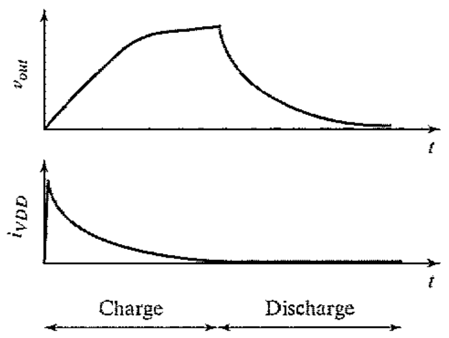


Figure 5-26 Output voltages and supply current during (dis)charge of C_L.

and

$$E_C = \int_0^{\infty} i_{VDD}(t)v_{out}dt = \int_0^{\infty} C_L\frac{dv_{out}}{dt}v_{out}dt = C_L\int_0^{V_{DD}} v_{out}dv_{out} = \frac{C_LV_{DD}^2}{2} \tag{5.41}$$

The corresponding waveforms of $v_{out}(t)$ and $i_{VDD}(t)$ are pictured in Figure 5-26.

These results can also be derived by observing that during the low-to-high transition, C_L is loaded with a charge C_LV_{DD}. Providing this charge requires an energy from the supply equal to $C_LV_{DD}^2$ ($= Q \times V_{DD}$). The energy stored on the capacitor equals $C_LV_{DD}^2/2$. This means that only half of the energy supplied by the power source is stored on C_L. The other half has been dissipated by the PMOS transistor. Notice that this energy dissipation is independent of the size (and hence the resistance) of the PMOS device! During the discharge phase, the charge is removed from the capacitor, and its energy is dissipated in the NMOS device. Once again, there is no dependence on the size of the device. In summary, each switching cycle (consisting of an L→H and an H→L transition) takes a fixed amount of energy, equal to $C_LV_{DD}^2$. In order to compute the power consumption, we have to take into account how often the device is switched. If the gate is switched **on and off** $f_{0\rightarrow1}$ times per second, the power consumption is given by

$$P_{dyn} = C_LV_{DD}^2f_{0\rightarrow1} \tag{5.42}$$

where $f_{0\rightarrow1}$ represents the frequency of energy-consuming transitions (these are $0 \rightarrow 1$ transitions for static CMOS).

Advances in technology result in ever-higher values of $f_{0\rightarrow1}$ (as t_p decreases). At the same time, the total capacitance on the chip (C_L) increases as more and more gates are placed on a single die. Consider, for instance, a 0.25-μm CMOS chip with a clock rate of 500 MHz and an average load capacitance of 15 fF/gate, assuming a fan-out of 4. The power consumption per gate for a 2.5-V supply then equals approximately 50 μW. For a design with 1 million gates, and assuming that a transition occurs at every clock edge, this would result in a power consumption of 50 W! This evaluation, fortunately, presents a pessimistic perspective. In reality, not all gates in the complete IC switch at the full rate of 500 Mhz. The actual activity in the circuit is substantially lower.

Example 5.11 Capacitive Power Dissipation of Inverter

The capacitive dissipation of the CMOS inverter of Example 5.4 is now easily computed. In Table 5-2, the value of the load capacitance was determined to equal 6 fF. For a supply voltage of 2.5 V, the amount of energy needed to charge and discharge that capacitance equals

$$E_{dyn} = C_L V_{DD}^2 = 37.5 \text{ fJ}$$

Assume that the inverter is switched at the (hypothetically) maximum possible rate ($T = 1/f = t_{pLH} + t_{pHL} = 2t_p$). For a t_p of 32.5 ps (Example 5.5), we find that the dynamic power dissipation of the circuit is

$$P_{dyn} = E_{dyn}/(2t_p) = 580 \text{ }\mu\text{W}$$

Of course, an inverter in an actual circuit is rarely switched at this maximum rate, and even if it would be, the output does not swing from rail to rail. The power dissipation will thus be substantially lower. For a rate of 4 GHz (T = 250 ps), the dissipation reduces to 150 μW. This is confirmed by simulations, which yield a power consumption of 155 μW.

Computing the dissipation of a complex circuit is complicated by the $f_{0\rightarrow1}$ factor, also called the *switching activity*. While the switching activity is easily computed for an inverter, it turns out to be far more complex in the case of more complex gates and circuits. One concern is that the switching activity of a network is a function of the nature and the statistics of the input signals: If the input signals remain unchanged, no switching happens, and the dynamic power consumption is zero! On the other hand, rapidly changing signals provoke plenty of switching and therefore dissipation. Other factors influencing the activity are the overall network topology and the function to be implemented. We can accommodate this by writing

$$P_{dyn} = C_L V_{DD}^2 f_{0\rightarrow1} = C_L V_{DD}^2 P_{0\rightarrow1} f = C_{EFF} V_{DD}^2 f \tag{5.43}$$

where f now presents the maximum possible event rate of the inputs (which is often the clock rate) and $P_{0\rightarrow1}$ the probability that a clock event results in a $0 \rightarrow 1$ (or power-consuming) event at the output of the gate. $C_{EFF} = P_{0\rightarrow1}C_L$ is called the *effective capacitance* and represents the average capacitance switched every clock cycle. For our example, an activity factor of 10% ($P_{0\rightarrow1} = 0.1$) reduces the average consumption to 5 W.

Example 5.12 Switching Activity

Consider the waveforms in Figure 5.27, where the upper waveform represents the idealized clock signal, and the bottom one shows the signal at the output of the gate. Power consuming transitions occur 2 out of 8 times, which is equivalent to a transition probability of 0.25 (or 25%).

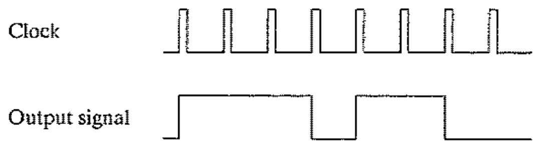


Figure 5-27 Clock and signal waveforms.

Low Energy–Power Design Techniques

With the increasing complexity of digital integrated circuits, it is anticipated that the power problem will only worsen in future technologies. This is one of the reasons that lower supply voltages are becoming more and more attractive. **Reducing V_{DD} has a quadratic effect on P_{dyn}**. For instance, reducing V_{DD} from 2.5 V to 1.25 V for our example drops the power dissipation from 5 W to 1.25 W. This assumes that the same clock rate can be sustained. Figure 5-17 demonstrates that this assumption is not that unrealistic as long as the supply voltage is substantially higher than the threshold voltage. A large performance penalty occurs once V_{DD} approaches 2 V_T.

When a lower limit on the supply voltage is set by external constraints (as often happens in real-world designs), or when the performance degradation due to lowering the supply voltage is intolerable, the only means of reducing the dissipation is by lowering the effective capacitance. This can be achieved by addressing both of its components: the physical capacitance and the switching activity.

A *reduction in the switching activity* can only be accomplished at the logic and architectural abstraction levels, and will be discussed in more detail in Chapter 11. *Lowering the physical capacitance* is a worthwhile goal overall, and it also may help to improve the performance of the circuit. As most of the capacitance in a combinational logic circuit is due to transistor capacitances (gate and diffusion), it makes sense to keep those contributions to a minimum when designing for low power. This means that transistors should be kept to *minimal size* whenever possible or reasonable. This definitely affects the performance of the circuit, but the effect can be offset by using logic or architectural speedup techniques. The only instances where transistors should be sized up is when the load capacitance is dominated by extrinsic capacitances (such as fan-out or wiring capacitance). This is contrary to common design practices used in cell libraries, where transistors are generally made large to accommodate a range of loading and performance requirements.

These observations lead to an interesting design challenge. Assume we have to minimize the energy dissipation of a circuit with a specified lower bound on the performance. An attractive approach is to lower the supply voltage as much as possible, and to compensate the loss in performance by increasing the transistor sizes. Yet, the latter causes the capacitance to increase. It may be foreseen that at a low enough supply voltage, the latter factor may start to dominate and cause energy to increase with a further drop in the supply voltage. ■

Example 5.13 Transistor Sizing for Energy Minimization

To analyze the transistor sizing for a minimum energy problem, we examine the simple case of a static CMOS inverter driving an external load capacitance C_{ext}, as in Figure 5.28. To take the input loading effects into account, we assume that the inverter itself is driven by a minimum-sized device. The goal is to minimize the energy dissipation of the complete circuit, while maintaining a lower bound on performance. The degrees of freedom are the size factor f of the inverter and the supply voltage V_{dd} of the circuit. The propagation delay of the optimized circuit should not be larger than that of a reference circuit, chosen to have as parameters $f = 1$ and $V_{dd} = V_{ref}$.

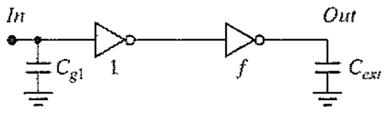


Figure 5-28 CMOS inverter driving an external load capacitance C_{ext}, while being driven by a minimum sized gate.

Using the approach introduced in Section 5.4.3 (*Sizing a Chain of Inverters*), we can derive the following expression for the propagation delay of the circuit:

$$t_p = t_{p0}\left(\left(1 + \frac{f}{\gamma}\right) + \left(1 + \frac{F}{f\gamma}\right)\right) \tag{5.44}$$

here $F = (C_{ext}/C_{g1})$ is the overall effective fan-out of the circuit, and t_{p0} is the intrinsic delay of the inverter. Its dependence upon V_{DD} is approximated by the following expression, derived from Eq. (5.21):

$$t_{p0} \sim \frac{V_{DD}}{V_{DD} - V_{TE}} \tag{5.45}$$

The energy dissipation for a single transition at the input is easily found once the total capacitance of the circuit is known:

$$E = V_{dd}^2 C_{g1}((1 + \gamma)(1 + f) + F) \tag{5.46}$$

The performance constraint now states that the propagation delay of the scaled circuit should be equal (or smaller) to the delay of the reference circuit ($f = 1$, $V_{dd} = V_{ref}$). To simplify the subsequent analysis, we make the assumption that the intrinsic output capacitance of the gate equals its gate capacitance, or $\gamma = 1$. Hence,

$$\frac{t_p}{t_{pref}} = \frac{t_{p0}\left(2 + f + \frac{F}{f}\right)}{t_{p0ref}(3 + F)} = \left(\frac{V_{DD}}{V_{ref}}\right)\left(\frac{V_{ref} - V_{TE}}{V_{DD} - V_{TE}}\right)\left(\frac{2 + f + \frac{F}{f}}{3 + F}\right) = 1 \tag{5.47}$$

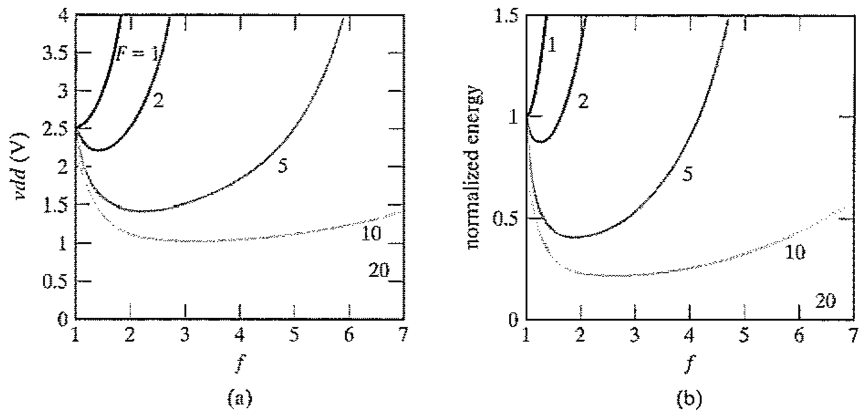


Figure 5-29 Sizing of an inverter for energy minimization. (a) Required supply voltage as a function of the sizing factor *f for different values of the overall effective fan-out F*; (b) Energy of scaled circuit (normalized with respect to the reference case) as a function of *f*. $V_{ref} = 2.5$ V, $V_{TE} = 0.5$ V.

