leakage levels. This leads to more stable electrical characteristics, but makes the wiring more complicated unless common n+/p+ strings are included.

Gate arrays allow rapid prototyping using semicustom logic. Since the designer has access to individual transistors, there are no a priori constraints on the circuits or logic. The process can be taken to a high level of automation since the gate map is well defined. One drawback is that the final configuration is from a masking step, so that the wafer must be sent through a portion of the manufacturing line. This adds time to the design cycle. Another problem may arise in the electrical characteristics. In a uniform sea of gates, the aspect ratios are the same for each FET polarity. It is not possible to adjust the sizes, eliminating some circuits that are sensitive to the transistor sizings. Large values of (W/L) are common to allow high-drive circuits to be designed in a simple manner. Since the wiring parasitics, especially the line capacitance, increase with length, switching speed may be a problem.

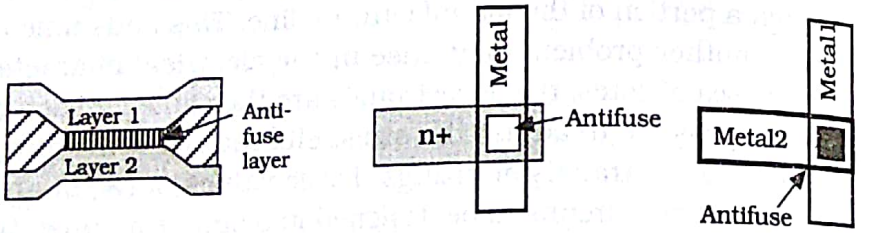
Gate array designs result in finished die that are larger than could be obtained with custom sizing. This is not usually a concern since the main objective is to provide fast turnaround with semicustom control of the circuitry at the FET level.

Field-programmable gate arrays (FPGAs) have become very popular for testing logic designs and limited production products. As implied by

their name, an FPGA can be programmed "in the field" as opposed to having to be in a laboratory or manufacturing line. Plug-in boards for PCs allow the user to define the logic and "burn" the circuit with a fast, simple procedure. Testing the logical design of a system becomes a more straightforward task. Modern FPGAs are extremely dense with hundreds of thousands of gates. Even complex systems like a 32-bit pipelined microprocessor can be emulated in hardware using an FPGA.

FPGAs are designed as general logic networks where the user defines what elements are activated and how they are wired. Programming is usually an electrical process that is achieved using an **antifuse** arrangement that is built into the structure at the physical level. Figure 13.49(a) shows an antifuse. By definition, an antifuse device normally acts like an open circuit between two conducting layers 1 and 2 as shown. The antifuse is "blown" by forcing a high current through it, which melts the antifuse layer and creates a low-resistance contact between the Metal1 and n+. This can be applied between a metal and active n+/p+ region as in Figure 13.49(b), or between a metal and poly layer as in Figure 13.49(c).

Primitive networks in FPGAs vary with the vendor. The most general approach is to provide a logic block that contains elements such as combinational logic, latches and/or flip-flops, multiplexors, and lookup tables. System design is facilitated by a CAD tool set and design notes that cover many standard situations, such as finite state machines or concurrent processors. The details of FPGA design are well beyond the scope of this text, but several excellent books have been written on the subject. Smith's book (Reference [10]) covers many different commercial products, and is almost encyclopedic in nature. Reference [3] is a very readable introduction to the subject. Vendor data from web sites and in hardcopy form is quite abundant, with much tutorial information usually provided to help the potential customer.



(a) Structure (b) Metal-active (c) Metal-metal

Figure 13.49 Programmable antifuse arrangements

System-Level Physical Design

14

CMOS VLSI design revolves around the physical characteristics of silicon circuitry. A high-level architectural function can be implemented using any of several different circuits, but the blocks must be wired together to complete the design of the chip. In this chapter we will examine aspects of macroscale physical design.

14.1 Large-Scale Physical Design

Up to this point we have concentrated on studying relatively simple logic functions using CMOS technology. Once a cell library is created, it can be used to build a complex VLSI system by instancing logic units into the master design. This bottom-up description represents the transition from low-level primitives to a high-level system. Associated with this change in the design hierarchy are physical considerations that arise when the emphasis evolves from the bit-level micron-sized structures to larger-size units and sections. Characteristics of interconnect lines, signal distribution, and large circuit blocks are only a few of the problems that must be dealt with. Since the performance of the chip is ultimately determined by its components, large-scale aspects of physical design and layout are critically important. Embedding a fast switching network into a slow interconnect mesh neutralizes the speed obtained in the low-level design.

A typical top-down design flow that illustrates some of the important problems is shown in Figure 14.1. After the HDL description of the system has been written and verified, logic synthesis provides a first design. The network is then subjected to simulation with emphasis on insuring that the design behaves as desired. Physical design steps of floorplanning, cell placement, clocking paths, and signal routing are next in the sequence. These are concerned with solving the large-scale problems of chip build-

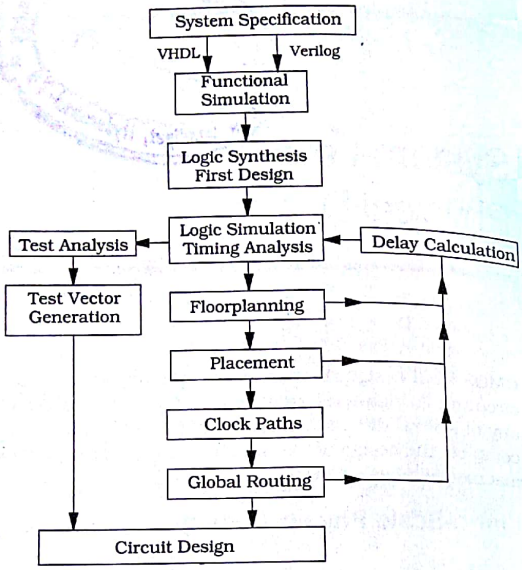


Figure 14.1 Design flow showing chip-level physical design issues

ing, and are the subject of the present chapter. Functionality of a high-speed digital system is closely tied to the overall timing: clock distribution, gate delays, latching, and other considerations. Timing information provides a feedback path that links each successive design level to the next. Testing problems are also examined during the design phase as indicated on the left side of the drawing. The problem of testing VLSI chips is so important to the industry that an entire chapter is dedicated to it later in the book.

The size of a silicon integrated circuit increases with the complexity and transistor count of the network. Overall, our target is to create a chip with the smallest area A_{chip} to increase both the number of die N per wafer and the overall yield Y. For a given process and wafer size, the goal may be to select the smallest standard size die for the design. Regardless of the actual die size, a number of physical design-related problems arise when building the entire chip. The most important issues are introduced in this chapter. Some of the problems, such as interconnect delay and wiring, are considered to be major obstacles for projected designs of the future.

14.2 Interconnect Delay Modeling

One of the most critical problems in high-density VLSI is dealing with interconnect lines. They introduce signal delays that affect system timing, and often lead to extremely complicated layout routing problems. We have seen many examples of the special treatment given to interconnect lines during the fabrication process: silicided poly lines, multiple metal interconnect layers, and the use of copper. In this section, we will build equivalent circuits that provide mathematical models for interconnect delays. Routing techniques are discussed in Section 14.5.

The starting point of our analysis is the simple isolated interconnect line shown in Figure 14.2 that represents an arbitrary material layer on the chip. The dimensions of the line are shown as having a length l, a width w, and a thickness t. The line resistance R_{line} from In to Out is given by the formula

$$R_{line} = R_s\left(\frac{l}{w}\right)\ \Omega \tag{14.1}$$

where R_s is the sheet resistance of the layer and (l/w) is the number of squares with dimensions $(w \times w)$. Defining the resistance per unit length r by

$$r = \frac{R_s}{w}\ \ \Omega/\text{cm} \tag{14.2}$$

the equation becomes

$$R_{line} = rl \tag{14.3}$$

This shows the increase of parasitic resistance with the line length l in a form that is easy to deal with. It is obvious that high-resistivity layers

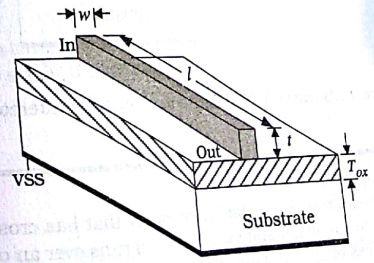


Figure 14.2 Isolated interconnect line

such as polysilicon are more problematic than low-resistivity metals.

The total line capacitance C_{line} can be estimated using the simple parallel-plate formula

$$C_{line} = \frac{\varepsilon_{ox} l w}{T_{ox}} \text{ F} \tag{14.4}$$

where T_{ox} is the thickness of the insulating oxide between the line and the substrate. C_{line} is also called the **self-capacitance** of the line. Although the equation provides a first estimate, it ignores the effects of fringing electric fields from the edges and sides when the line is at a positive voltage; Figure 14.3 illustrates the origin of the fringing field corrections. An empirical equation for the capacitance per unit length c that accounts for these effects is [10]

$$c = \varepsilon_{ox}\left[1.15\left(\frac{w}{T_{ox}}\right) + 2.8\left(\frac{t}{T_{ox}}\right)^{0.222}\right] \text{ F/cm} \tag{14.5}$$

such that

$$C_{line} = cl \tag{14.6}$$

is the total capacitance in farads. Conceptually, the first term accounts for fringing from the bottom side of the line while the second term depends on the thickness t and is due to the sidewall effects. In a CMOS chip, the largest line capacitance values will be on the layers that are the closest to the substrate.

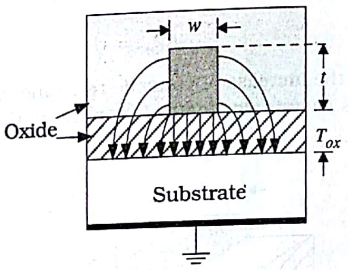


Figure 14.3 Electric field lines for an isolated interconnect

Example 14.1

Consider a first-level metal interconnect that has cross-sectional dimensions of $w = 0.35$ μm and $t = 0.7$ μm and runs over an oxide layer that has a thickness of $T_{ox} = 0.9$ μm. The capacitance per unit length is

$$c = (3.9)(8.854\times10^{-14})\left[1.15\left(\frac{0.35}{0.9}\right) + 2.8\left(\frac{0.7}{0.9}\right)^{0.222}\right]$$

$$= 1.07 \text{ pF/cm} \tag{14.7}$$

If the sheet resistance is $R_s = 0.02$ Ω, then the resistance per unit length is

$$r = \frac{0.02}{0.35\times10^{-4}} = 571 \text{ Ω/cm} \tag{14.8}$$

An interconnect with a length of $l = 40$ μm is characterized by

$$R_{line} = (571)(40\times10^{-4}) = 2.29 \text{ Ω}$$

$$C_{line} = (1.07)(40\times10^{-4}) = 4.28 \text{ fF} \tag{14.9}$$

Increasing the length to $l = 225$ μm gives

$$R_{line} = (571)(225\times10^{-4}) = 12.85 \text{ Ω}$$

$$C_{line} = (1.07)(225\times10^{-4}) = 24.1 \text{ fF} \tag{14.10}$$

While the resistance remains relatively small (because R_s is small), the parasitic line capacitance is on the order of MOSFET values, making it important to the analysis.

Calculating the values of R_{line} and C_{line} allows us to construct circuit models to study the effects of the parasitics. The simplest approach to modeling the line is to construct the simple two-element circuit shown in Figure 14.4 to include the effects of the line from In to Out. This is called a "single-rung ladder circuit" because of the way it is drawn. If the input voltage $v_i(t)$ changes from a 0 to a 1 (or vice versa) in a step-like manner, the change in the output voltage $v_o(t)$ is delayed by the time constant

$$\tau = R_{line}C_{line} \tag{14.11}$$

which constitutes a low-order estimate of how the interconnect line affects the signal transmission. We must delve deeper into the origin of

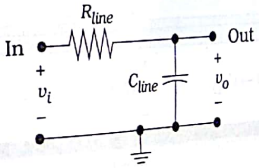


Figure 14.4 Single-rung ladder model

increases, the position z needed to keep ξ at the same value also increases in a non-linear (squared) manner.

In practice, it is easier to use the numerical values provided in computational programs such as MatLab and MathCad. While the differential equation provides more accurate values for the signal delays, adding realistic constraints such as a finite line with a capacitive node makes the analysis quite complicated. Only a few problems can be solved in closed form, making numerical analysis mandatory. Because of this, VLSI designers tend to prefer the simpler RC models for first estimates in most signal paths.

14.2.1 Signal Delay versus Line Length

One of the most important results of this analysis is the dependence of the delay time constant τ on the length l of the line. Consistent results can be obtained using any of the analysis techniques discussed above.

The simplest estimate for the time constant is from equation (14.11) in the form

$$\tau = R_{line}C_{line} \tag{14.34}$$

Substituting from equations (14.3) and (14.6) gives

$$\tau = Bl^2 \tag{14.35}$$

where $B = rc$ is a constant for the line with units of sec/cm^2. This shows that the signal delay is proportional to the square of the line length. The quadratic dependence is illustrated in Figure 14.11, and has a major effect on the use of long interconnects in VLSI networks. Since unequal line lengths have different signal delays, this requires that interconnect routing be carefully planned, especially in critical datapaths. The system designer must take care to accurately model and design interconnect networks to insure that the system can operate at the desired speed.

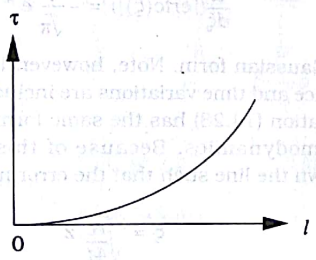


Figure 14.11 Parabolic dependence of the time delay on line length

Example 14.2

Suppose that the signal delay on an interconnect of length 50 μm is known to be 0.13 ps. If the line is increased to 100 μm, the delay rises to a value of

$$\tau = \left(\frac{0.13}{50^2}\right)100^2 = 0.52 \text{ ps} \tag{14.36}$$

where we have used the given data to find B in the equation. A line that is 200 μm long has a delay of

$$\tau = \left(\frac{0.13}{50^2}\right)200^2 = 2.08 \text{ ps} \tag{14.37}$$

This shows that the relative lengths of interconnect wires become the important factor.

14.2.2 Dealing with Interconnect Delays

Signal delays along interconnect lines can be limiting factors in high-speed system design. In critical single-bit paths, they must be added to the normal gate delays to obtain an accurate picture of the problem. They become especially important for global distribution of signals such as the clock ϕ in a synchronous system. In word-oriented architectures where every bit of an n-bit word must be transmitted from one unit to another, the slowest bit-transmission path determines the data flow speed for the entire word. Careful routing schemes are used in an attempt to equalize the line length for every bit.

Since interconnect delays are intrinsically circuit and layout problems, detailed analyses are usually performed by circuit design groups. They are charged with the task of creating accurate circuit models that can be used in simulation programs without consuming excessive amounts of computer time. Design manuals often provide this type of information in circuit or code form that can be used directly by other designers by inserting numerical values for the parameters. High-level system and logic designers are then able to estimate interconnect delays along all paths for use in architectural verifications.

The importance of interconnect delays cannot be overemphasized. More examples will be presented later in the context of specific problem areas. Of these, the problem of global clock distribution discussed in the next chapter is one of the most critical aspects of high-speed synchronous design.

14.3 Crosstalk

Whenever an interconnect line is placed in close proximity to any other interconnect line, the conductors are coupled by a parasitic capacitance. Pulsing a voltage on one of the lines induces a stray signal on all lines that are coupled to it. This phenomenon is called **crosstalk**. Since a stray signal at the input to a logic gate may cause an incorrect output, dealing with crosstalk problems is a very important aspect of designing high-density VLSI chips.

Consider the layout shown in Figure 14.12 where Line 1 and Line 2 are coupled to each other by parasitic capacitance. Capacitance increases as the distance between two conductors decreases. The strongest coupling thus occurs where the two lines are separated by the minimum distance S. Suppose that Line 1 and Line 2 have voltages of V_1 and V_2, respectively. Let us denote the total coupling capacitance by C_c. The voltage difference $V_{12} = (V_1 - V_2)$ induces a current i_{12} flowing from Line 1 to Line 2 as described by the basic capacitor equation

$$i_{12} = C_c \frac{dV_{12}}{dt} = C_c \frac{d(V_1 - V_2)}{dt} \tag{14.38}$$

This expresses the fundamental basis of capacitive crosstalk. Strong coupling exists if C_c is large, or if the voltage difference $(v_1 - v_2)$ changes very quickly in time. Since high-speed design requires large time derivatives (corresponding to rapidly changing signals), VLSI design usually deals with crosstalk by reducing C_c and then examining the switching limits.

Figure 14.13 provides a cross-sectional view of two adjacent interconnect lines (labeled 1 and 2) that are separated by a spacing S. An empirical equation for the coupling capacitance per unit length is [10]

$$c_c = \varepsilon_{ox}\left[0.03\left(\frac{w}{T_{ox}}\right) + 0.83\left(\frac{t}{T_{ox}}\right) - 0.07\left(\frac{t}{T_{ox}}\right)^{0.222}\right]\left(\frac{S}{T_{ox}}\right)^{-1.34} \tag{14.39}$$

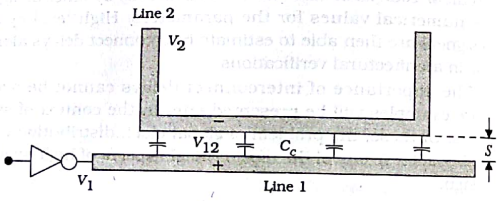


Figure 14.12 Capacitive coupling between two lines

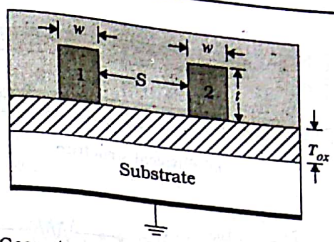


Figure 14.13 Geometry used for coupling capacitance calculation

in units of F/cm. The total coupling capacitance C_c in farads is computed from

$$C_c = c_c l_c \tag{14.40}$$

where l_c is the length of the coupled section with spacing S. This shows that c_c increases as $(1/S)^{4/3}$, so that using a small line spacing increases the coupling capacitance. Layout design rules account for this fact in the numerical values specified for S_{min}. A crosstalk-based design rule overrides the fact that the lithography may be capable of producing finer line spacing.

An example of using the coupling capacitance C_c is shown in Figure 14.14. The original circuit [Figure 14.14(a)] along with the layout dimensions provides the details needed to compute the parasitics including the value of C_c. This can be used in a lumped-element equivalent model such as that shown in Figure 14.14(b). This approach uses a symmetric RC equivalent circuit for each interconnect line and models the coupling using a single capacitor with value C_c in the middle. The alternate model shown in Figure 14.15 divides the coupling into two capacitors ($C_c/2$); other topologies are possible.

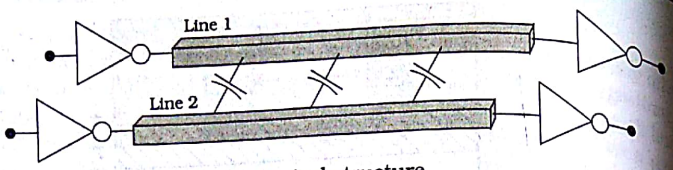
The total capacitance of a line consists of the self-capacitance (from the line to ground) and any coupling terms. Let us denote the self-capacitance of Lines 1 and 2 by C_{11} and C_{22}, respectively; these are just the appropriate values of C_{line} for each. The total capacitance seen looking into Line 1 is given by

$$C_1 = C_{11} + C_c \tag{14.41}$$

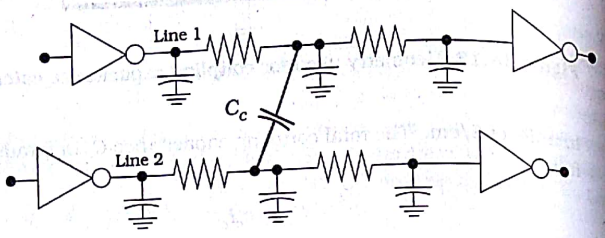
Similarly, the total capacitance of Line 2 is

$$C_2 = C_{22} + C_c \tag{14.42}$$

These values are important for designing the driving circuits for each line. If an interconnect line is coupled to two adjacent lines, then both lines



(a) Physical structure



(b) RC model

Figure 14.14 Lumped-element coupling circuit model

contribute to the total capacitance. Figure 14.16 illustrates the case where Line 1 interacts with both Line 2 and Line 3. The total capacitance of Line 1 per unit length in the closely spaced sections is

$$c_1 = c + 2c_c \text{ F/cm} \tag{14.43}$$

Multiplying by the length gives the total capacitance. Theoretically, every conductor on the chip interacts with every other conductor. In practice, however, we usually limit ourselves to **nearest-neighbor** coupling; this is justified by the decrease in c_c with increasing S.

Let us examine the physics of the interaction for the 3-line network.

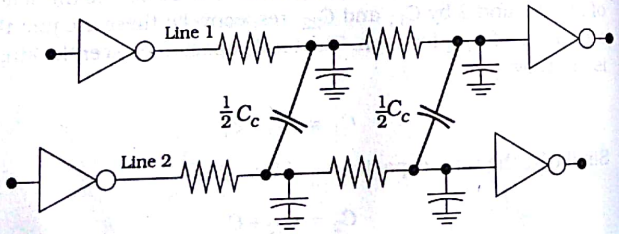


Figure 14.15 Alternate model for coupling circuit

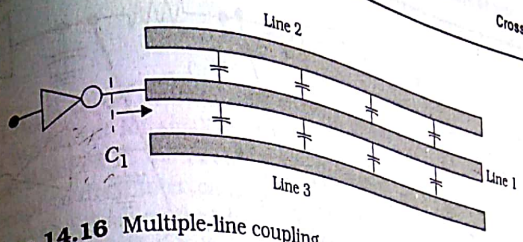


Figure 14.16 Multiple-line coupling

With nearest-neighbor coupling, the charge Q_1 on Line 1 is written as

$$Q_1 = C_{11}V_1 + C_{12}(V_1 - V_2) + C_{13}(V_1 - V_3) \tag{14.44}$$

where C_{12} and C_{13} are the coupling capacitances from lines 1 to 2 and lines 1 to 3, respectively, and V_i is the voltage on the i-th line ($i = 1, 2, 3$). The charges on Line 2 and Line 3 are

$$\begin{aligned} Q_2 &= C_{21}(V_2 - V_1) + C_{22}V_2 \\ Q_3 &= C_{31}(V_3 - V_1) + C_{33}V_3 \end{aligned} \tag{14.45}$$

since they do not interact with each other. These equations may be combined to give the matrix form

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} = \begin{bmatrix} (C_{11}+C_{12}+C_{13}) & -C_{12} & -C_{13} \\ -C_{21} & (C_{22}+C_{21}) & 0 \\ -C_{31} & 0 & (C_{33}+C_{31}) \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} \tag{14.46}$$

We can show that the capacitance matrix is symmetric with $C_{ij} = C_{ji}$. Since current is the time derivative of the charge, we compute

$$\begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} = \begin{bmatrix} (C_{11}+C_{12}+C_{13}) & -C_{12} & -C_{13} \\ -C_{21} & (C_{22}+C_{21}) & 0 \\ -C_{31} & 0 & (C_{33}+C_{31}) \end{bmatrix} \frac{d}{dt}\begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} \tag{14.47}$$

This shows that any change in the line voltage (dV_1/dt) changes both $i_2(t)$ and $i_3(t)$, with the magnitude of the effect dependent on the size of the capacitors and the rate of change of the voltage. Similarly, changing voltages of (dV_2/dt) or (dV_3/dt) causes $i_1(t)$ to flow. A circuit level model for the 3-line network is shown in Figure 14.17. This may be analyzed using a standard circuit simulator.

It is worthwhile at this point to introduce a formula for the capacitance of an isolated plate as shown in Figure 14.18. The total plate capacitance C_p [F] may be estimated using

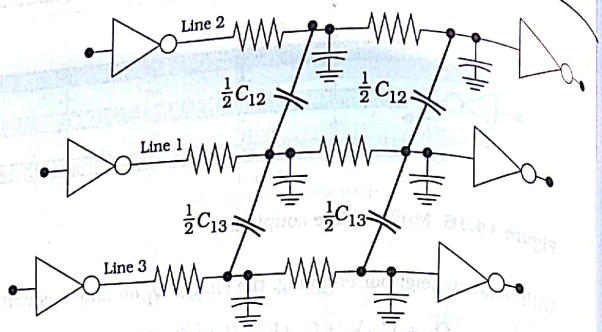


Figure 14.17 Circuit model for 3-line coupling problem

$$C_p = \varepsilon_{ox}\left[1.15\left(\frac{A}{T_{ox}}\right) + 1.40P\left(\frac{t}{T_{ox}}\right)^{0.222} + 4.12T_{ox}\left(\frac{t}{T_{ox}}\right)^{0.728}\right] \quad (14.48)$$

where $A = wl$ is the bottom area of the plate and $P = 2(w+l)$ is the circumference. In this expression, the first term accounts for the bottom and fringing contributions, the second term gives the sidewall additions, and the last term accounts for the corners.