Equation (5.47) establishes a relationship between the sizing factor f and the supply voltage, plotted in Figure 5-29a for different values of F. Those curves show a clear minimum. Increasing the size of the inverter from the minimum initially increases the performance, and hence allows for a lowering of the supply voltage. This is fruitful until the optimum sizing factor of $f = \sqrt{F}$ is reached, which should not surprise those who read the previous sections carefully. Further increases in the device sizes only increase the self-loading factor, deteriorate the performance, and require an increase in supply voltage. Also, observe that for the case of $F = 1$, the reference case is the best solution; any resizing just increases the self-loading.

With the $V_{DD}(f)$ relationship in hand, we can derive the energy of the scaled circuit (normalized with respect to the reference circuit) as a function of the sizing factor f:

$$\frac{E}{E_{ref}} = \left(\frac{V_{DD}}{V_{ref}}\right)^2\left(\frac{2 + 2f + F}{4 + F}\right) \tag{5.48}$$

Finding an analytical expression for the optimal sizing factor is possible, but yields a complex and messy equation. A graphical approach is just as effective. The resulting charts are plotted in Figure 5-29b, from which a number of conclusions can be drawn:[6]

- **Device sizing, combined with supply voltage reduction, is a very effective approach in reducing the energy consumption of a logic network.** This is especially true for networks with large effective fan-outs, where energy reductions with almost a factor of 10 can be obtained. The gain is also sizable for smaller values of F. The only exception is the $F = 1$ case, where the minimum size device is also the most effective one.

[6]We will revisit some of these conclusions in Chapter 11 in a broader context.

- Oversizing the transistors beyond the optimal value comes at a hefty price in energy. This is, unfortunately, a common approach in many of today's designs.
- The optimal sizing factor for energy is smaller than the one for performance, especially for large values of F. For example, for a fan-out of 20, f_{opt}(energy) = 3.53, while f_{opt}(performance) = 4.47. Increasing the device sizes only leads to a minimal supply reduction once V_{DD} starts approaching V_{TE}, thus leading to very minimal energy gains.

Dissipation Due to Direct-Path Currents

In actual designs, the assumption of the zero rise and fall times of the input wave forms is not correct. The finite slope of the input signal causes a direct current path between V_{DD} and GND for a short period of time during switching, while the NMOS and the PMOS transistors are conducting simultaneously. This is illustrated in Figure 5-30. Under the (reasonable) assumption that the resulting current spikes can be approximated as triangles and that the inverter is symmetrical in its rising and falling responses, we can compute the energy consumed per switching period as follows:

$$E_{dp} = V_{DD}\frac{I_{peak}t_{sc}}{2} + V_{DD}\frac{I_{peak}t_{sc}}{2} = t_{sc}V_{DD}I_{peak} \tag{5.49}$$

We compute the average power consumption as

$$P_{dp} = t_{sc}V_{DD}I_{peak}f = C_{sc}V_{DD}^2 f \tag{5.50}$$

The direct-path power dissipation is proportional to the switching activity, similar to the capacitive power dissipation. t_{sc} represents the time both devices are conducting. For a linear input slope, this time is reasonably well approximated by Eq. (5.51) where t_s represents the 0–100% transition time,

$$t_{sc} = \frac{V_{DD}-2V_T}{V_{DD}}t_s \approx \frac{V_{DD}-2V_T}{V_{DD}} \times \frac{t_{r(f)}}{0.8} \tag{5.51}$$

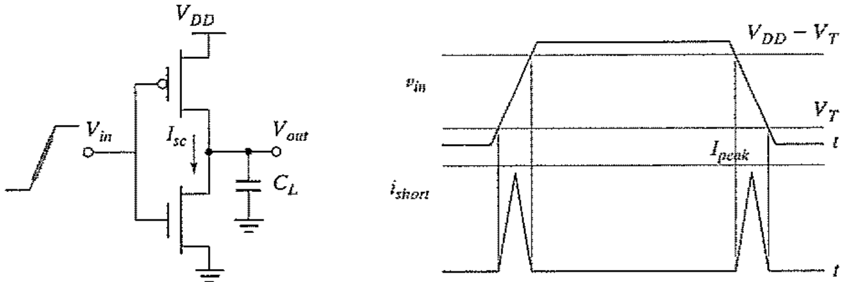
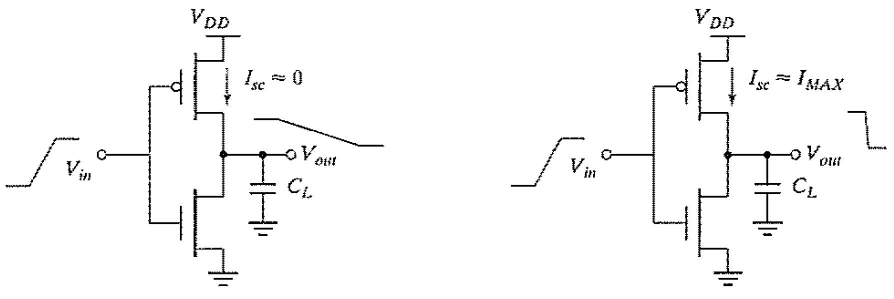


Figure 5-30 Short-circuit currents during transients.

I_{peak} is determined by the saturation current of the devices and is hence directly proportional to the sizes of the transistors. The peak current is also **a strong function of the ratio between input and output slopes.** This relationship is best illustrated by the following simple analysis: Consider a static CMOS inverter with a $0 \rightarrow 1$ transition at the input. Assume first that the load capacitance is very large, so that the output fall time is significantly larger than the input rise time (Figure 5-31a). Under those circumstances, the input moves through the transient region before the output starts to change. As the source-drain voltage of the PMOS device is approximately 0 during that period, the device shuts off without ever delivering any current. The short-circuit current is close to zero in this case. Consider now the reverse case, where the output capacitance is very small, and the output fall time is substantially smaller than the input rise time (Figure 5-31b). The drain-source voltage of the PMOS device equals V_{DD} for most of the transition period, guaranteeing the maximal short-circuit current (equal to the saturation current of the PMOS). This clearly represents the worst case condition. The conclusions of the preceding analysis are confirmed in Figure 5-32, which plots the short-circuit current through the NMOS transistor during a low-to-high transition as a function of the load capacitance.



(a) Large capacitive load (b) Small capacitive load

Figure 5-31 Impact of load capacitance on short-circuit current.

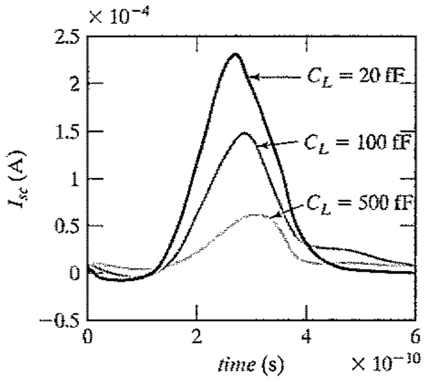


Figure 5-32 CMOS inverter short-circuit current through NMOS transistor as a function of the load capacitance (for a fixed input slope of 500 ps).

This analysis leads to the conclusion that the short-circuit dissipation is minimized by making the output rise/fall time larger than the input ris/fall time. On the other hand, making the output rise/fall time too large slows down the circuit and can cause short-circuit currents in the fan-out gates. This presents a perfect example of how local optimization and forgetting the global picture can lead to an inferior solution.

Design Techniques

A more practical rule, which optimizes the power consumption in a global way, can be formulated ([Veendrick84]):

> The power dissipation due to short-circuit currents is minimized by matching the rise/fall times of the input and output signals. At the overall circuit level, this means that rise/fall times of all signals should be kept constant within a range.

Making the input and output rise times of a gate identical is not the optimum solution for that particular gate on its own, but keeps the overall short-circuit current within bounds. This is shown in Figure 5-33, which plots the short-circuit energy dissipation of an inverter (normalized with respect to the zero-input rise time dissipation) as a function of the ratio r between input and output rise/fall times. When the load capacitance is too small for a given inverter size ($r > 2...3$ for $V_{DD} = 5$ V), the power is dominated by the short-circuit current. For very large capacitance values, all power dissipation is devoted to charging and discharging the load capacitance. When the rise/fall times of inputs and outputs are equalized, most power dissipation is associated with the dynamic power, and only a minor fraction (< 10%) is devoted to short-circuit currents.