Crosstalk also occurs between overlapping lines on different material layers. Figure 14.19 illustrates the case where Metal2 crosses over a Metal1 line. The critical parameter is the oxide thickness $T_{ox,12}$ between the two layers. The simplest approximation for overlap capacitance $C_{ov,12}$ is the parallel-plate formula

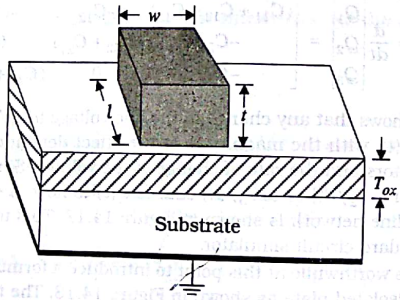


Figure 14.18 Plate geometry

$$C_{ov,12} = \frac{\varepsilon_{ox}A_{ov}}{T_{ox,12}} \quad (14.49)$$

with $A_{ov} = w_1 w_2$ as the overlap area. Although this ignores fringing fields, it is sufficient for small overlap areas. Since we attempt to minimize all capacitances in the circuit, this leads to the layout strategy where we attempt to draw interconnects on a given layer so that they are perpendicular to the lines on the layers directly above and below it. In other words, we try to draw Metal1 lines that are perpendicular to Metal2 lines, Metal2 lines that are perpendicular to Metal3 lines, and so on.

14.3.1 Dealing with Crosstalk

Crosstalk problems can be very involved and often require specialized group studies. While simplified equations are useful for estimating the coupling parameters, computer programs have been developed to calculate 2- and 3-dimensional coupling parameters directly from Maxwell's equations of electromagnetic theory. In addition to detailed information on the field strengths and gradients, these codes provide numerical values for parameters such as c and c_c that can be used directly to calculate the capacitances. The line resistance and capacitances are used to create equivalent circuit models, which are then subjected to simulation studies using programs such as SPICE.

The detailed examination of crosstalk is usually delegated to the domain of circuit designers and electromagnetics specialists. VLSI system designers usually see the results of these studies in various parametric forms such as noise fluctuation levels on the nodes. Other times, the research results in design rule changes at both the device and system level.

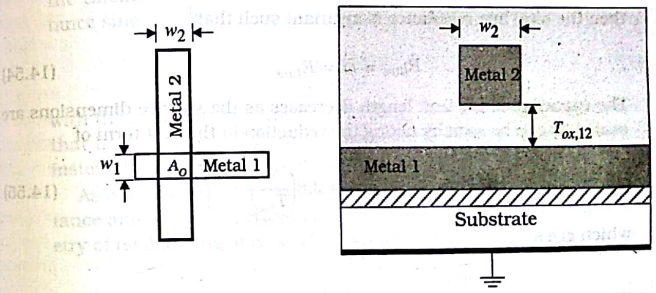


Figure 14.19 Overlap capacitance

14.4 Interconnect Scaling

Although scaling theory[2] was originally introduced to describe FETs, it can be applied to interconnect lines to yield useful results. This is consistent with the view that a shrinking transistor is accompanied by reduced size interconnects that affect the overall performance of a circuit.

The three geometrical dimensions of an interconnect line that are set in the processing are the width w, the thickness t, and the oxide thickness T_{ox}. Improved lithography allows us to reduce the linewidth to a smaller value

$$\tilde{w} = \frac{w}{s} \tag{14.50}$$

where $s > 1$ is the scaling factor. This is the fundamental effect of scaling the surface geometry of an interconnect line.

To understand how this affects the electrical characteristics, recall that the sheet resistance of a layer is given by

$$R_s = \frac{\rho}{t} \tag{14.51}$$

The resistivity ρ is not changed by shrinking w, so that the line resistance per unit length increases as seen by writing

$$\tilde{r} = \frac{R_s}{\tilde{w}} = sr \tag{14.52}$$

If we assume that the line length l scales according to

$$\tilde{l} = \frac{l}{s} \tag{14.53}$$

then the total line resistance is invariant such that[3]

$$\tilde{R}_{line} = \tilde{r}\tilde{l} = R_{line} \tag{14.54}$$

The capacitance per unit length decreases as the surface dimensions are scaled as can be seen by noting the reduction in the first term of

$$\tilde{c} = \varepsilon_{ox}\left[1.15\left(\frac{\tilde{w}}{T_{ox}}\right) + 2.8\left(\frac{t}{T_{ox}}\right)^{0.222}\right] \tag{14.55}$$

which gives

[2] Scaling theory was introduced in Section 6.5.1.

[3] Note that scaling a layout does not mean that the interconnect lengths scale in the same manner.

$$\tilde{c} = \varepsilon_{ox}\left[1.15\left(\frac{\tilde{w}}{sT_{ox}}\right) + 2.8\left(\frac{t}{T_{ox}}\right)^{0.222}\right] \tag{14.56}$$

If we can ignore fringing or assume that the first term dominates,

$$\tilde{c} \approx \frac{c}{s} \tag{14.57}$$

Scaling the line length l then approximates the new line capacitance as

$$\tilde{C}_{line} = \tilde{c}\tilde{l} = \frac{C_{line}}{s^2} \tag{14.58}$$

which shows a $1/s^2$ reduction.

A polysilicon line will exhibit the highest sheet resistance in a process even if it is silicided. In this case, it would be important to decrease the line length so as not to increase the value of R_{line}. The time constant for the line scales according to

$$\tilde{\tau} = \tilde{R}_{line}\tilde{C}_{line} = \frac{\tau}{s^2} \tag{14.59}$$

which is due to the reduction in the line capacitance. Note that if l is not scaled, then τ is not affected by the surface scaling. The same comments apply to an arbitrary metal line where the time constant is dominated by the line capacitance.

Let us now examine the situation where the vertical dimensions t and T_{ox} are reduced such that

$$\tilde{t} < t \quad , \quad \tilde{T}_{ox} < T_{ox} \tag{14.60}$$

which are generally valid for any vertical scaling factor $s_v > 1$. Reducing the thickness t has the undesirable effect of increasing the sheet resistance since

$$R_s = \frac{\rho}{t} \tag{14.61}$$

with the resistivity ρ a constant. Similarly, a thinner oxide increases c, so that both R_{line} and C_{line} would increase, leading to longer delays. If we instead **increase** t and T_{ox}, both r and c would be smaller.

As a final case, let us examine how scaling affects the coupling capacitance and, therefore, crosstalk. A brute-force scaling of the surface geometry of neighboring lines would stipulate that

$$\tilde{S} = \frac{S}{s} \tag{14.62}$$

where S is the spacing between the lines as shown previously in Figure

14.13. To see how this affects the coupling, let us examine the basic formula

$$c_c = \varepsilon_{ox}\left[0.03\left(\frac{\tilde{w}}{T_{ox}}\right) + 0.83\left(\frac{t}{T_{ox}}\right) - 0.07\left(\frac{t}{T_{ox}}\right)^{0.222}\right]\left(\frac{S}{T_{ox}}\right)^{-1.34}$$

for the coupling capacitance per unit length. The overall multiplier

$$c_c \propto \left(\frac{1}{S}\right)^{1.34}$$

shows that decreasing S increases the coupling capacitance. While actual increase may be offset somewhat by scaling other terms such as and T_{ox}, reducing the crosstalk often dominates all other considerations including real estate consumption. As processes evolve, reduced values S are possible, but line spacings do not scale as much as FET sizes.

This short discussion of interconnect scaling illustrates how the idea is used to provide ideas for improving performance. By itself, it is a highly idealized approach that cannot be implemented in practice due to processing limitations. However, it does act as a catalyst for future improvements which explains in part why it is still considered worth studying.

14.5 Floorplanning and Routing

Cell-based VLSI design employs predesigned electronic circuit modules that are instanced as needed to create the system. At the chip level, each module is viewed as a block that consumes area and must be wired into the network. This step links the system and subsystem architecture directly to the silicon physical design. At this scale, the physical design problems are very different from those encountered in transistor and gate layout. Long interconnects, complex wiring meshes, and other large-scale factors are critical to the overall performance of the finished design. Many aspects of **design automation** are devoted to these problems.

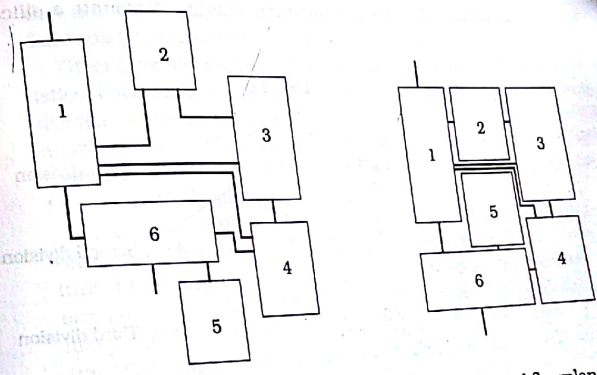
Floorplanning deals with the placement of the logic blocks into the overall design. This is done very early in the design cycle so that area budgets can be assigned, and the overall size of the chip can be estimated. The initial floorplan can be based on large, complex functional units and how they are wired together by the system architecture. Once an area is allocated, the designs of the subsystems that make up the large units are themselves constrained. Floorplans are drawn before the physical design is even started, so it requires an experienced group of designers to provide guidelines based on previous designs.

When a logic module is placed into the design, it must be wired to other units. While simple point-to-point wires may be easy to add, mod-

ern VLSI systems require millions of connections. Interconnect routing schemes have been developed to provide a structured approach to attacking this problem. **Place-and-route** CAD tools are useful for wiring complex systems. The designer specifies the beginning and end points of an interconnect wire, and the tool generates a solution that does not violate any design rules. These codes are based on different types of graph algorithms, and exhibit various degrees of success.

Let us examine the problem of floorplanning first. Any digital system can be decomposed into a set of units that are wired together in a specific manner. A simple example is shown in Figure 14.20(a). The interconnect lines indicate communication between distinct blocks and each carries a different number of bits. If the dimensions of the blocks are scaled according to their actual size in the layout, then we may use the block diagram to create a first-try floorplan as in Figure 14.20(b). Wiring channels are provided in between adjacent blocks to facilitate the wire routing. This is important for minimizing interconnect lengths, and may be mandatory if we are limited to only one or two interconnect layers.

This example can be used to illustrate a **sliceable floorplan**, which is one of the simplest approaches to large-scale layout. A sliceable floorplan is defined as either a single module, or a floorplan that can be partitioned into modules (or module groups) using a vertical or horizontal line that traverses a contiguous group of modules. Let us redraw the floorplan of Figure 14.20(b) into the equivalent representation shown in Figure 14.21(a). A vertical cut line may be used to obtain the first division shown in Figure 14.21(b). The second division into the groups portrayed in Figure 14.21(c) is obtained using two horizontal cut lines. This process may



(a) System block diagram (b) Initial floorplan

Figure 14.20 Using a block diagram for initial floorplanning

14.13. To see how this affects the coupling, let us examine the basic formula

$$c_c = \varepsilon_{ox}\left[0.03\left(\frac{\tilde{w}}{T_{ox}}\right)+0.83\left(\frac{t}{T_{ox}}\right)-0.07\left(\frac{t}{T_{ox}}\right)^{0.222}\right]\left(\frac{S}{T_{ox}}\right)^{-1.34} \tag{14.63}$$

for the coupling capacitance per unit length. The overall multiplying factor

$$c_c \propto \left(\frac{1}{S}\right)^{1.34} \tag{14.64}$$

shows that decreasing S increases the coupling capacitance. While the actual increase may be offset somewhat by scaling other terms such as $\tilde{w}$ and T_{ox}, reducing the crosstalk often dominates all other considerations including real estate consumption. As processes evolve, reduced values of S are possible, but line spacings do not scale as much as FET sizes.