Observe also that the impact of **short-circuit current is reduced when we lower the supply voltage**, as is apparent from Eq. (5.51). In the extreme case, when $V_{DD} < V_{Tn} + |V_{Tp}|$, short-circuit dissipation is completely eliminated, because both devices are never on simultaneously. With threshold voltages scaling at a slower rate than the supply voltage, short-circuit power dissipation is becoming less important

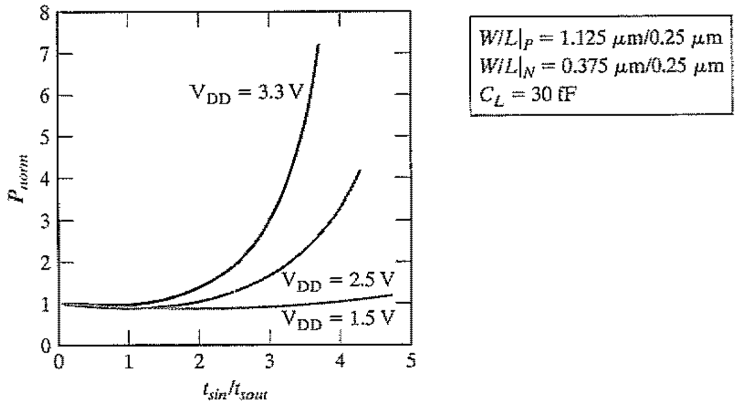


Figure 5-33 Power dissipation of a static CMOS inverter as a function of the ratio between input and output rise/fall times. The power is normalized with respect to zero input rise-time dissipation. At low values of the slope ratio, input/output coupling leads to some extra dissipation.

in deep submicron technologies. At a supply voltage of 2.5 V and thresholds around 0.5 V, an input/output slope ratio of 2 is needed to cause a 10% degradation in dissipation. ■

Finally, it is worth observing that the short-circuit power dissipation can be modeled by adding a load capacitance $C_{sc} = t_{sc}I_{peak}/V_{DD}$ in parallel with C_L, as is apparent in Eq. (5.50). The value of this short-circuit capacitance is a function of V_{DD}, the transistor sizes, and the input/output slope ratio.

5.5.2 Static Consumption

The static (or steady-state) power dissipation of a circuit is expressed by the relation

$$P_{stat} = I_{stat}V_{DD} \tag{5.52}$$

where I_{stat} is the current that flows between the supply rails in the absence of switching activity.

Ideally, the static current of the CMOS inverter is equal to zero, as the PMOS and NMOS devices are never on simultaneously in steady-state operation. There is, unfortunately, a leakage current flowing through the reverse-biased diode junctions of the transistors, located between the source or drain and the substrate, as shown in Figure 5-34. This contribution is, in general, very small and can be ignored. For the device sizes under consideration, the leakage current per unit drain area typically ranges between 10–100 pA/μm^2 at room temperature. For a die with 1 million gates, each with a drain area of 0.5 μm^2 and operated at a supply voltage of 2.5 V, the worst case power consumption due to diode leakage equals 0.125 mW, which clearly is not much of an issue.

However, be aware that the junction leakage currents are caused by thermally generated carriers. Their value increases with increasing junction temperature, and this occurs in an exponential fashion. At 85°C (a commonly imposed upper bound for junction temperatures in commercial hardware), the leakage currents increase by a factor of 60 over their room-temperature values. Keeping the overall operation temperature of a circuit low is consequently a desirable goal. As the temperature is a strong function of the dissipated heat and its removal mechanisms, this can only be accomplished by limiting the power dissipation of the circuit or by using chip packages that support efficient heat removal.

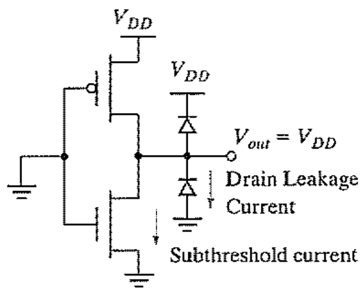


Figure 5-34 Sources of leakage currents in CMOS inverter (for V_{in} = 0 V).

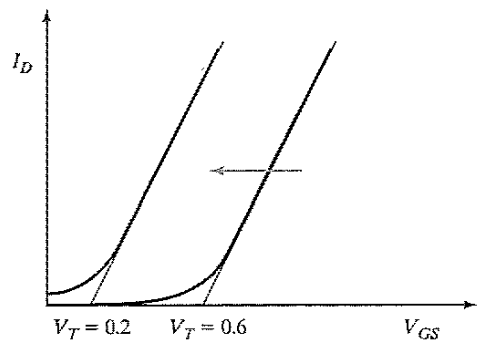


Figure 5-35 Decreasing the threshold increases the subthreshold current at $V_{GS} = 0$.

An emerging source of leakage current is the subthreshold current of the transistors. As discussed in Chapter 3, an MOS transistor can experience a drain-source current, even when V_{GS} is smaller than the threshold voltage (see Figure 5-35). The closer the threshold voltage is to zero volts, the larger the leakage current at $V_{GS} = 0$ V and the larger the static power consumption. To offset this effect, the threshold voltage of the device has generally been kept high enough. Standard processes feature V_T values that are never smaller than 0.5–0.6 V and that in some cases are even substantially higher (~ 0.75 V).

This approach is being challenged by the reduction in supply voltages that typically goes with deep submicron technology scaling, as became apparent in Figure 3-41. We concluded earlier (Figure 5-17) that scaling the supply voltages while keeping the threshold voltage constant results in an important loss in performance, especially when V_{DD} approaches 2 V_T. One approach to address this performance issue is to scale the device thresholds down as well. This moves the curve of Figure 5-17 to the left, which means that the performance penalty for lowering the supply voltage is reduced. Unfortunately, the threshold voltages are lower bounded by the amount of allowable subthreshold leakage current, as demonstrated in Figure 5-35. The choice of the threshold voltage thus represents a trade-off between performance and static power dissipation. The continued scaling of the supply voltage predicted for the next generations of CMOS technologies, however, forces the threshold voltages ever downwards, and makes subthreshold conduction a major source of power dissipation. Process technologies that contain devices with sharper turn-off characteristics will therefore become more attractive. An example of the latter is the SOI (Silicon-on-Insulator) technology whose MOS transistors have slope factors that are close to the ideal 60 mV/decade.

Example 5.14 Impact of Threshold Reduction on Performance and Static Power Dissipation

Consider a minimum size NMOS transistor in the 0.25-μm CMOS technology. In Chapter 3, we derived that the slope factor S for this device equals 90 mV/decade. The off-current (at $V_{GS} = 0$) of the transistor for a V_T of approximately 0.5 V equals 10^{-11} A

(Figure 3-22). Reducing the threshold by 200 mV to 0.3 V multiplies the off-current of the transistors with a factor of 170! Assuming a million gate design with a supply voltage of 1.5 V, this translates into a static power dissipation of $10^6 \times 170 \times 10^{-11} \times 1.5 =$ 2.6 mW. A further reduction of the threshold to 100 mV results in an unacceptable dissipation of almost 0.5 W! At that supply voltage, the threshold reductions correspond to a performance improvement of 25% and 40%, respectively.

This lower bound on the thresholds is in some sense artificial. The idea that the leakage current in a static CMOS circuit has to be zero is a misconception. Certainly, the presence of leakage currents degrades the noise margins, because the logic levels are no longer equal to the supply rails, but as long as the noise margins are within range, this is not a compelling issue. The leakage currents, of course, cause an increase in static power dissipation. This is offset by the drop in supply voltage, which is enabled by the reduced thresholds at no cost in performance, and results in a quadratic reduction in dynamic power. For a 0.25-μm CMOS process, the following circuit configurations obtain the same performance: 3-V supply–0.7-V V_T; and 0.45-V supply–0.1-V V_T. The dynamic power consumption of the latter is, however, 45 times smaller [Liu93]! Choosing the correct values of supply and threshold voltages once again requires a trade-off. The optimal operation point depends upon the activity of the circuit. In the presence of a sizable static power dissipation, it is essential that nonactive modules are *powered down*, lest static power dissipation would become dominant. Power-down (also called *standby*) can be accomplished by disconnecting the unit from the supply rails, or by lowering the supply voltage.

5.5.3 Putting It All Together

The total power consumption of the CMOS inverter is now expressed as the sum of its three components:

$$P_{tot} = P_{dyn} + P_{dp} + P_{stat} = (C_L V_{DD}^2 + V_{DD} I_{peak} t_s) f_{0 \to 1} + V_{DD} I_{leak} \tag{5.53}$$

In typical CMOS circuits, the capacitive dissipation is by far the dominant factor. The direct-path consumption can be kept within bounds by careful design, and thus should not be an issue. Leakage is ignorable at present, but this might change in the not-too-distant future.

The Power-Delay Product, or Energy per Operation

In Chapter 1, we introduced the *power-delay product* (PDP) as a quality measure for a logic gate:

$$PDP = P_{av} t_p \tag{5.54}$$

The *PDP* presents a measure of energy, as is apparent from the units (W × s = Joule). Assuming that the gate is switched at its maximum possible rate of $f_{max} = 1/(2t_p)$, and ignoring the contributions of the static- and direct-path currents to the power consumption, we find that

$$PDP = C_L V_{DD}^2 f_{max} t_p = \frac{C_L V_{DD}^2}{2} \tag{5.55}$$

Dell Ex. 1025
Page 114

Here, *PDP* stands for the **average energy consumed per switching event** (i.e., for a $0 \rightarrow 1$, or a $1 \rightarrow 0$ transition). Remember that earlier we had defined E_{av} as the average energy per switching cycle (or per energy-consuming event). As each inverter cycle contains a $0 \rightarrow 1$, and a $1 \rightarrow 0$ transition E_{av} thus is twice the *PDP*.

Energy-Delay Product

The validity of the *PDP* as a quality metric for a process technology or gate topology is questionable. It measures the energy needed to switch the gate, which is an important property. For a given structure, however, this number can be made arbitrarily low by reducing the supply voltage. From this perspective, the optimum voltage to run the circuit would be the lowest possible value that still ensures functionality. This comes at the expense of performance, as discussed earlier. A more relevant metric should combine a measure of performance and energy. The energy-delay product (or *EDP*) does exactly that:

$$EDP = PDP \times t_p = P_{av} t_p^2 = \frac{C_L V_{DD}^2}{2} t_p \tag{5.56}$$

It is worth analyzing the voltage dependence of the EDP. Higher supply voltages reduce delay, but harm the energy, and the opposite is true for low voltages. An optimum operation point should therefore exist. Assuming that NMOS and PMOS transistors have comparable threshold and saturation voltages, we can simplify the propagation delay expression Eq. (5.21) as

$$t_p \approx \frac{\alpha C_L V_{DD}}{V_{DD} - V_{Te}} \tag{5.57}$$

where $V_{Te} = V_T + V_{DSAT}/2$, and α is a technology parameter. Combining Eq. (5.56) and Eq. (5.57)[7] yields

$$EDP = \frac{\alpha C_L^2 V_{DD}^3}{2(V_{DD} - V_{TE})} \tag{5.58}$$

The optimum supply voltage can be obtained by taking the derivative of Eq. (5.58) with respect to V_{DD}, and equating the result to 0. The result is

$$V_{DDopt} = \frac{3}{2} V_{TE} \tag{5.59}$$

The remarkable outcome from this analysis is the low value of the supply voltage that simultaneously optimizes performance and energy. For submicron technologies with thresholds in the range of 0.5 V, the optimum supply is situated around 1 V.

[7]This equation is only accurate as long as the devices remain in velocity saturation, which is probably not the case for the lower supply voltages. This introduces some inaccuracy in the analysis, but will not distort the overall result.

Example 5.15 Optimum Supply Voltage for 0.25-μm CMOS Inverter

From the technology parameters for our generic CMOS process presented in Chapter 3, the value of V_{TE} can be derived as follows:

$$V_{Tn} = 0.43\ \text{V},\ V_{Dsatn} = 0.63\ \text{V},\ V_{TEn} = 0.74\ \text{V}$$
$$V_{Tp} = -0.4\ \text{V},\ V_{Dsatp} = -1\ \text{V},\ V_{TEp} = -0.9\ \text{V}$$
$$V_{TE} \approx (V_{TEn} + |V_{TEp}|)/2 = 0.8\ \text{V}$$

Hence, $V_{DDopt} = (3/2) \times 0.8\ \text{V} = 1.2\ \text{V}$. The simulated graphs of Figure 5-36, which plot normalized delay, energy, and energy-delay product, confirm this result. The optimum supply voltage is predicted to equal 1.1 V. The charts clearly illustrate the trade-off between delay and energy.

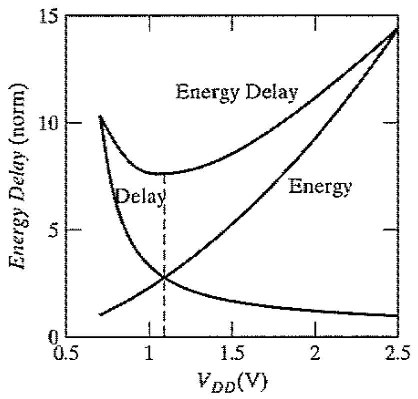


Figure 5-36 Normalized delay, energy, and energy-delay plots for CMOS inverter in 0.25-μm CMOS technology.

WARNING: While the preceding example demonstrates that a supply voltage exists that minimizes the energy-delay product of a gate, this voltage does not necessarily represent the optimum voltage for a given design problem. For instance, some designs require a minimum performance, which requires a higher voltage at the expense of energy. Similarly, a lower energy design is possible by operating at a lower voltage and by obtaining the overall system performance through the use of architectural techniques such as pipelining or concurrency.

5.5.4 Analyzing Power Consumption by Using SPICE

A definition of the average power consumption of a circuit was provided in Chapter 1, and is repeated here for the sake of convenience. We write

$$P_{av} = \frac{1}{T}\int_0^T p(t)dt = \frac{V_{DD}}{T}\int_0^T i_{DD}(t)dt \tag{5.60}$$

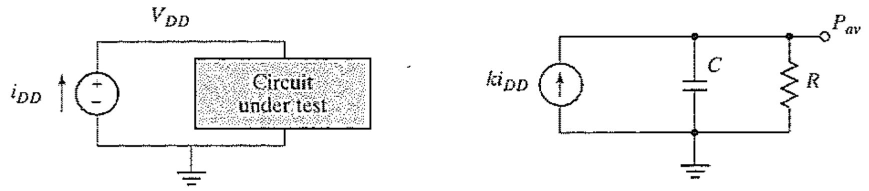


Figure 5-37 Equivalent circuit to measure average power in SPICE.

with T the period of interest, and V_{DD} and i_{DD} the supply voltage and current, respectively. Some implementations of SPICE provide built-in functions to measure the average value of a circuit signal. For instance, the HSPICE .MEASURE TRAN I(VDD) AVG command computes the area under a computed transient response (I(VDD)) and divides it by the period of interest. This is identical to the definition given in Eq. (5.60). Other implementations of SPICE are, unfortunately, not as powerful. This is not as bad as it seems, as long as one realizes that SPICE is actually a differential equation solver. A small circuit can easily be conceived that acts as an integrator and whose output signal is nothing but the average power.

Consider, for instance, the circuit of Figure 5-37. The current delivered by the power supply is measured by the current-controlled current source and integrated on the capacitor C. The resistance R is only provided for DC-convergence reasons and should be chosen as high as possible to minimize leakage. A clever choice of the element parameter ensures that the output voltage P_{av} equals the average power consumption. The operation of the circuit is summarized in Eq. under the assumption that the initial voltage on the capacitor C is zero:

$$C\frac{dP_{av}}{dt} = ki_{DD}$$

or (5.61)

$$P_{av} = \frac{k}{C}\int_0^T i_{DD}dt$$

Equating Eq. (5.60) and Eq. yields the necessary conditions for the equivalent circuit parameters: $k/C = V_{DD}/T$. Under these circumstances, the equivalent circuit shown presents a convenient means of tracking the average power in a digital circuit.

Example 5.16 Average Power of Inverter

The average power consumption of the inverter of Example 5.4 is analyzed using the above technique for a toggle period of 250 ps (T = 250 ps, k = 1, V_{DD} = 2.5 V, hence C = 100 pF). The resulting power consumption is plotted in Figure 5-38, showing an average power consumption of approximately 157.3 μW. The .MEAS AVG command yields a value of 160.3 μW, which demonstrates the approximate equivalence of both methods. These numbers are equivalent to an energy of 39 fJ (which is close to the 37.5 fJ derived in

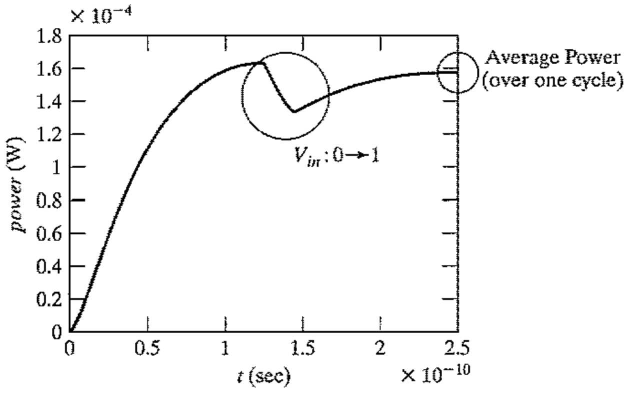


Figure 5-38 Deriving the power consumption by using SPICE.

Example 5.11). Observe the slightly negative dip during the high-to-low transition. This is due to the injection of current into the supply, when the output briefly overshoots V_{DD} as a result of the capacitive coupling between input and output (as is apparent from in the transient response of Figure 5-16).

5.6 Perspective: Technology Scaling and its Impact on the Inverter Metrics

In Section 3.5, we have explored the impact of the scaling of technology on some of the important design parameters, such as area, delay, and power. For the sake of clarity, we repeat here some of the most important entries of the scaling table (Table 3-8).

The validity of these theoretical projections can be verified by looking back and observing the trends during the past few decades. From Figure 5-39, we can see that the gate delay indeed decreases exponentially at a rate of 13% per year, or halving every five years. This rate is on course with the prediction of Table 5-4, since S averages approximately 1.15 as we had already

Table 5-4 Scaling scenarios for short-channel devices (S and U represent the technology and voltage scaling parameters, respectively).

Parameter	Relation	Full Scaling	General Scaling	Fixed-Voltage Scaling
Area–Device	*W–L*	$1/S^2$	$1/S^2$	$1/S^2$
Intrinsic Delay	$R_{on}C_{gate}$	$1/S$	$1/S$	$1/S$
Intrinsic Energy	$C_{gate}V^2$	$1/S^3$	$1/SU^2$	$1/S$
Intrinsic Power	*Energy–Delay*	$1/S^2$	$1/U^2$	1
Power Density	*P–Area*	1	S^2/U^2	S^2

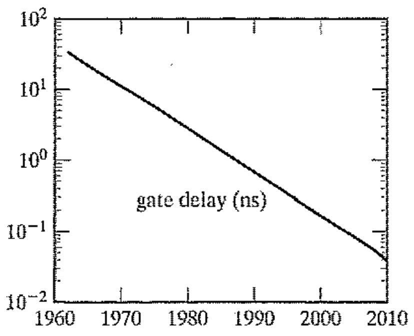


Figure 5-39 Scaling of the gate delay (from [Dally98]).

observed in Figure 3-40. The delay of a two-input NAND gate with a fan-out of four has gone from tens of nanoseconds in the 1960s to a tenth of a nanosecond in the year 2000, and is projected to be a few tens of picoseconds by 2010.

Reducing power dissipation has only been a second-order priority until recently. Hence, statistics on dissipation per gate or design are only marginally available. An interesting chart is shown in Figure 5-40, which plots the power density measured over a large number of designs produced between 1980 and 1995. Although the variation is large—even for a fixed technology—it shows the power density increasing approximately with S^2. This is in correspondence with the fixed-voltage scaling scenario presented in Table 5-4. For more recent years, we expect a scenario more in line with the full-scaling model—which predicts a constant power density—due to the accelerated supply-voltage scaling and the increased attention to power-reducing design techniques. Even under these circumstances, power dissipation per chip will continue to increase due to the ever-larger die sizes.

The scaling model presented has one major flaw, however. The performance and power predictions produce purely "intrinsic" numbers that take only device parameters into account. In Chapter 4, we concluded that the interconnect wires exhibit a different scaling behavior, and that wire parasitics may come to dominate the overall performance. Similarly, charging and discharging the wire capacitances may dominate the energy budget. To get a crisper perspective, one has to construct a combined model that considers device and wire scaling models simultaneously. The impact of the wire capacitance and its scaling behavior is summarized in Table 5-5. We adopt the fixed-resistance model introduced in Chapter 4. We furthermore assume that the resistance of the driver dominates the wire resistance, which is definitely the case for short to medium-long wires.

The model predicts that the interconnect-caused delay (and energy) gain in importance with the scaling of technology. This impact is limited to an increase with εc for short wires ($S = S_L$), but it becomes increasingly more significant for medium-range and long wires ($S_L < S$). These conclusions

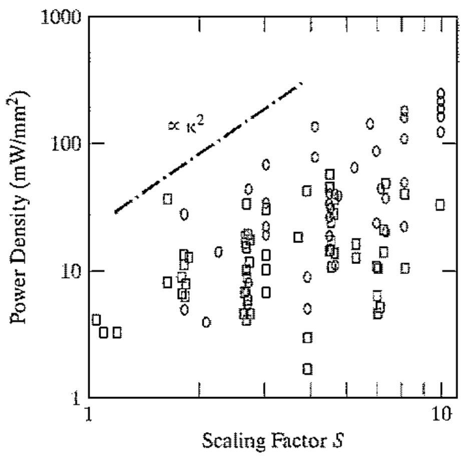


Figure 5-40 Evolution of power density in micro- and DSP processors, as a function of the scaling factor S ([Kuroda95]). S is normalized to 1 for a 4-μm process.