This short discussion of interconnect scaling illustrates how the theory is used to provide ideas for improving performance. By itself, it is a highly idealized approach that cannot be implemented in practice due to processing limitations. However, it does act as a catalyst for future improvements which explains in part why it is still considered worth studying.

14.5 Floorplanning and Routing

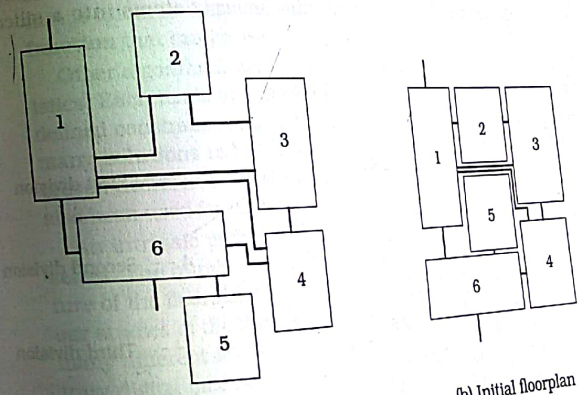
Cell-based VLSI design employs predesigned electronic circuit modules that are instanced as needed to create the system. At the chip level, every module is viewed as a block that consumes area and must be wired into the network. This step links the system and subsystem architecture directly to the silicon physical design. At this scale, the physical design problems are very different from those encountered in transistor and gate layout. Long interconnects, complex wiring meshes, and other large-scale factors are critical to the overall performance of the finished design. Many aspects of **design automation** are devoted to these problems.

Floorplanning deals with the placement of the logic blocks into the overall design. This is done very early in the design cycle so that area budgets can be assigned, and the overall size of the chip can be estimated. The initial floorplan can be based on large, complex functional units and how they are wired together by the system architecture. Once an area is allocated, the designs of the subsystems that make up the large units are themselves constrained. Floorplans are drawn before the physical design is even started, so it requires an experienced group of designers to provide guidelines based on previous designs.

When a logic module is placed into the design, it must be wired to other units. While simple point-to-point wires may be easy to add, modern VLSI systems require millions of connections. Interconnect routing schemes have been developed to provide a structured approach to attacking this problem. **Place-and-route** CAD tools are useful for wiring complex systems. The designer specifies the beginning and end points of an interconnect wire, and the tool generates a solution that does not violate any design rules. These codes are based on different types of graph algorithms, and exhibit various degrees of success.

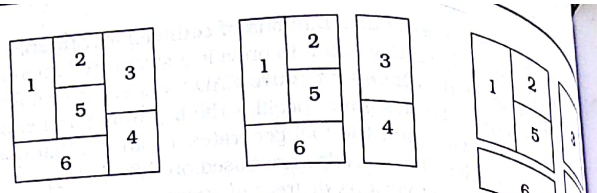
Let us examine the problem of floorplanning first. Any digital system can be decomposed into a set of units that are wired together in a specific manner. A simple example is shown in Figure 14.20(a). The interconnect lines indicate communication between distinct blocks and each carries a different number of bits. If the dimensions of the blocks are scaled according to their actual size in the layout, then we may use the block diagram to create a first-try floorplan as in Figure 14.20(b). Wiring channels are provided in between adjacent blocks to facilitate the wire routing. This is important for minimizing interconnect lengths, and may be mandatory if we are limited to only one or two interconnect layers.

This example can be used to illustrate a **sliceable floorplan**, which is one of the simplest approaches to large-scale layout. A sliceable floorplan is defined as either a single module, or a floorplan that can be partitioned into modules (or module groups) using a vertical or horizontal line that traverses a contiguous group of modules. Let us redraw the floorplan of Figure 14.20(b) into the equivalent representation shown in Figure 14.21(a). A vertical cut line may be used to obtain the first division shown in Figure 14.21(b). The second division into the groups portrayed in Figure 14.21(c) is obtained using two horizontal cut lines. This process may



(a) System block diagram (b) Initial floorplan

Figure 14.20 Using a block diagram for initial floorplanning



(a) Original floorplan (b) First division (c) Second divis

Figure 14.21 Sliceable floorplan example

be continued until only separate modules remain. The divisions ca described using the tree structure drawn in Figure 14.22. The num denote the connected module groups, and the slicing process can be s by the division level indicated at the bottom of each branch. Note that lowest entries are the basic modules. **An** example of a non-sliceable flo plan is shown in Figure 14.23(a). **This** cannot be divided by using either horizontal or a vertical cut line without dividing up a module. However can be used as module in a larger sliceable floorplan, such as illustrated in Figure 14.23(b). The sliceable design can be described by a tree str work. If a non-sliceable module is used in a sliceable design, then a tr structure can still be created, but the cut lines must be defined in a re restrictive manner. When a floorplan tree can be constructed, the desig is called a **hierarchically defined floorplan**. These are conceptually straightforward to understand and have been used as the basis for deri ing layout algorithms that follow the tree structure.

One approach to floorplanning is the concept of **simulated annealin** which is based on the manner in which crystal formation occurs in sol materials. Recall that implanting impurity atoms into a silicon wafe

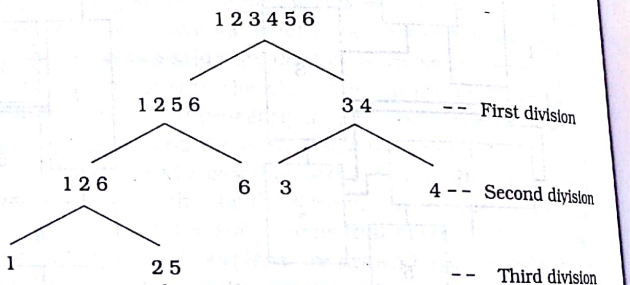
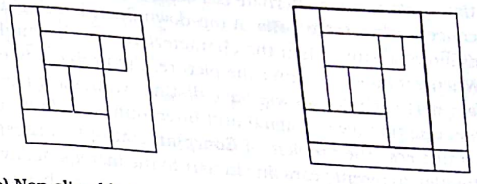


Figure 14.22 Binary division tree



(a) Non-sliceable floorplan (b) Use in a sliceable design

Figure 14.23 Example of a non-sliceable floorplan

causes damage to the crystal structure that is healed using an annealing step. The wafer is heated, imparting thermal energy to the atoms and inducing motion. As the temperature is lowered, the atoms seek minimum energy positions which places them at required points in the crystal lattice. Simulated floorplan annealing is an algorithmic technique that is based on the possible groupings at the vertex of a binary tree; the dependency on the tree structure restricts the approach to sliceable floorplans.

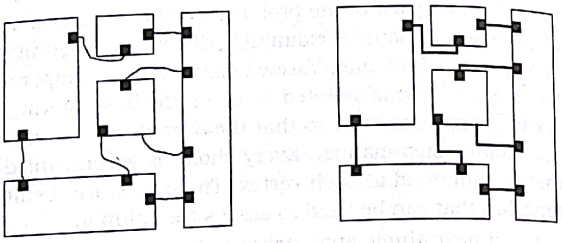
Consider a sliceable floorplan that describes a complex system. It is usually possible to construct more than one binary tree that describes the floorplan. Since every tree is valid, the existence of multiple trees expands the **solution space** of the problem. Simulated annealing takes advantage of multiple solutions by examining different configurations of the problem in a sequential manner. Various solutions are compared at each vertex of the tree, and one is selected as being the best. Moving to the next vertex starts the process over, so that the overall solution has been constructed in a step-by-step manner. Every choice is accompanied by a **cost metric** that is minimized at each vertex. The procedure results in a **global cost function** that can be used to assess the solution.

Other algorithmic approaches to floorplanning have similar characteristics. Each tends to centralize on designing floorplans using a few well-defined constraints and minimization parameters. Iterative routines allow many solutions to be compared. Hierarchical-based approaches have proven useful in many situations, and linear programming packages have also been used for tackling the problem.

An intrinsic problem of floorplanning is defining the modules themselves. The number of modules and the wiring nets establish the structure of the floorplan tree, which in turn affects the design. We know from our studies of the VLSI design hierarchy that cellular-based designs have many different levels of complexity and implementations. Given an arbitrary digital module, it can always be decomposed into smaller units or it can be used as a primitive element in a larger module. If only one implementation is available for a module, it is called a **fixed cell**

times two or more alternate cell designs are available for use, defining the concept of **variable cells**. A top-down floorplanning algorithm produces different results when the characteristics of the modules are changed. When variable cells enter the picture, the problem is complicated by the fact that each design will have distinct values for the chip area, power consumption levels, signal and interconnect delays, and other critical parameters. The problem of **floorplan sizing** has received considerable attention in recent years due in part to the increasing layout complexity of large chips. A common constraint in most approaches is the requirement of rectangular cells since they are the easiest to use in solving the general problems of floorplanning and cell placement in general.

Once the general floorplan has been established, interconnect routing becomes a critical concern. Although routing considerations enter into the choice of floorplan, interconnect-induced delay and timing problems won't be evident until the physical design is well under way. Routing is usually performed in two steps: global and detailed. Global routing deals with finding connection paths at the system level without specifying the actual geometric information. Detailed routing uses the global results and provides the layout specifics such as the layers used and placement of vias. Figure 14.24 provides a conceptual view of the two. Once the detailed routing is completed, the interconnect delays are set and the design must undergo verification.



(a) Global (b) Detailed

Figure 14.24 Routing steps

Global routing is usually modeled using graph theory. Many variations have been published in the literature, with some more intuitive than others. The routing model must be closely tied to the processing and the layout design style. Grid models and checkerboard-based algorithms are among the simplest. An example of a grid model is shown in Figure 14.25. Grid points are overlaid on the floorplan; grid points between the modules are used to connect the lines that make up a given net. A **line-probe** algorithm provides a routing path from a source point to a target point by generating lists of lines from both that are not blocked by a module. A

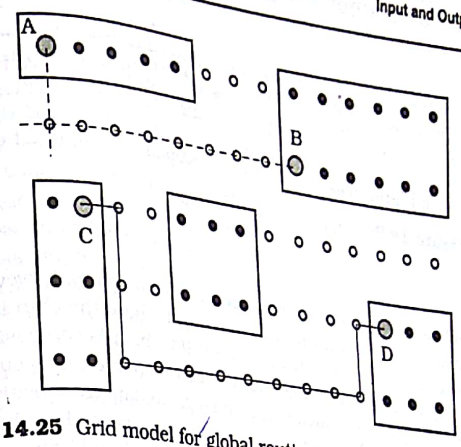


Figure 14.25 Grid model for global routing

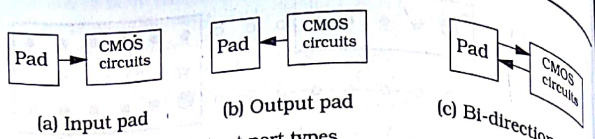
solution is found when a line from the source list intersects a line from the target list, as from A to B in the drawing. If a solution is not found, additional lists are created and linked to the source and target data. The **maze-routing** approach is an alternate technique that starts by examining possible paths from the source, and then expanding outward until the target is hit by one. The path drawn from C to D is an example of routing using the maze viewpoint. Many path-finding algorithms have been studied in an effort to find a quick and efficient approach.

Detailed routing is approached by solving one region of the global solution at a time. Routing regions are divided into two types. A **channel** is a rectangular routing area that is bounded on two sides by modules; a **switchbox** region is a rectangular area that is bounded on all four sides. Given a routing region, the solution must lead to a layout that can physically fit into the allocated area. Increasing the number of interconnect metal layers makes the routing easier, but via placement enters the problem.

The field of design automation has evolved to be a very sophisticated discipline within VLSI. New products to deal with complexity and verification issues are announced regularly. As the density and complexity of chips continue to increase, developing better and more efficient CAD tools becomes mandatory for advancement of the discipline.

14.6 Input and Output Circuits

On-chip CMOS circuitry is aided by the fact that it is shielded from th outside world by the packaging. This simplifies the design environme considerably. Capacitance levels are limited to the femtofarad level a



(a) Input pad (b) Output pad (c) Bi-directional pad

Figure 14.26 Input/Output port types

only local interactions are present. At some point, however, the circuitry must interact with the external world where the chip is used. Capacitances of populated printed-circuit (PC) boards can easily be in the 50-100 picofarad (pF) range. Static electric charge can accumulate to levels exceeding 50 kV, which is high enough to damage sensitive MOS transistors. Input and output circuitry must be designed to shield the internal network from being affected by these types of problems.