Table 5-5 Scaling scenarios for wire capacitance. S and U represent the technology and voltage scaling parameters, respectively, while S_L stands for the wire-length scaling factor. ε_c represents the impact of fringing and interwire capacitances.

Parameter	Relation	General Scaling
Wire Capacitance	WL/t	ε_c/S_L
Wire Delay	$R_{on}C_{int}$	ε_c/S_L
Wire Energy	$C_{int}V^2$	ε_c/S_LU^2
Wire Delay / Intrinsic Delay	▨	ε_cS/S_L
Wire Energy / Intrinsic Energy	▨	ε_cS/S_L

have been confirmed by a number of studies, an example of which is shown in Figure 5-41. How the ratio of wire over intrinsic contributions will actually evolve is debatable, as it depends upon a wide range of independent parameters, such as system architecture, design methodology, transistor sizing, and interconnect materials. The doomsday scenario that interconnect may cause CMOS performance to saturate in the very near future may very well be exaggerated. Yet, it is clear that increased attention to interconnect is an absolute necessity, and may change the way the next-generation circuits are designed and optimized (e.g., [Sylvester98]).

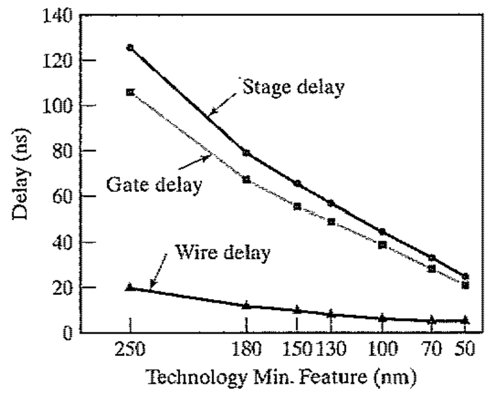


Figure 5-41 Evolution of wire delay-to-gate delay ratio with respect to technology (from [Fisher98]).

5.7 Summary

This chapter presented a rigorous and in-depth analysis of the static CMOS inverter. The key characteristics of the gate are summarized as follows:

- The static CMOS inverter combines a pull-up PMOS section with a pull-down NMOS device. The PMOS is normally made wider than the NMOS due to its lower current-driving capabilities.
- The gate has an almost ideal voltage-transfer characteristic. The logic swing is equal to the supply voltage and is not a function of the transistor sizes. The noise margins of a symmetrical inverter (where PMOS and NMOS transistor have equal current-driving strength) approach $V_{DD}/2$. The steady-state response is not affected by fan-out.
- Its propagation delay is dominated by the time it takes to charge or discharge the load capacitor C_L. To a first order, it can be approximated as follows:

$$t_p = 0.69C_L\left(\frac{R_{eqn} + R_{eqp}}{2}\right)$$

  Keeping the load capacitance small is the most effective means of implementing high-performance circuits. Transistor sizing may help to improve performance as long as the delay is dominated by the extrinsic (or load) capacitance of fan-out and wiring.
- The power dissipation is dominated by the dynamic power consumed in charging and discharging the load capacitor. It is given by $P_{0\rightarrow 1}\ C_L V_{DD}^2 f$. The dissipation is proportional to the activity in the network. The dissipation due to the direct-path currents occurring during switching can be limited by careful tailoring of the signal slopes. The static dissipation usually can be ignored, but might become a major factor in the future as a result of subthreshold currents.

- Scaling the technology is an effective means of reducing the area, propagation delay and power consumption of a gate. The impact is even more striking if the supply voltage is scaled simultaneously.
- The interconnect component is gradually taking a larger fraction of the delay and performance budget.

5.8 To Probe Further

The operation of the CMOS inverter has been the topic of numerous publications and textbooks. Virtually every book on digital design devotes a substantial number of pages to the analysis of the basic inverter gate. An extensive list of references was presented in Chapter 1. Some references of particular interest that we quoted in this chapter follow.

References

[Dally98] W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998.

[Fisher98] P. D. Fisher and R. Nesbitt, "The Test of Time: Clock-Cycle Estimation and Test Challenges for Future Microprocessors," *IEEE Circuits and Devices Magazine,* 14(2), pp. 37–44, 1998.

[Hedenstierna87] N. Hedenstierna and K. Jeppson, "CMOS Circuit Speed and Buffer Optimization," *IEEE Transactions on CAD*, vol. CAD-6, no. 2, pp. 270–281, March 1987.

[Kuroda95] T. Kuroda and T. Sakurai, "Overview of low-power ULSI circuit techniques," *IEICE Trans. on Electronics*, vol. E78-C, no. 4, pp. 334-344, April 1995.

[Liu93] D. Liu and C. Svensson, "Trading speed for low power by choice of supply and threshold voltages," *IEEE Journal of Solid-State Circuits*, vol. 28, no.1, pp. 10–17, Jan. 1993, p.10–17.

[Mead80] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[Sedra87] A. Sedra and K. Smith, *MicroElectronic Circuits*, Holt, Rinehart and Winston, 1987.

[Swanson72] R. Swanson and J. Meindl, "Ion-Implanted Complementary CMOS transistors in Low-Voltage Circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-7, no. 2, pp.146–152, April 1972.

[Sylvester98] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron," *Proceedings ICCAD Conference*, pp. 203, San Jose, November 1998.

[Veendrick84] H. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-19, no. 4, pp. 468–473, 1984.

Exercises and Design Problems

REMINDER: Please refer to **http://bwrc.eecs.berkeley.edu/IcBook** for up-to-date problem sets, design problems, and exercises. By making the exercises electronically available instead of in print, we can provide a dynamic environment that tracks the rapid evolution of today's digital integrated circuit design technology.

CHAPTER

6

Designing Combinational Logic Gates in CMOS

In-depth discussion of logic families in CMOS—
static and dynamic, pass-transistor, nonratioed and ratioed logic

Optimizing a logic gate for area, speed, energy, or robustness

Low-power and high-performance circuit-design techniques

6.1 Introduction

The design considerations for a simple inverter circuit were presented in the previous chapter. We now extend this discussion to address the synthesis of arbitrary digital gates, such as NOR, NAND, and XOR. The focus is on *combinational logic* or *nonregenerative* circuits—that is, circuits having the property that at any point in time, the output of the circuit is related to its current input signals by some Boolean expression (assuming that the transients through the logic gates have settled). No intentional connection from outputs back to inputs is present.

This is in contrast to another class of circuits, known as *sequential* or *regenerative*, for which the output is not only a function of the current input data, but also of previous values of the input signals (see Figure 6-1). This can be accomplished by connecting one or more outputs intentionally back to some inputs. Consequently, the circuit "remembers" past events and has a sense of *history*. A sequential circuit includes a combinational logic portion and a module that holds the state. Example circuits are registers, counters, oscillators, and memory. Sequential circuits are the topic of the next chapter.

There are numerous circuit styles to implement a given logic function. As with the inverter, the common design metrics by which a gate is evaluated are area, speed, energy, and power. Depending on the application, the emphasis will be on different metrics. For example, the switching speed of digital circuits is the primary metric in a high-performance processor, while in a battery operated circuit, it is energy dissipation. Recently, power dissipation also has become an important concern and considerable emphasis is placed on understanding the sources of power and approaches to dealing with power. In addition to these metrics, robustness to noise and reliability are also very important considerations. We will see that certain logic styles can significantly improve performance, but they usually are more sensitive to noise.

6.2 Static CMOS Design

The most widely used logic style is static complementary CMOS. The static CMOS style is really an extension of the static CMOS inverter to multiple inputs. To review, the primary advantage of the CMOS structure is robustness (i.e., low sensitivity to noise), good performance, and low power consumption with no static power dissipation. Most of those properties are carried over to large fan-in logic gates implemented using a similar circuit topology.

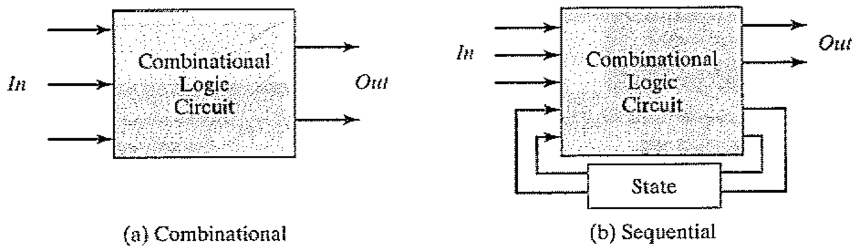


Figure 6-1 High-level classification of logic circuits.

The complementary CMOS circuit style falls under a broad class of logic circuits called *static* circuits in which **at every point in time, each gate output is connected to either V_{DD} or V_{SS} via a low-resistance path.** Also, the outputs of the gates assume at all times the value of the Boolean function implemented by the circuit (ignoring, the transient effects during switching periods). This is in contrast to the *dynamic* circuit class, which relies on temporary storage of signal values on the capacitance of high-impedance circuit nodes. The latter approach has the advantage that the resulting gate is simpler and faster. Its design and operation are, however, more involved and prone to failure because of increased sensitivity to noise.

In this section, we sequentially address the design of various static circuit flavors, including complementary CMOS, ratioed logic (pseudo-NMOS and DCVSL), and pass-transistor logic. We also deal with issues of scaling to lower power supply voltages and threshold voltages.

6.2.1 Complementary CMOS

Concept

A static CMOS gate is a combination of two networks—the *pull-up network* (PUN) and the *pull-down* network (PDN), as shown in Figure 6-2. The figure shows a generic N-input logic gate where all inputs are distributed to both the pull-up and pull-down networks. The function of the PUN is to provide a connection between the output and V_{DD} anytime the output of the logic gate is meant to be 1 (based on the inputs). Similarly, the function of the PDN is to connect the output to V_{SS} when the output of the logic gate is meant to be 0. The PUN and PDN networks are constructed in a mutually exclusive fashion such that *one and only one* of the networks is conducting in steady state. In this way, once the transients have settled, a path always exists between V_{DD} and the output F for a high output ("one"), or between V_{SS} and F for a low output ("zero"). This is equivalent to stating that the output node is always a *low-impedance* node in steady state.