The interface ports between VLSI circuits and the outside world can be classified into three types: input, output, or bi-directional. These are shown for pad-type I/O in Figure 14.26. Input port circuits are designed to protect the sensitive CMOS circuits from high voltages, while the design of the output circuits tends to concentrate on driving high-capacitance PC board lines. Bi-directional circuits are combinations of these and allow a port to work as both an input and an output.

14.6.1 Input Circuits

Static electric charges tend to accumulate on dielectric (non-conductive) surfaces and can lead to extremely high voltages. A common cause of static charge buildup is due to frictional interactions.[4] Most people have experienced this in some form or another. Walking across a carpeted room may accumulate enough charge to create a spark when touching a door-knob. Physically, the voltage has grown to a value that is large enough to break down the insulating properties of air.

MOS field-effect transistors are extremely sensitive to electrostatic discharge (ESD) events. Consider the MOS structure shown in Figure 14.27 where the gate oxide thickness t_{ox} is typically less than 100 Å = 10 nm. With a gate voltage of V_G applied, the oxide electric field E_{ox} V/cm can be estimated as

$$E_{ox} \approx \frac{V_G}{t_{ox}} \tag{14.65}$$

With V_G = 3 V and t_{ox} = 0.01 μm, the electric field in the insulating layer has a value of about $E_{ox} \approx 3\times10^6$ V/cm. The maximum electric field that

[4] The study of friction is the basis for the field of **tribology**.

can be applied across a silicon dioxide insulator is typically on the order of about $E_{max} \approx 5\text{–}10\times10^6$ V/cm, depending upon the processing. If the electric field exceeds this value, breakdown occurs and current flows to the substrate. This destroys the insulating properties of the oxide and, hence, the transistor characteristics. Even a single bad FET renders the entire chip useless so this problem is taken quite seriously.

Static charge sources are encountered throughout the manufacturing process. In addition, the chips must be handled for everyday procedures such as shipping, unpacking, and board insertion. To account for this, modern CMOS chips are designed with circuits to protect the input transistors from excessive charge levels, and the devices are shipped in conductive foam. Even with these safeguards, it is prudent to follow the manufacturers' recommendations when handling chips. These are designed to reduce static charge levels by using grounded work planes and wrist straps that can drain excess charges to ground.

CMOS input protection circuits are designed to provide discharge paths away from the transistors. The network must be relatively transparent to the input signals and operate only when abnormal voltages are applied. The most common designs use resistor-diode networks to provide charge-dumping paths. Figure 14.28 summarizes the relevant characteristics of a pn junction diode structure. The anode and cathode sides of a diode are defined by the p- and n-regions shown in Figure 14.28(a). The voltage V is defined positive with the polarity shown: + on the anode and – on the cathode. Positive current I is defined to flow into the anode and out of the cathode. The circuit symbol for a diode is shown in Figure 14.28(b) with positive voltage and current. The I-V characteristics in Figure 14.28(c) show I as a function of V. A condition of $V > 0$ defines a forward bias where substantial current flows. Reversing the polarity so that the + side is on the cathode and the – side is on the anode defines a state

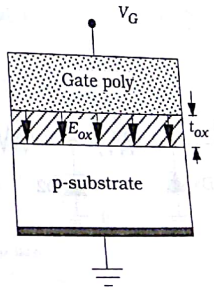
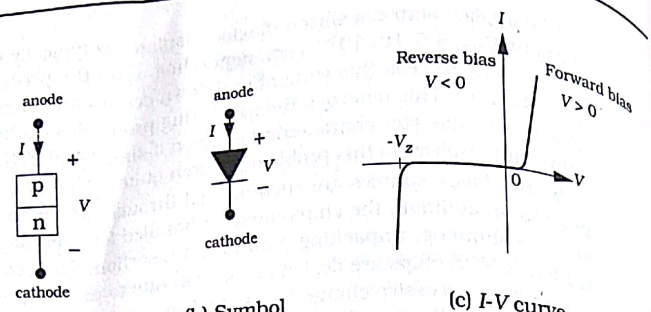


Figure 14.27 Oxide electric field in an MOS structure



(a) Structure (b) Symbol (c) *I-V* curve

Figure 14.28 Diode characteristics

of reverse bias such that $V < 0$. A reverse-biased diode blocks the current flow for low voltages, but exhibits a breakdown at the **Zener voltage** V_z when $V = -V_z$ as shown in the plot. The value of V_z is set by the doping levels, and is a characteristic of the junction. Junction breakdown is non-destructive: if the voltage is removed and then reapplied, the diode acts in a normal fashion.

Diodes can be used with resistors to form the input protection circuit shown in Figure 14.29. If an excessively large positive voltage is applied to the input pad, the resistors drop the voltage level along the input line. Under these conditions, the diodes D1 and D2 undergo breakdown and steer charge away from the gates of the input stage transistors. The diodes in a CMOS circuit typically have V_z = 10–12 V or smaller depending upon the junction. At the physical design level, the resistors can be made using n-implanted layers in p-substrate as shown in Figure 14.30(a). The n+ region uses a serpentine pattern to obtain a square layout with a resistance of

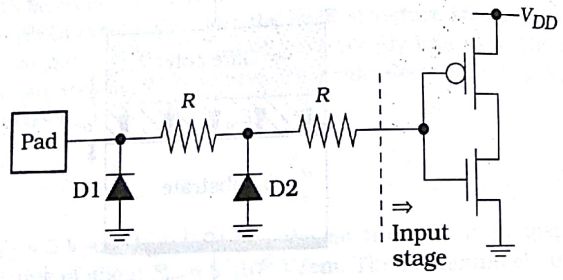
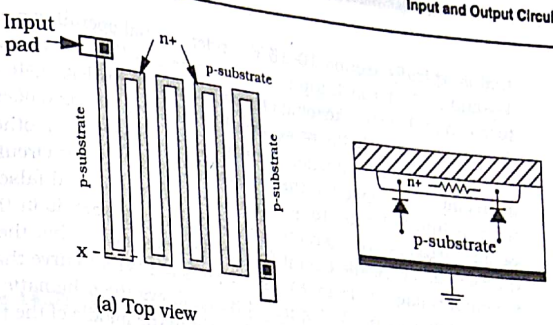


Figure 14.29 Input ESD protection circuit



(a) Top view (b) Cross-section at x

Figure 14.30 Input resistor-diode structure

$$R = R_{s,n} n \tag{14.66}$$

where $n = (l/w)$ is the number of squares. In this type of geometry. A 90° corner does not count as full squares; electrostatic analysis shows that $n_{corner} \approx 0.69$ is a reasonable estimate. The reverse-biased pn junction diodes are automatically built into the structure as shown in Figure 14.30(b). Both the resistor and the diode are distributed structures, not discrete devices as in the circuit diagram.

Another input protection scheme is shown in Figure 14.31. This uses diodes D1 and D2, but additional diodes D3 and D4 have been added between the input and the power supply. The circuit keeps the DC voltage reaching the gate in the range $[-V_d, V_{DD} + V_d]$ where $V_d \approx 0.7$ V is the **on-voltage** of the diode, i.e., the value required to induce current flow. A special high-threshold voltage nFET has been included to provide additional charge drainage. This uses the thick isolation field oxide (FOX) as a gate insulator so that the weak field-effect gives a high threshold voltage V_{TF}

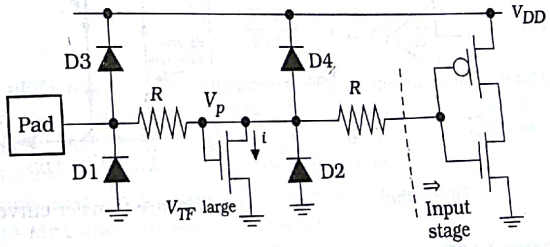
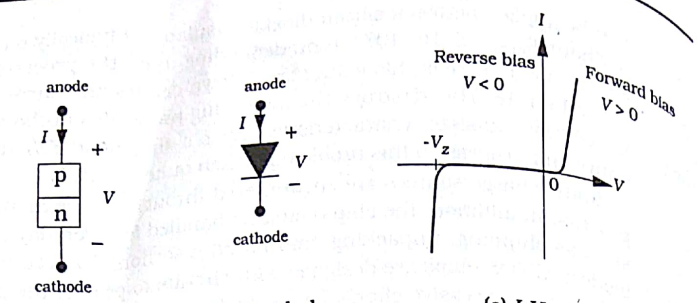


Figure 14.31 Alternate input protection circuits



(a) Structure (b) Symbol (c) *I-V* curve

Figure 14.28 Diode characteristics

of reverse bias such that $V < 0$. A reverse-biased diode blocks the current flow for low voltages, but exhibits a breakdown at the **Zener voltage** V_z when $V = -V_z$ as shown in the plot. The value of V_z is set by the doping levels, and is a characteristic of the junction. Junction breakdown is non-destructive: if the voltage is removed and then reapplied, the diode acts in a normal fashion.

Diodes can be used with resistors to form the input protection circuit shown in Figure 14.29. If an excessively large positive voltage is applied to the input pad, the resistors drop the voltage level along the input line. Under these conditions, the diodes D1 and D2 undergo breakdown and steer charge away from the gates of the input stage transistors. The diodes in a CMOS circuit typically have V_z = 10–12 V or smaller depending upon the junction. At the physical design level, the resistors can be made using n-implanted layers in p-substrate as shown in Figure 14.30(a). The n+ region uses a serpentine pattern to obtain a square layout with a resistance of

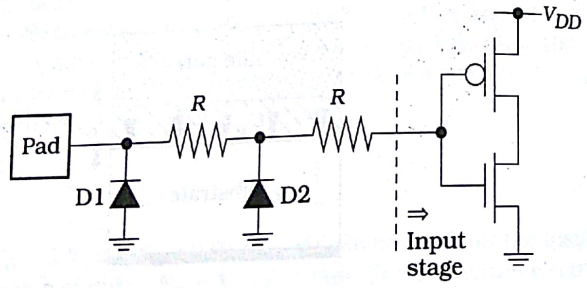
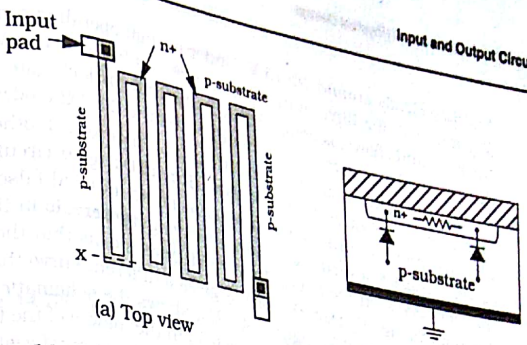


Figure 14.29 Input ESD protection circuit



(a) Top view (b) Cross-section at x

Figure 14.30 Input resistor-diode structure

$$R = R_{s,n}n \tag{14.66}$$

where $n = (l/w)$ is the number of squares. In this type of geometry. A 90° corner does not count as full squares; electrostatic analysis shows that $n_{corner} \approx 0.69$ is a reasonable estimate. The reverse-biased pn junction diodes are automatically built into the structure as shown in Figure 14.30(b). Both the resistor and the diode are distributed structures, not discrete devices as in the circuit diagram.

Another input protection scheme is shown in Figure 14.31. This uses diodes D1 and D2, but additional diodes D3 and D4 have been added between the input and the power supply. The circuit keeps the DC voltage reaching the gate in the range $[-V_d, V_{DD} + V_d]$ where $V_d \approx 0.7$ V is the **on-voltage** of the diode, i.e., the value required to induce current flow. A special high-threshold voltage nFET has been included to provide additional charge drainage. This uses the thick isolation field oxide (FOX) as a gate insulator so that the weak field-effect gives a high threshold voltage V_{TF}

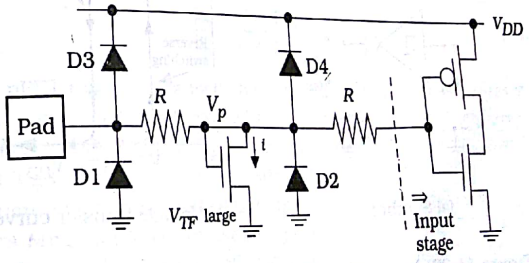
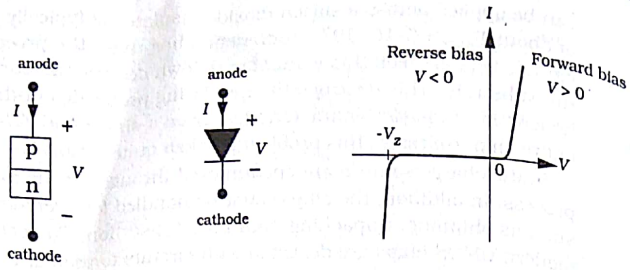


Figure 14.31 Alternate input protection circuits



(a) Structure (b) Symbol (c) *I-V* curve

Figure 14.28 Diode characteristics

of reverse bias such that $V < 0$. A reverse-biased diode blocks the current flow for low voltages, but exhibits a breakdown at the **Zener voltage** V_z when $V = -V_z$ as shown in the plot. The value of V_z is set by the doping levels, and is a characteristic of the junction. Junction breakdown is non-destructive: if the voltage is removed and then reapplied, the diode acts in a normal fashion.