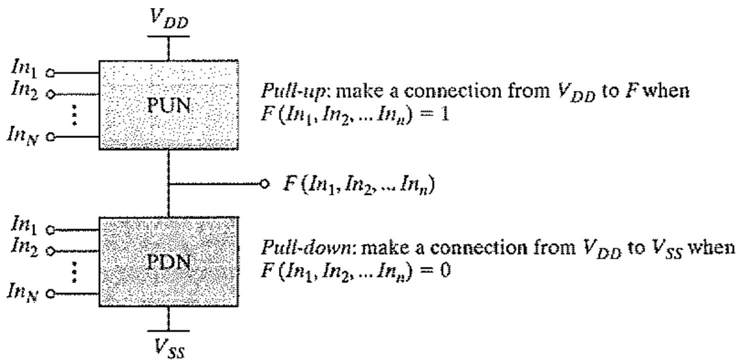
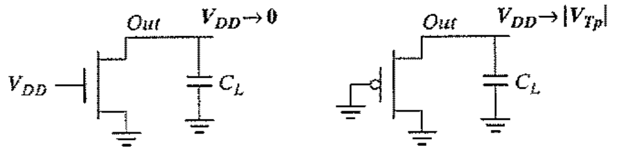

V_{DD}
In_1
In_2
In_N
PUN
Pull-up: make a connection from V_{DD} to F when $F(In_1, In_2, \ldots In_n) = 1$
$F(In_1, In_2, \ldots In_n)$
In_1
In_2
In_N
PDN
Pull-down: make a connection from V_{DD} to V_{SS} when $F(In_1, In_2, \ldots In_n) = 0$
V_{SS}

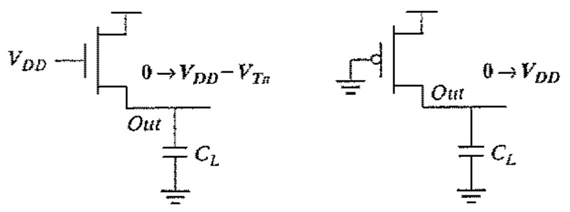
Figure 6-2 Complementary logic gate as a combination of a PUN (pull-up network) and a PDN (pull-down network).

In constructing the PDN and PUN networks, the designer should keep the following observations in mind:

- A transistor can be thought of as a switch controlled by its gate signal. An NMOS switch is *on* when the controlling signal is high and is *off* when the controlling signal is low. A PMOS transistor acts as an inverse switch that is *on* when the controlling signal is low and *off* when the controlling signal is high.
- The PDN is constructed using NMOS devices, while PMOS transistors are used in the PUN. The primary reason for this choice is that NMOS transistors produce "strong zeros," and PMOS devices generate "strong ones." To illustrate this, consider the examples shown in Figure 6-3. In Figure 6-3a, the output capacitance is initially charged to V_{DD}. Two possible discharge scenarios are shown. An NMOS device pulls the output all the way down to GND, while a PMOS lowers the output no further than $|V_{Tp}|$—the PMOS turns *off* at that point and stops contributing discharge current. NMOS transistors are thus the preferred devices in the PDN. Similarly, two alternative approaches to charging up a capacitor are shown in Figure 6-3b, with the output initially at GND. A PMOS switch succeeds in charging the output all the way to V_{DD}, while the NMOS device fails to raise the output above $V_{DD} - V_{Tn}$. This explains why PMOS transistors are preferentially used in a PUN.
- A set of rules can be derived to construct logic functions (see Figure 6-4). NMOS devices connected in series correspond to an AND function. With all the inputs high, the series combination conducts and the value at one end of the chain is transferred to the other end. Similarly, NMOS transistors connected in parallel represent an OR function. A conducting path exists between the output and input terminal if at least one of the inputs is high. Using similar arguments, construction rules for PMOS networks can be formulated. A series con-

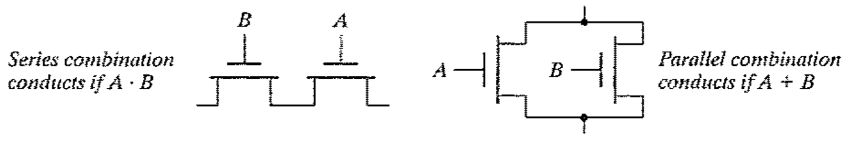


(a) Pulling down a node by using NMOS and PMOS switches



(b) Pulling down a node by using NMOS and PMOS switches

Figure 6-3 Simple examples illustrate why an NMOS should be used as a pull-down, and a PMOS should be used as a pull-up device.


Series combination conducts if $A \cdot B$
B
A
Parallel combination conducts if $A + B$

(a) Series (b) Parallel

Figure 6-4 NMOS logic rules—series devices implement an AND, and parallel devices implement an OR.

nection of PMOS conducts if both inputs are low, representing a NOR function ($\overline{A} \cdot \overline{B} = \overline{A + B}$), while PMOS transistors in parallel implement a NAND ($\overline{A} + \overline{B} = \overline{A \cdot B}$).

- Using De Morgan's theorems ($\overline{A + B} = \overline{A} \cdot \overline{B}$ and $\overline{A \cdot B} = \overline{A} + \overline{B}$), it can be shown that the pull-up and pull-down networks of a complementary CMOS structure are *dual* networks. This means that a parallel connection of transistors in the pull-up network corresponds to a series connection of the corresponding devices in the pull-down network, and vice versa. Therefore, to construct a CMOS gate, one of the networks (e.g., PDN) is implemented using combinations of series and parallel devices. The other network (i.e., PUN) is obtained using the duality principle by walking the hierarchy, replacing series subnets with parallel subnets, and parallel subnets with series subnets. The complete CMOS gate is constructed by combining the PDN with the PUN.
- The complementary gate is naturally *inverting*, implementing only functions such as NAND, NOR, and XNOR. The realization of a noninverting Boolean function (such as AND OR, or XOR) in a single stage is not possible, and requires the addition of an extra inverter stage.
- The number of transistors required to implement an N-input logic gate is $2N$.

Example 6.1 Two-Input NAND Gate

Figure 6-5 shows a two-input NAND gate ($F = \overline{A \cdot B}$). The PDN network consists of two NMOS devices in series that conduct when both A and B are high. The PUN is the dual

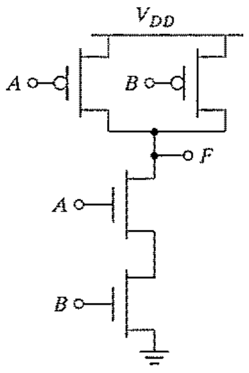


Figure 6-5 Two-input NAND gate in complementary static CMOS style.

network, and it consists of two parallel PMOS transistors. This means that F is 1 if $A = 0$ or $B = 0$, which is equivalent to $F = \overline{A \cdot B}$. The truth table for the simple two input NAND gate is given in Table 6-1. It can be verified that the output F is always connected to either V_{DD} or GND, but never to both at the same time.

Table 6-1 Truth Table for two-Input NAND.

A	*B*	*F*
0	0	1
0	1	1
1	0	1
1	1	0

Example 6.2 Synthesis of Complex CMOS Gate

Using complementary CMOS logic, consider the synthesis of a complex CMOS gate whose function is $F = \overline{D + A \cdot (B + C)}$. The first step in the synthesis of the logic gate is to derive the pull-down network as shown in Figure 6-6a by using the fact that NMOS devices in series implements the AND function and parallel device implements the OR function. The next step is to use duality to derive the PUN in a hierarchical fashion. The PDN network is broken into smaller networks (i.e., subset of the PDN) called subnets that simplify the derivation of the PUN. In Figure 6-6b, the subnets (SN) for the pull-down net-

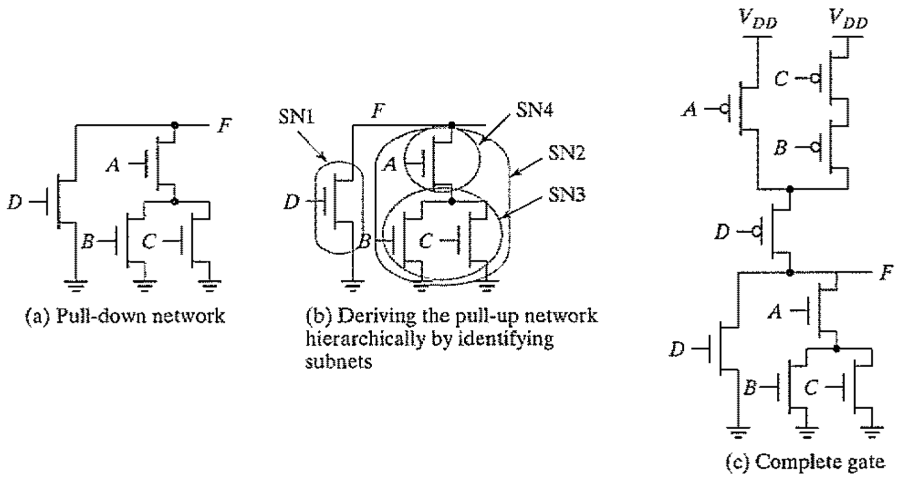


Figure 6-6 Complex complementary CMOS gate.

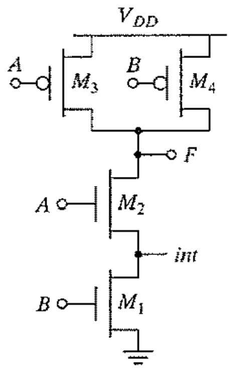
work are identified. At the top level, SN1 and SN2 are in parallel, so that in the dual network they will be in series. Since SN1 consists of a single transistor, it maps directly to the pull-up network. On the other hand, we need to sequentially apply the duality rules to SN2. Inside SN2, we have SN3 and SN4 in series, so in the PUN they will appear in parallel. Finally, inside SN3, the devices are in parallel, so they appear in series in the PUN. The complete gate is shown in Figure 6-6c. The reader can verify that for every possible input combination, there always exists a path to either V_{DD} or GND.

Static Properties of Complementary CMOS Gates

Complementary CMOS gates inherit all the nice properties of the basic CMOS inverter. They exhibit rail-to-rail swing with $V_{OH} = V_{DD}$ and V_{OL} = GND. The circuits also have no static power dissipation, since the circuits are designed such that the pull-down and pull-up networks are mutually exclusive. The analysis of the DC voltage transfer characteristics and the noise margins is more complicated than for the inverter, as these parameters **depend upon the data input patterns** applied to gate.

Consider the static two-input NAND gate shown in Figure 6-7. Three possible input combinations switch the output of the gate from high to low: (a) $A = B = 0 \rightarrow 1$, (b) $A = 1, B = 0 \rightarrow 1$, and (c) $B = 1, A = 0 \rightarrow 1$. The resulting voltage transfer curves display significant differences. The large variation between case (a) and the others (b and c) is explained by the fact that in the former case, both transistors in the pull-up network are on simultaneously for $A = B = 0$, representing a strong pull-up. In the latter cases, only one of the pull-up devices is on. The VTC is shifted to the left as a result of the weaker PUN.