Diodes can be used with resistors to form the input protection circuit shown in Figure 14.29. If an excessively large positive voltage is applied to the input pad, the resistors drop the voltage level along the input line. Under these conditions, the diodes D1 and D2 undergo breakdown and steer charge away from the gates of the input stage transistors. The diodes in a CMOS circuit typically have V_z = 10–12 V or smaller depending upon the junction. At the physical design level, the resistors can be made using n-implanted layers in p-substrate as shown in Figure 14.30(a). The n+ region uses a serpentine pattern to obtain a square layout with a resistance of

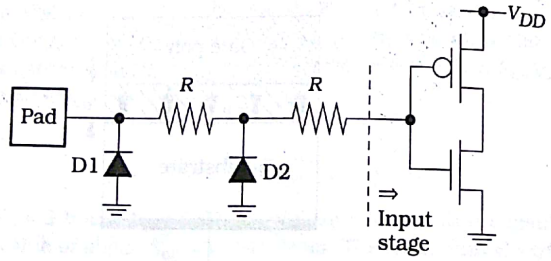
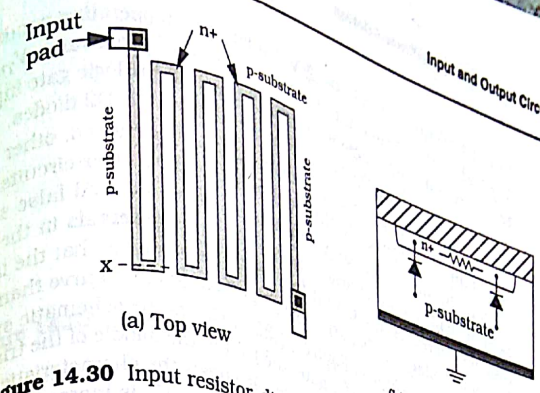


Figure 14.29 Input ESD protection circuit



(a) Top view (b) Cross-section at x

Figure 14.30 Input resistor-diode structure

$$R = R_{s,n}n \tag{14.66}$$

where $n = (l/w)$ is the number of squares. In this type of geometry. A 90° corner does not count as full squares; electrostatic analysis shows that $n_{corner} \approx 0.69$ is a reasonable estimate. The reverse-biased pn junction diodes are automatically built into the structure as shown in Figure 14.30(b). Both the resistor and the diode are distributed structures, not discrete devices as in the circuit diagram.

Another input protection scheme is shown in Figure 14.31. This uses diodes D1 and D2, but additional diodes D3 and D4 have been added between the input and the power supply. The circuit keeps the DC voltage reaching the gate in the range $[-V_d, V_{DD} + V_d]$ where $V_d \approx 0.7$ V is the **on-voltage** of the diode, i.e., the value required to induce current flow. A special high-threshold voltage nFET has been included to provide additional charge drainage. This uses the thick isolation field oxide (FOX) as a gate insulator so that the weak field-effect gives a high threshold voltage V_{TF}

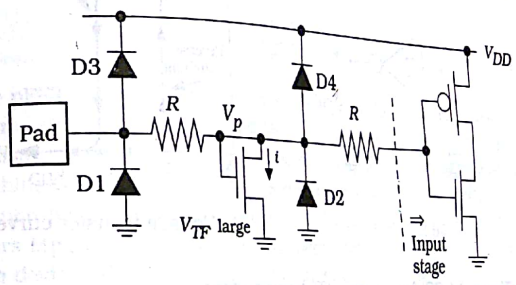
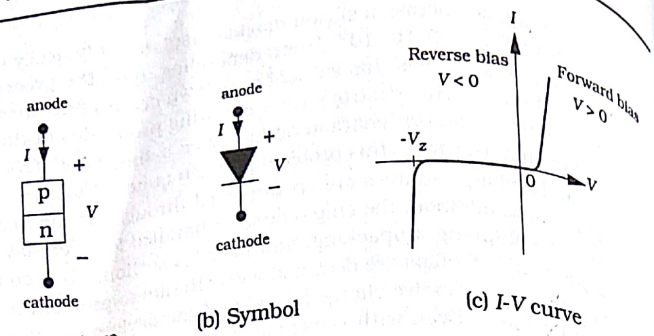


Figure 14.31 Alternate input protection circuits



(a) Structure (b) Symbol (c) *I-V* curve

Figure 14.28 Diode characteristics

of reverse bias such that $V < 0$. A reverse-biased diode blocks the current flow for low voltages, but exhibits a breakdown at the **Zener voltage** V_z when $V = -V_z$ as shown in the plot. The value of V_z is set by the doping levels, and is a characteristic of the junction. Junction breakdown is non-destructive: if the voltage is removed and then reapplied, the diode acts in a normal fashion.

Diodes can be used with resistors to form the input protection circuit shown in Figure 14.29. If an excessively large positive voltage is applied to the input pad, the resistors drop the voltage level along the input line. Under these conditions, the diodes D1 and D2 undergo breakdown and steer charge away from the gates of the input stage transistors. The diodes in a CMOS circuit typically have V_z = 10–12 V or smaller depending upon the junction. At the physical design level, the resistors can be made using n-implanted layers in p-substrate as shown in Figure 14.30(a). The n+ region uses a serpentine pattern to obtain a square layout with a resistance of

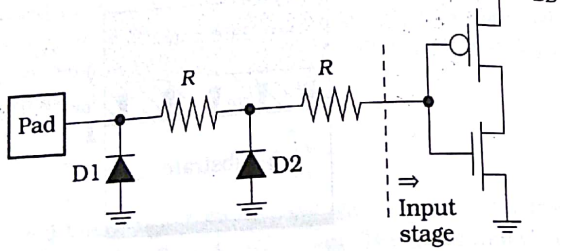
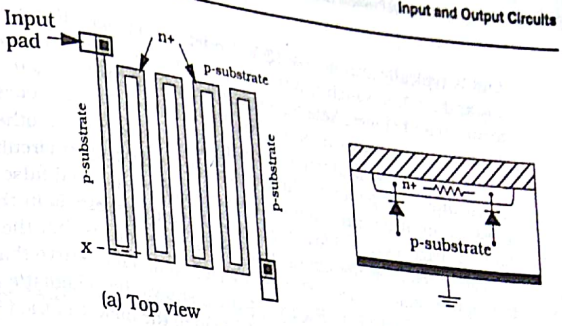


Figure 14.29 Input ESD protection circuit



(a) Top view (b) Cross-section at x

Figure 14.30 Input resistor-diode structure

$$R = R_{s,n} n \tag{14.66}$$

where $n = (l/w)$ is the number of squares. In this type of geometry. A 90° corner does not count as full squares; electrostatic analysis shows that $n_{corner} \approx 0.69$ is a reasonable estimate. The reverse-biased pn junction diodes are automatically built into the structure as shown in Figure 14.30(b). Both the resistor and the diode are distributed structures, not discrete devices as in the circuit diagram.

Another input protection scheme is shown in Figure 14.31. This uses diodes D1 and D2, but additional diodes D3 and D4 have been added between the input and the power supply. The circuit keeps the DC voltage reaching the gate in the range $[-V_d, V_{DD} + V_d]$ where $V_d \approx 0.7$ V is the **on-voltage** of the diode, i.e., the value required to induce current flow. A special high-threshold voltage nFET has been included to provide additional charge drainage. This uses the thick isolation field oxide (FOX) as a gate insulator so that the weak field-effect gives a high threshold voltage V_T

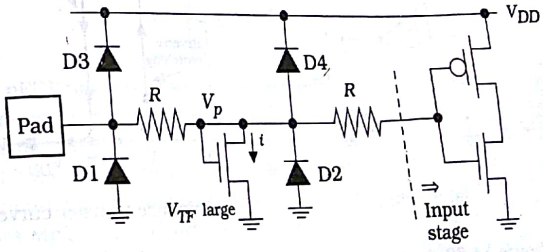
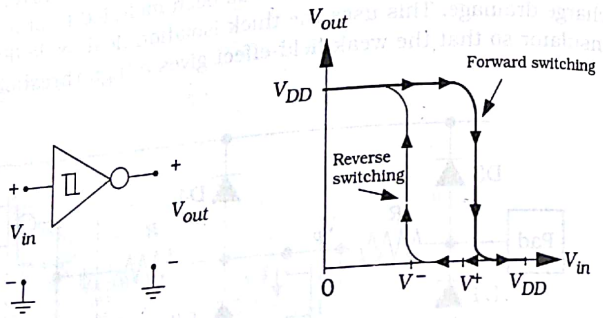


Figure 14.31 Alternate input protection circuits

that is typically around 10–15 V. Under normal operating conditions V_P < V_{TF} and $i = 0$. If a high input voltage increases V_P to a value V_{TF}, the FET turns on and i flows, keeping charge away from the logic gate input. Some designs employ only the protection FET and the D1, D2 diodes.

After the input protection network has been designed, other considerations must be applied to the receiver. **Schmitt trigger** circuits are often used as input circuits to guard against noise-induced false switching. Schmitt triggers are characterized by having **hysteresis** in their voltage transfer curves. At the circuit level, hysteresis means that the increasing the input voltage V_{in} from 0 V to V_{DD} gives a different curve than decreasing V_{in} from V_{DD} to 0 V. Figure 14.32(a) shows the schematic symbol for an inverting Schmitt trigger gate; the icon in the middle of the triangle distinguishes it from a simple inverter. It shows the characteristic shape of the VTC illustrated in Figure 14.32(b). When V_{in} is increased from 0 V, V_{out} stays high at V_{DD} until V_{in} reaches the forward trigger voltage V^+; V_{out} then drops to 0 V. For reverse switching, V_{in} starts at V_{DD} and is decreased giving $V_{out} = 0$ V. The output remains low until V_{in} is decreased to the reverse trigger voltage V^-. For $V_{in} < V^-$, $V_{out} = V_{DD}$. Note that $V^- < V^+$ is required for a functional Schmitt trigger. The hysteresis insures that small fluctuations on the rising or falling edge of the input signal do not induce a false switching event.

A CMOS Schmitt trigger circuit is shown in Figure 14.33. This uses a mirror design where the nFETs determine V^+ and the pFETs determine the value of V^-. Consider the nFET circuits. M1 and M2 are in series and are both driven by the input voltage. When $V_{in} = 0$, $V_{out} = V_{DD}$ and M3 is on. Since the drain of M3 is connected to the power supply, it acts as a feedback path. As V_{in} is increased, it keeps M2 off even after M1 turns on. The analysis shows that the forward trigger voltage is given by



(a) Symbol

(b) Voltage transfer curve

Figure 14.32 An inverting Schmitt trigger

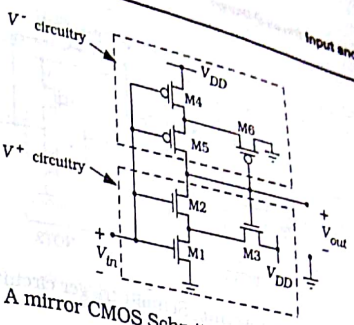


Figure 14.33 A mirror CMOS Schmitt trigger

$$V^+ = \frac{V_{DD} + \sqrt{\frac{\beta_1}{\beta_3}}\, V_{Tn}}{1 + \sqrt{\frac{\beta_1}{\beta_3}}} \qquad (14.67)$$

where the device ratio (β_1/β_3) is the design variable. Since both M1 and M3 are nFETs, this reduces to the ratio of device aspect ratios

$$\frac{\beta_1}{\beta_3} = \frac{(W/L)_1}{(W/L)_3} \qquad (14.68)$$

In the same manner, M6 is the feedback transistor for the pFET group. The reverse trigger voltage is found from

$$V^- = \frac{\sqrt{\frac{\beta_4}{\beta_6}}(V_{DD} - |V_{Tp}|)}{1 + \sqrt{\frac{\beta_4}{\beta_6}}} \qquad (14.69)$$

where

$$\frac{\beta_4}{\beta_6} = \frac{(W/L)_4}{(W/L)_6} \qquad (14.70$$

is the pFET ratio. The ratioed characteristic of the circuit can result in design using relatively large FETs. This is because the series-connecte transistors must be made large to compensate for the resistance while t switching voltages are set by the sizes selected for M3 and M6.