The difference between (b) and (c) results mainly from the state of the internal node *int* between the two NMOS devices. For the NMOS devices to turn on, both gate-to-source voltages



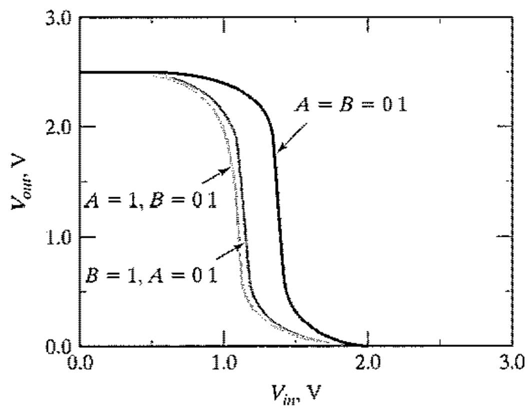


Figure 6-7 The VTC of a two-input NAND is data dependent. NMOS devices are 0.5 μm/0.25 μm while the PMOS devices are sized at 0.75 μm/0.25 μm.

must be above V_{Tn}, with $V_{GS2} = V_A - V_{DS1}$ and $V_{GS1} = V_B$. The threshold voltage of transistor M_2 will be higher than transistor M_1 due to the body effect. The threshold voltages of the two devices are given by the following equations:

$$V_{Tn2} = V_{Tn0} + \gamma((\sqrt{|2\phi_f| + V_{int}}) - \sqrt{|2\phi_f|}) \tag{6.1}$$

$$V_{Tn1} = V_{Tn0} \tag{6.2}$$

For case (b), M_3 is turned *off*, and the gate voltage of M_2 is set to V_{DD}. To a first order, M_2 may be considered as a resistor in series with M_1. Since the drive on M_2 is large, this resistance is small and has only a small effect on the voltage transfer characteristics. In case (c), transistor M_1 acts as a resistor, causing a V_T increase in M_2 due to body effect. The overall impact is quite small, as seen from the plot.

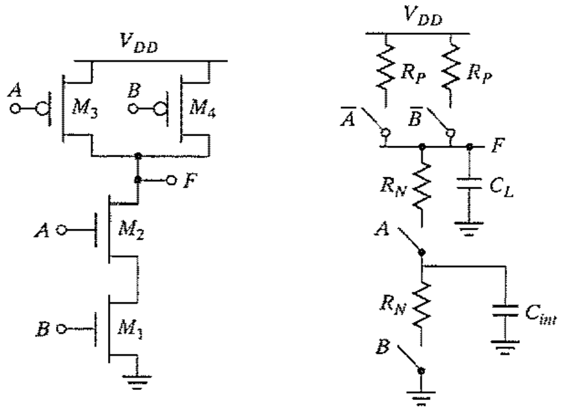
Design Consideration

The important point to take away from the preceding discussion is that the **noise margins are input/pattern dependent**. In Example 6.2, a glitch on only one of the two inputs has a larger chance of creating a false transition at the output than if the glitch were to occur on both inputs simultaneously. Therefore, the former condition has a lower low-noise margin. A common practice when characterizing gates such as NAND and NOR is to connect all the inputs together. Unfortunately, this does not represent the worst case static behavior; the data dependencies should be carefully modeled. ■

Propagation Delay of Complementary CMOS Gates

The computation of propagation delay proceeds in a fashion similar to the static inverter. For the purpose of delay analysis, each transistor is modeled as a resistor in series with an ideal switch. The value of the resistance is dependent on the power supply voltage and an equivalent large signal resistance, scaled by the ratio of device width over length, must be used. The logic is transformed into an equivalent *RC* network that includes the effect of internal node capacitances. Figure 6-8 shows the two-input NAND gate and its equivalent *RC* switch level model. Note that the internal node capacitance C_{int}—attributable to the source/drain regions and the gate overlap capacitance of M_2 and M_1—is included here. While complicating the analysis, the capacitance of the internal nodes can have quite an impact in some networks such as large fan-in gates. In a first pass, we ignore the effect of the internal capacitance.

A simple analysis of the model shows that, similarly to the noise margins, **the propagation delay depends on the input patterns**. Consider, for instance, the low-to-high transition. Three possible input scenarios can be identified for charging the output to V_{DD}. If both inputs are driven low, the two PMOS devices are on. The delay in this case is $0.69 \times (R_p/2) \times C_L$, since the two resistors are in parallel. This is not the worst case low-to-high transition, which occurs when only one device turns on, and is given by $0.69 \times R_p \times C_L$. For the pull-down path, the output is discharged only if both *A* and *B* are switched high, and the delay is given by $0.69 \times (2R_N) \times C_L$ to a first order. In other words, adding devices in series slows down the circuit, and devices must be made wider to avoid a performance penalty. When sizing the transistors in a gate with multiple inputs, we should pick the combination of inputs that triggers the worst case conditions.



(a) Two-input NAND (b) RC-equivalent model

Figure 6-8 Equivalent RC model for a two-input NAND gate.

For the NAND gate to have the same pull-down delay (t_{phl}) as a minimum-sized inverter, the NMOS devices in the PDN stack must be made twice as wide so that the equivalent resistance of the NAND pull-down network is the same as the inverter. The PMOS devices can remain unchanged.[1]

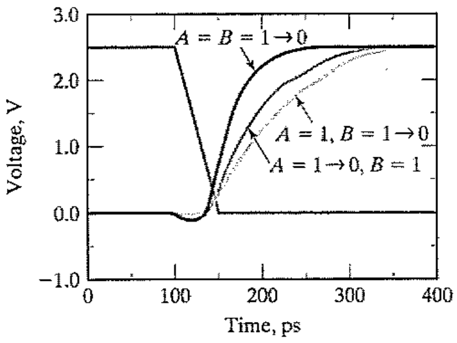
This first-order analysis assumes that the extra capacitance introduced by widening the transistors can be ignored. This is not a good assumption, in general, but it allows for a reasonable first cut at device sizing.

Example 6.3 Delay Dependence on Input Patterns

Consider the NAND gate of Figure 6-8a. Assume NMOS and PMOS devices of 0.5 μm/0.25 μm and 0.75 μm/0.25 μm, respectively. This sizing should result in approximately equal worst case rise and fall times (since the effective resistance of the pull-down is designed to be equal to the pull-up resistance).

Figure 6-9 shows the simulated low-to-high delay for different input patterns. As expected, the case in which both inputs transition go low ($A = B = 1 \to 0$) results in a smaller delay, compared with the case in which only one input is driven low. Notice that the worst case low-to-high delay depends upon which input (A or B) goes low. The reason for this involves the internal node capacitance of the pull-down stack (i.e., the source of M_2). For the case in which $B = 1$ and A transitions from $1 \to 0$, the pull-up PMOS device only has to charge up the output node capacitance (M_2 is turned off). On the other hand, for the case in which $A = 1$ and B transitions from $1 \to 0$, the pull-up PMOS device

[1] In deep-submicron processes, even larger increases in the width are needed due to the on-set of velocity saturation. For a two-input NAND, the NMOS transistors should be made 2.5 times as wide instead of 2 times.



Input Data Pattern	Delay (ps)
$A = B = 0 \rightarrow 1$	69
$A = 1, B = 0 \rightarrow 1$	62
$A = 0 \rightarrow 1, B = 1$	50
$A = B = 1 \rightarrow 0$	35
$A = 1, B = 1 \rightarrow 0$	76
$A = 1 \rightarrow 0, B = 1$	57

Figure 6-9 Example showing the delay dependence on input patterns.

has to charge up the sum of the output and the internal node capacitances, which slows down the transition.

The table in Figure 6-9 shows a compilation of various delays for this circuit. The first-order transistor sizing indeed provides approximately equal rise and fall delays. An important point to note is that the high-to-low propagation delay depends on the initial state of the internal nodes. For example, when both inputs transition from $0 \rightarrow 1$, it is important to establish the state of the internal node. The worst case happens when the internal node is initially charged up to $V_{DD} - V_{Tn}$, which can be ensured by pulsing the A input from $1 \rightarrow 0 \rightarrow 1$, while input B only makes the $0 \rightarrow 1$ transition. In this way, the internal node is initialized properly.

The important point to take away from this example is that estimation of delay can be fairly complex, and requires a careful consideration of internal node capacitances and data patterns. Care must be taken to model the worst case scenario in the simulations. A brute force approach that applies all possible input patterns may not always work, because it is important to consider the state of internal nodes.

The CMOS implementation of a NOR gate ($F = \overline{A + B}$) is shown in Figure 6-10. The output of this network is high, if and only if both inputs A and B are low. The worst case pull-down transition happens when only one of the NMOS devices turns on (i.e., if either A or B is high). Assume that the goal is to size the NOR gate such that it has approximately the same delay as an inverter with the following device sizes: NMOS of 0.5 μm/0.25 μm and PMOS of 1.5 μm/0.25 μm. Since the pull-down path in the worst case is a single device, the NMOS devices (M_1 and M_2) can have the same device widths as the NMOS device in the inverter. For the output to be pulled high, both devices must be turned on. Since the resistances add, the devices must be made two times larger compared with the PMOS in the inverter (i.e., M_3 and M_4 must have a size of 3 μm/0.25 μm). Since PMOS devices have a lower mobility relative to NMOS devices, stacking devices in series must be avoided as much as possible. A NAND implementation is clearly preferred over a NOR implementation for implementing generic logic.

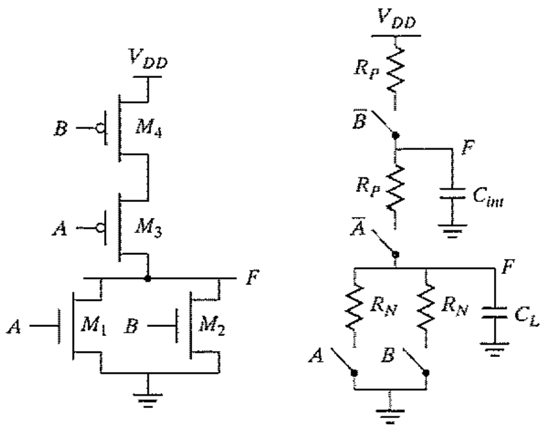


Figure 6-10 Sizing of a NOR gate.

Problem 6.1 Transistor Sizing in Complementary CMOS Gates

Determine the sizes of the transistors in Figure 6-6c such that it has approximately the same t_{plh} and t_{phl} as an inverter with the following sizes: NMOS: 0.5 μm/0.25 μm and a PMOS: 1.5 μm/0.25 μm.