A non-inverting Schmitt trigger circuit is shown in Figure 14.34. Tra sistors Mp1 and Mn1 are used respectively as weak pull-up and pu down devices that are controlled by the output voltage V_{out} through feedback connection. Suppose that $V_{in} = 0$ V. The output of the inverter (NOT1) is high, so that $V_{out} = 0$ V from the second inverter (NO

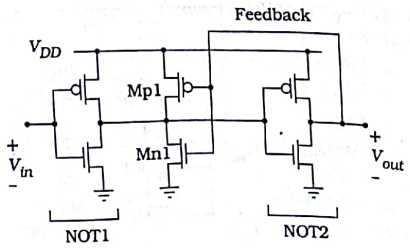


Figure 14.34 A non-inverting Schmitt trigger circuit

This biases Mp1 on and Mn1 off. If we increase V_{in}, the output node of NOT1 is held high by Mp1, which delays the switching. The aspect ratios of Mp1 and Mn1 must be small so as to still allow switching to take place.

14.6.2 Output Drivers

Output circuits must drive the pad capacitance along with the external load connected to the pin. Equation (14.48) may be used to find the capacitance C_{pad} of the pad, but the off-chip load varies with the application. A typical design value is around 80 pF, which is about the loading presented by a test probe. Since this is much larger than the femtofarad levels encountered in normal on-chip design, we must use large output transistors to maintain high speeds.

Example 14.3

Suppose that a 0.5 µm CMOS process uses I/O pads on a Metal3 layer that is characterized by a capacitance of 14 aF/µm^2. The unit aF stands for attofarad such that 1 aF = 10^{-18} F.

If we use pads that have dimensions of 75 µm × 75 µm, the pad capacitance is

$$C_{pad} = (14)(75^2) = 78.75 \text{ fF} \tag{14.71}$$

that must be added to the external capacitance contributions.

Off-chip driver design is a critical aspect of CMOS VLSI design. We have seen that on-chip switching times of simple logic gates are in the sub-nanosecond range. Transferring high on-chip data rates to the outside world is complicated by the large capacitance values. Scaled driver chains as discussed in Section 8.3 can be used to deal with the problem. Figure 14.35 shows a 4-stage output circuit that must drive a large picofarad-level load capacitor C_L. Theoretically, the delay minimization analysis specifies the number of stages N in the chain as

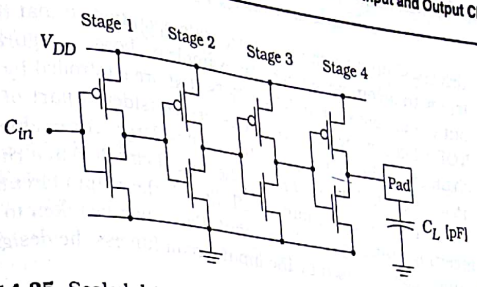


Figure 14.35 Scaled driver chain output circuit

$$N = \frac{\ln\left(\frac{C_L}{C_{in}}\right)}{\ln(S)} \tag{14.72}$$

where S is the scaling factor. However, large output capacitances may lead to large N-values and aspect ratios, so it is sometimes more practical to look at the output switching times requirements. In the example, the output characteristics are determined by Stage 4 since it drives C_L. If we specify values for the rise and fall times t_r and t_f, then the time constant expressions can be used to write

$$t_r = 2.2R_pC_L$$
$$t_f = 2.2R_nC_L \tag{14.73}$$

for the fourth stage. Once the resistances are known, the aspect ratios for can be calculated from

$$\left(\frac{W}{L}\right)_{p,4} = \frac{1}{k'_p(V_{DD}-|V_{Tp}|)R_p}$$
$$\left(\frac{W}{L}\right)_{n,4} = \frac{1}{k'_n(V_{DD}-V_{Tn})R_n} \tag{14.74}$$

The input capacitance into Stage 4 is

$$C_4 = C_{ox}[(WL)_{n4}+(WL)_{p4}] \tag{14.75}$$

which is taken to be the output capacitance seen by Stage 3. Each stag can be designed using the same rise and fall time values, working fro the output side toward the interior circuitry. This is repeated until th input capacitance C_{in} is to a "normal" level, which determines the numbe of stages. Equalizing the stage delays is equivalent to using linear scalin

A bi-directional pad provides circuitry for both input and output sig nals. The input circuits are identical to those described above Outpu

drivers should be capable of tri-state operation so that they do not interfere with incoming signals. An example is shown in Figure 14.36. The output circuit uses large driver FETs that are controlled by the NAND2 and NOR2 logic network. The gates are considered part of a scaled driver chain since the FET capacitances will be large. The enable signal En is the tri-state control. When $En = 0$, the output circuit is in a Hi-Z state and the pad can be used for inputs. With $En = 1$, the output circuit acts as a non-inverting buffer for the Data input. Care must be taken to insure that the output signal is used by the input circuit (unless the design requires it).

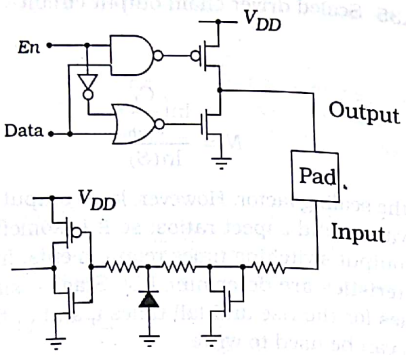


Figure 14.36 A bi-directional I/O circuit

14.7 Power Distribution and Consumption

The power supply values V_{DD} and V_{SS} are externally applied sources that enter the chip environment via two separate pads. The **power supply distribution grid** is a set of metal lines that provide the voltages to every part of the circuit. It must be designed with a geometrical patterning that allows a high packing density while providing the necessary current flow levels.

Two electrical problems tend to dominate the design of power supply grids. The first is electromigration where metal atoms are moved from one end to the other when the current density J [A/cm^2] is large. Since the total current I in amperes is related to the cross-sectional area tw by

$$I = J(tw) \tag{14.76}$$

we can increase the line width w to insure that J remains at acceptable levels. This is usually specified by a design rule table that provides the minimum width w for different ranges of currents. The second problem is

the resistance R_{line} of the line. By Ohm's law, the voltage drop across the line is

$$V_{line} = IR_{line} \tag{14.77}$$

so that the voltage that reaches the circuits is altered by this amount. Linewidths, routing, and via placement all contribute to the total resistance between two points. Both problems can be solved by using wide lines, but this brute-force solution consumes excessive area.

Tree-like structures are the most common approach to designing the distribution scheme. The general idea is portrayed in Figure 14.37. The primary V_{DD} line is designed to have a width large enough to carry the total current I for the entire circuit. This is fed into branches, each carrying an average current of I_1 such that

$$I = N_1 I_1 \tag{14.78}$$

where N_1 is the number of secondary lines. Each secondary line feeds into ternary lines that carry a current I_3 and so on, until the individual logic cells are powered. The widths can be calculated once the values of the currents are known. Since digital CMOS circuits have current requirements that vary in time, average values are used to find the widths. Transient characteristics may require widening some lines.

Realistic power grids are designed by routing supply lines and then strapping them together to form a power mesh. Power buses from the pad are usually isolated from those applied to the cells in order to reduce noise problems. The idea can be understood from the drawing in Figure 14.38 where a signal line is placed between two VSS (ground) lines to provide electrical shielding. From the physical viewpoint, isolation is

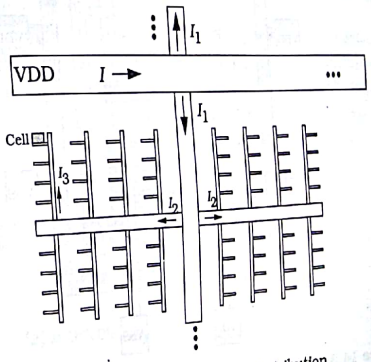


Figure 14.37 Linewidth sizing for power distribution

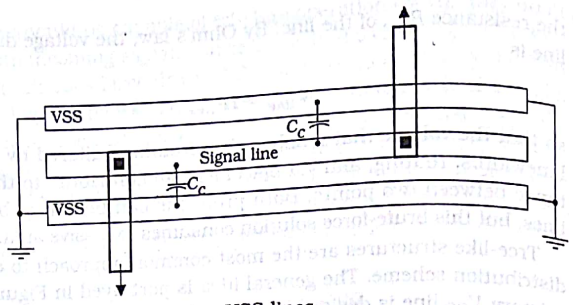


Figure 14.38 Isolation using VSS lines

achieved because coupling capacitances C_c are connected to (relatively) noise-free VSS lines. This can be applied to the 2-metal distribution model presented schematically in Figure 14.39. The power supply voltages VDD and VSS are used to form power rings around the interior circuit regions. The VSS ring acts as an electrical shield for the cells. Metal widths are sized according to the current draw from the logic circuits. Cells are placed between the smallest width VDD and VSS rails.

VLSI design libraries usually have pad frames cells that are used to power the chip, so the design centers on the interior regions. Electrical design rules for widths and loading levels can be applied on a line-by-line basis in critical circuits, but the sheer numbers involved usually mandate an algorithmic approach based on the architecture and circuit design.

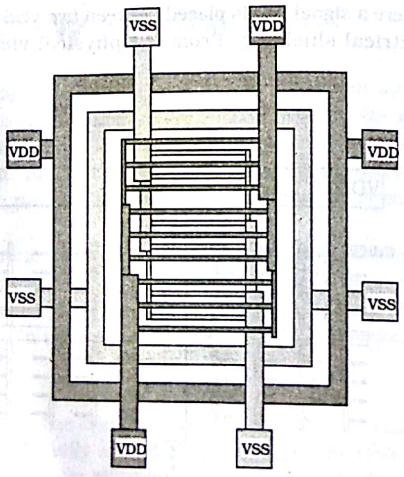


Figure 14.39 Power distribution scheme

14.7.1 Simultaneous Switching Noise

Simultaneous switching noise (SSN) is a fluctuation of the line voltage that is caused by many simultaneous switching events. The effect is also called **delta-I** (ΔI) and **ground-bounce** in the literature. SSN occurs when the current draw on a line changes very rapidly in time. Since high-speed digital networks are designed for fast switching, SSN problems are common in VLSI.

All conductive lines exhibit some **inductance** L that accounts for the magnetic energy storage property of the current flow. Although we usually ignore inductive effects in interconnects that transmit low current signals, they are important to power distribution lines that feed high-speed circuits. The basic problem can be understood from basic electromagnetic theory. An electrical current i produces a magnetic field which gives a magnetic flux Φ such that

$$\Phi = Li \tag{14.79}$$

The inductance L has units of Henrys [H] and has been introduced as a constant of proportionality. Figure 14.40(a) shows the physical model. Faraday's Law of Induction states that a time-varying magnetic flux $\Phi(t)$ induces a voltage $v(t)$ according to

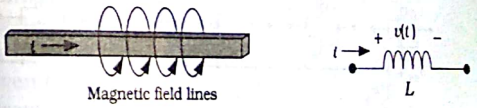
$$v(t) = \frac{d\Phi}{dt} \tag{14.80}$$

Substituting for the flux gives the I-V relation

$$v(t) = L\frac{di}{dt} \tag{14.81}$$

for an inductor. The inductor symbol shown in Figure 14.40(b). This equation shows that the induced voltage is proportional to the time rate of change of the current di/dt.

The problems of SSN can be understood by reviewing the switching characteristics of the simple inverter illustrated in Figure 14.41. When the input voltage $V_{in} = 0$, the nFET is off and $i \approx 0$; the zero-current situation also holds for $V_{in} = V_{DD}$ since the pFET is off. Direct current flow from



(a) Interconnect line (b) Equivalent element

Figure 14.40 Origin of line inductance

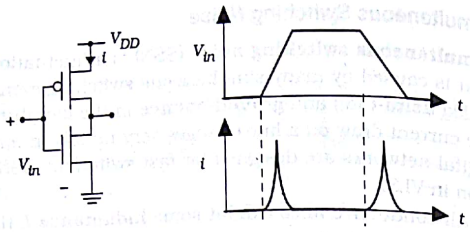


Figure 14.41 Current flow in an inverter circuit

V_{in} to ground occurs only when both transistors are conducting during a change in V_{in}. The plot shows (di/dt) has both positive and negative values. If the input voltage V_{in} can be changed at a faster rate (i.e., a steeper slope), the current derivatives increase.

Consider now the case shown in Figure 14.42 where a common power supply rail is used for a group of circuits. If every gate is switched at the same time, then the current i_1 through the parasitic inductance L_1 is computed from

$$i_1(t) = \sum_j i_j(t) \tag{14.82}$$

where the sum is over the entire collection of gates. The voltage across the inductor is

$$v_1(t) = L_1\left(\frac{di_1}{dt}\right) = L_1\sum_j\left(\frac{di_j}{dt}\right) \tag{14.83}$$

so that the actual value v_{dd} of the voltage applied to the pFETs is

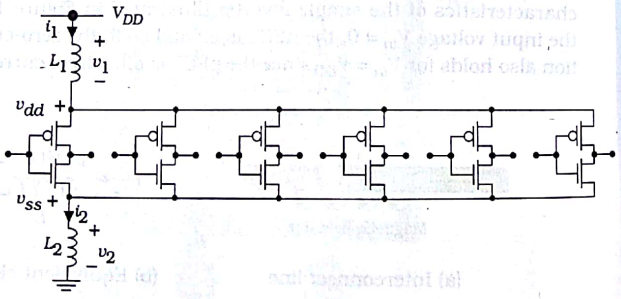


Figure 14.42 Simultaneous switching noise example

$$v_{dd}(t) = V_{DD} - v_1(t)$$
$$= V_{DD} - L_1\sum_j\left(\frac{di_j}{dt}\right) \tag{14.84}$$

The voltage v_2 across L_2 is calculated in the same manner so that

$$v_{ss}(t) = v_2(t) = L_2\sum_k\left(\frac{di_k}{dt}\right) \tag{14.85}$$

is the effective voltage applied to the nFET source terminals. In general, i_1 and i_2 are not equal because of current flow into, or out of, the output nodes of the gates. Note that the derivatives can be positive or negative depending upon the transition. A more complete analysis usually employs the circuit model in Figure 14.43. This adds drain and source resistances R_d and R_s, in addition to line capacitances C_1 and C_2 to obtain higher accuracy in the waveform simulations.

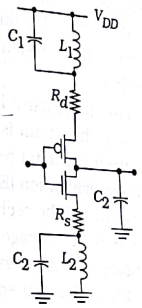


Figure 14.43 Complete circuit model

The main result of this analysis is that the instantaneous values of $v_{dd}(t)$ and $v_{ss}(t)$ establish the output voltages of the logic gates, not the DC values V_{DD} and 0 V. The modified values of the logic voltage levels may cause errors in the logic. Since the total current is a sum of the individual gate contributions, even moderate switching speeds will cause problems if many gates are switched at the same time. A histogram plot such as that shown in Figure 14.44 shows the number of gates that are switching at any given time. The data is usually weighted according to the peak current draw for each gate. Periods of high activity where a large number of gates are changing are of particular concern. The ideal situation is where the number of gates changing at any given time is a constant, since this would allow steady-state voltages to be achieved.

Predicting gate activity is a complicated task due to the complexity of

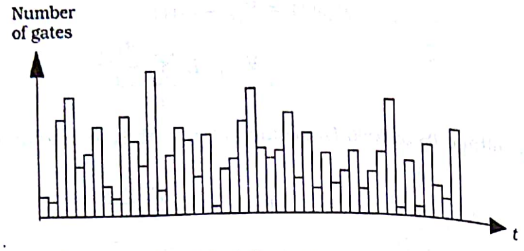


Figure 14.44 Gate switching distribution

modern VLSI systems. Moreover, the circuit design style and clocking have a direct effect on the results. The simplest case of a cascaded chain of random logic gates would be described by the current distribution shown in Figure 14.45. This represents the case where the inputs to the gate 1 logic group are switched at time t_0 and ripple down the chain causing a switch in every group. The spacing in between the current spikes can be estimated as the gate delay times. This type of consideration can be used in the logic and circuit design stage to equalize the current draw level as a function of time.

The dynamic logic circuit of Figure 14.46 is more predictable due to the clock control. When $\phi = 0$, the circuit is in precharge (P) and charging current i_{ch} flows to the output capacitance C_{out}. When the clock changes to $\phi = 1$, the circuit undergoes evaluation (E). If a logic 0 is produced, i_{dis} flows to ground and the output will be recharged during the next precharge interval. If the charge is held, leakage will occur and the stage will still require partial recharging. The discharge current flow takes place during the evaluation phase and is distributed according to the location of each stage in the logic cascade.

SSN levels are also dependent on the packaging and the wiring technique used to connect the die to the pins. In multi-chip modules (MCMs), SSN can cause unwanted interactions among different chips. These and other related problems are the center of many research projects.

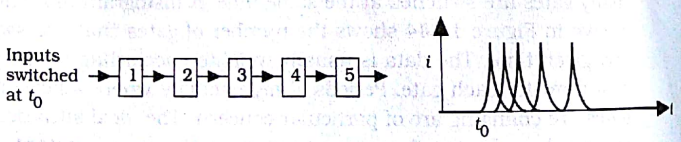


Figure 14.45 Switching current in a random logic chain

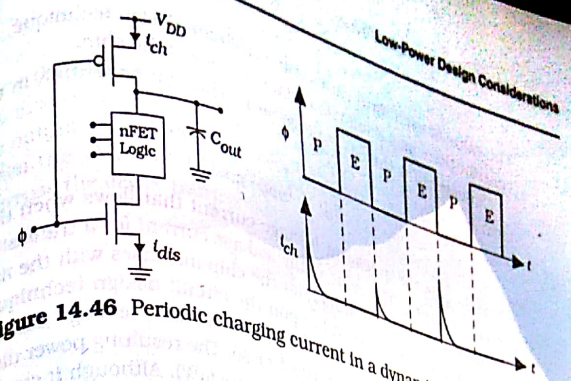


Figure 14.46 Periodic charging current in a dynamic logic gate

14.8 Low-Power Design Considerations

The overall power dissipation P [W] of the chip is critically important in modern VLSI. If a battery power supply is used, then P determines the operating time before a recharge is needed. Even on a desktop system, the power dissipation must be kept low to insure that the silicon doesn't melt (the worst-case situation) and the system cooling scheme is sufficient. Low-power design techniques have been developed at both the circuit and system level. Substantial amounts of research are devoted to studying the problem and solutions.

There are three main sources of power dissipation in a digital CMOS circuit.

- DC power P_{DC} that is due to direct conduction paths from V_{DD} to ground when inputs are stable. Leakage currents are the cause of this component in standard static CMOS logic circuits.
- Switching power P_{sw} that is dissipated when an input change causes the power supply to have a direct current flow path to ground through the transistors. This occurs during the transition portion of a voltage transfer curve (VTC) and was discussed in the previous section as the origin of SSN problems.
- Dynamic switching power P_{dyn} due to charging and discharging capacitive nodes. This is estimated from the general formula

$$P_{dyn} = aCV^2 f \tag{14.86}$$

where C is the capacitance in farads, a is the activity coefficient, V is the voltage swing, and f the signal frequency.

The instantaneous total power dissipation is the sum

$$P = P_{DC} + P_{sw} + P_{dyn} \tag{14.87}$$

The value of each term varies with the circuit design technique, and some contributions may dominate certain sections of the chip.

Consider first the DC leakage term. This can be written in simplified form as

$$P_{DC} = I_{DDQ}V_{DD} \tag{14.88}$$

where I_{DDQ} is the quiescent leakage current that flows when the inputs are not changing. The value of the leakage current for a transistor is process-dependent. The total I_{DDQ} for the chip increases with the number of transistors and also depends upon the circuit design technique. Static CMOS logic gates exhibit the smallest quiescent leakage currents with chip values usually less than 10 μA or so. The resulting power dissipation is on the order of a few tens of microwatts (μW). Although it can be larger in certain designs, it is usually the smallest among the three.

Switching power P_{sw} is a consequence of a gate input signal transition causing a direct current flow path from V_{DD} to ground and is the origin of SSN. It occurs every time the output voltage undergoes a voltage transition, and originates with the circuit design. Static logic gates always dissipate switching power since the conduction path cannot be eliminated. A simple estimate is

$$P_{sw} = \langle I_{sw} \rangle V_{DD} \tag{14.89}$$

where $\langle I_{sw} \rangle$ is the average DC current flow. The contribution from an isolated gate varies with the transistor aspect ratios, since (W/L) determines the current flow level through a FET. The actual magnitude depends upon the shape of the input waveform, making it difficult to calculate using closed-form equations. Circuit simulations are the most accurate.

The dynamic power dissipation is usually considered to be the most difficult to deal with. The general expression

$$P_{dyn} = aCV^2f \tag{14.90}$$

shows that P_{dyn} increases proportionately with the signal switching frequency f so that it grows with the speed of the circuit. One approach to decreasing the magnitude of this term is to reduce the power supply voltage V_{DD} since it is the maximum (DC) value for V. This also reduces the values of the other contributions. Processor core voltages are currently below 2 V, and the push is on for even lower operating voltages. A reduced power supply voltage is also advantageous in battery-operated units.

Although this may seem like a simple technique, it introduces problems at the circuit level that leads to slower switching speed. This, of course, defeats the purpose of decreasing V_{DD} in the first place. To understand this statement, recall that a non-saturated FET has a current of

$$I_D = \frac{\beta}{2}[2(V_{GS}-V_T)V_{DS}-V_{DS}^2] \tag{14.91}$$

Since the highest voltage in the circuit is V_{DD}, reducing it implies that I_D will also decrease. Reducing I_D implies that it will take longer to charge the output capacitance, increasing both the rise and fall times. This slows down the gate switching speed. To compensate for this effect we can increase the device transconductance term

$$\beta = \mu_n C_{ox}\left(\frac{W}{L}\right) \tag{14.92}$$

Since

$$C_{ox} = \frac{\varepsilon_{ox}}{t_{ox}} \tag{14.93}$$

shrinking the gate oxide thickness t_{ox} increases β so improved processing helps. Otherwise, we must increase the channel width W to maintain the speed.

Many novel and unique approaches for reducing the power dissipation of VLSI chips have been published in the literature. The problem itself is usually tackled at both the circuit design and the architectural level. The interested reader is referred to the literature. Several books on the subject have been listed in the reference section.

14.9 References for Further Study

[1] Abdellatif Bellaouar and Mohamed I. Elmasry, **Low-Power Digital VLSI Design**, Kluwer Academic Publishers, Norwell, MA, 1995.

[2] Anantha P. Chandrakasan and Robert W. Brodersen, **Low Power Digital CMOS Design**, Kluwer Academic Publishers, Norwell, MA, 1995.

[3] Dan Clein, **CMOS IC Layout**, Newnes, Woburn, MA, 2000.

[4] Sabih H. Gerez, **Algorithms for VLSI Design Automation**, John Wiley & Sons, Chichester, England, 1999.

[5] Bryan Preas and Michael Lorenzetti (eds.), **Physical Design Automation of VLSI Systems**, Benjamin-Cummings Publishing Company, Menlo Park, CA, 1988.

[6] Jan M. Rabaey, **Digital Integrated Circuits**, Prentice Hall, Upper Saddle River, NJ, 1996.

[7] Jan M. Rabaey and Massoud Pedram, **Low Power Design Methodologies**, Kluwer Academic Publishers, Norwell, MA, 1996.

[8] Michael Reed and Ron Rohrer, **Applied Introductory Circuit Analysis**, Prentice Hall, Upper Saddle River, NJ, 1999.

[9] Kaushik Roy and Sharat C. Prasad, **Low-Power CMOS VLSI Circuit Design**, John Wiley & Sons, New York, 2000.