So far in the analysis of propagation delay, we have ignored the effect of internal node capacitances. This is often a reasonable assumption for a first-order analysis. However, in more complex logic gates with large *fan-ins*, the internal node capacitances can become significant. Consider a four-input NAND gate, as drawn in Figure 6-11, which shows the equivalent *RC* model of the gate, including the internal node capacitances. The internal capacitances consist of the junction capacitances of the transistors, as well as the gate-to-source and gate-to-drain capacitances. The latter are turned into capacitances to ground using the Miller equivalence. The delay analysis for such a circuit involves solving distributed *RC* networks, a problem we already encountered when analyzing the delay of interconnect networks. Consider the pull-down delay of the circuit. The output is discharged when all inputs are driven high. The proper initial conditions must be placed on the internal nodes (i.e., the internal nodes must be charged to $V_{DD} - V_{TN}$) before the inputs are driven high.

The propagation delay can be computed by using the Elmore delay model:

$$t_{pHL} = 0.69(R_1 \cdot C_1 + (R_1 + R_2) \cdot C_2 + (R_1 + R_2 + R_3) \cdot C_2 + (R_1 + R_2 + R_3 + R_4) \cdot C_L) \quad (6.3)$$

Notice that the resistance of M_1 appears in all the terms, which makes this device especially important when attempting to minimize delay. Assuming that all NMOS devices have an equal size, Eq. (6.3) simplifies to

$$t_{pHL} = 0.69R_N(C_1 + 2 \cdot C_2 + 3 \cdot C_3 + 4 \cdot C_L) \quad (6.4)$$

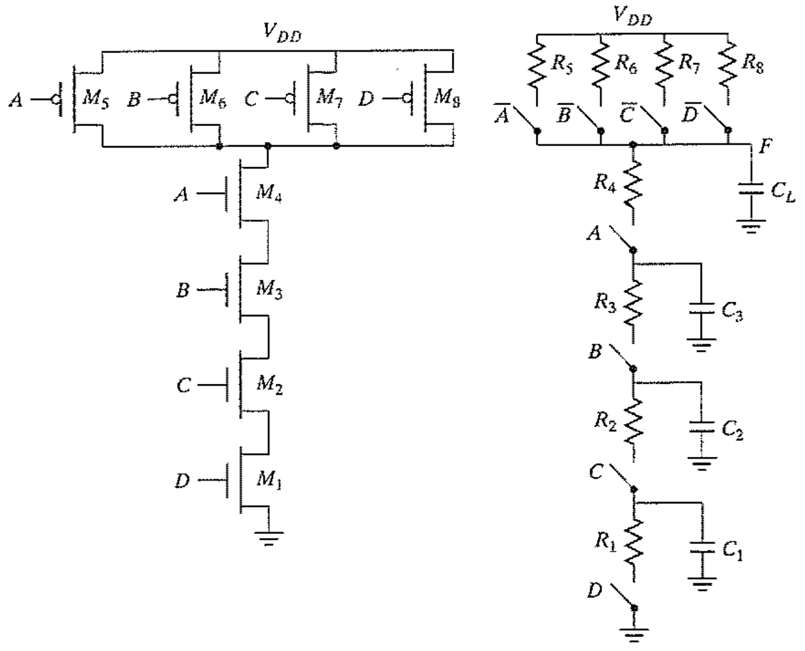


Figure 6-11 Four-input NAND gate and its *RC* model.

Example 6.4 A Four-Input Complementary CMOS NAND Gate

In this example, we evaluate the *intrinsic (or unloaded) propagation delay* of a four-input NAND gate (without any loading) is evaluatedusing hand analysis and simulation. The layout of the gate is shown in Figure 6-12. Assume that all NMOS devices have a *W/L* of 0.5 μm/0.25 μm, and all PMOS devices have a device size of 0.375 μm/0.25 μm. The devices are sized such that the worst case rise and fall times are approximately equal to a first order (ignoring the internal node capacitances).

By using techniques similar to those employed for the CMOS inverter in Chapter 5, the capacitance values can be computed from the layout. Notice that in the pull-up path, the PMOS devices share the drain terminal, in order to reduce the overall parasitic contribution. Using our standard design rules, we find that the area and perimeter for various devices can be easily computed, as shown in Table 6-2.

In this example, we focus on the pull-down delay, and the capacitances will be computed for the high-to-low transition at the output. While the output makes a transition from V_{DD} to 0, the internal nodes only transition from $V_{DD} - V_{Tn}$ to GND. We need to linearize the internal junction capacitances for this voltage transition, but, to simplify the analysis, we use the same K_{eff} for the internal nodes as for the output node.

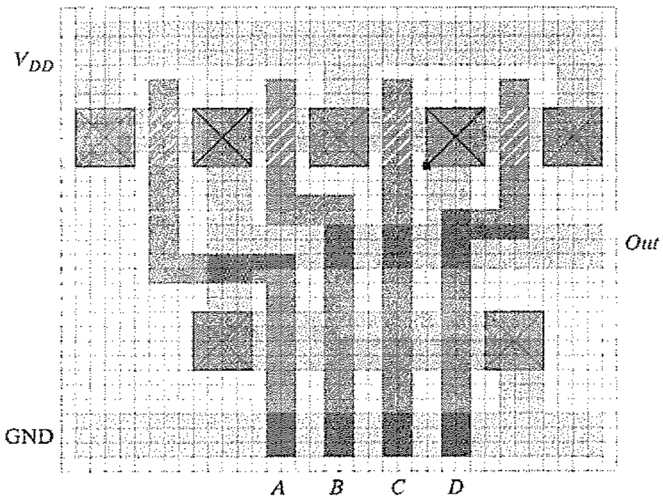


Figure 6-12 Layout a four-input NAND gate in complementary CMOS. See also Colorplate 7.

Table 6-2 Area and perimeter of transistors in four-input NAND gate.

Transistor	W (μm)	AS (μm^2)	AD (μm^2)	PS (μm)	PD (μm)
1	0.5	0.3125	0.0625	1.75	0.25
2	0.5	0.0625	0.0625	0.25	0.25
3	0.5	0.0625	0.0625	0.25	0.25
4	0.5	0.0625	0.3125	0.25	1.75
5	0.375	0.297	0.172	1.875	0.875
6	0.375	0.172	0.172	0.875	0.875
7	0.375	0.172	0.172	0.875	0.875
8	0.375	0.297	0.172	1.875	0.875

It is assumed that the output connects to a single, minimum-size inverter. The effect of intracell routing, which is small, is ignored. The various contributions are summarized in Table 6-3. For the NMOS and PMOS junctions, we use $K_{eq} = 0.57$, $K_{eqsw} = 0.61$, and $K_{eq} = 0.79$, $K_{eqsw} = 0.86$, respectively. Notice that the gate-to-drain capacitance is multiplied by a factor of two for all internal nodes as well as the output node, to account for the Miller effect. (This ignores the fact that the internal nodes have a slightly smaller swing due to the threshold drop.)

Table 6-3 Computation of capacitances for high-to-low transition at the output. The table shows the intrinsic delay of the gate without extra loading. Any *fan-out* capacitance would simply be added to the C_L term.

Capacitor	Contributions (H → L)	Value (fF) (H → L)
C1	$C_{d1} + C_{s2} + 2 * C_{gd1} + 2 * C_{gs2}$	(0.57 * 0.0625 * 2 + 0.61 * 0.25 * 0.28) + (0.57 * 0.0625 * 2 + 0.61 * 0.25* 0.28) + 2 * (0.31 * 0.5) + 2 * (0.31 * 0.5) = 0.85 fF
C2	$C_{d2} + C_{s3} + 2 * C_{gd2} + 2 * C_{gs3}$	(0.57 * 0.0625 * 2 + 0.61 * 0.25 * 0.28) + (0.57 * 0.0625 * 2 + 0.61 * 0.25* 0.28) + 2 * (0.31 * 0.5) + 2 * (0.31 * 0.5) = 0.85 fF
C3	$C_{d3} + C_{s4} + 2 * C_{gd3} + 2 * C_{gs4}$	(0.57 * 0.0625 * 2+ 0.61 * 0.25 * 0.28) + (0.57 * 0.0625 * 2+ 0.61 * 0.25* 0.28) + 2 * (0.31 * 0.5) + 2 * (0.31 * 0.5) = 0.85 fF
CL	$C_{d4} + 2 * C_{gd4} + C_{d5} + C_{d6} + C_{d7} + C_{d8} + 2 * C_{gd5} + 2 * C_{gd6} + 2 * C_{gd7} + 2 * C_{gd8} = C_{d4} + 4 * C_{d5} + 4 * 2 * C_{gd6}$	(0.57 * 0.3125 * 2 + 0.61 * 1.75 *0.28) + 2 * (0.31 * 0.5)+ 4 * (0.79 * 0.171875* 1.9+ 0.86 * 0.875 * 0.22)+ 4 * 2 * (0.27 * 0.375) = 3.47 fF

Using Eq. (6.4), we compute the propagation delay, as follows:

$$t_{pHL} = 0.69\left(\frac{13\text{K}\Omega}{2}\right)(0.85\text{ fF} + 2\cdot 0.85\text{ fF} + 3\cdot 0.85\text{ fF} + 4\cdot 3.47\text{ fF}) = 85\text{ ps}$$

The simulated delay for this particular transition was found to be 86 ps! The hand analysis gives a fairly accurate estimate, given all of the assumptions and linearizations that were made. For example, we assume that the gate–source (or gate–drain) capacitance only consists of the overlap component. This is not entirely the case, because, during the transition, some other contributions come in place depending upon the operating region. Once again, the goal of hand analysis is not to provide a totally accurate delay prediction, but rather to give intuition into what factors influence the delay and to aid in initial transistor sizing. Accurate timing analysis and transistor optimization is usually done using SPICE. The simulated worst case low-to-high delay time for this gate was 106 ps.

While complementary CMOS is a very robust and simple approach for implementing logic gates, there are two major problems associated with using this style as the complexity of the gate (i.e., *fan-in*) increases. First, the number of transistors required to implement an *N* fan-in gate is 2*N*. This can result in a significantly large implementation area.

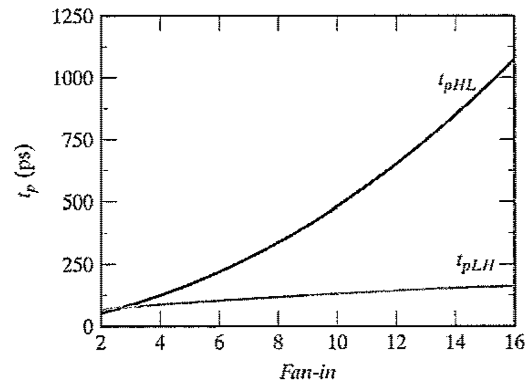


Figure 6-13 Propagation delay of CMOS NAND gate as a function of fan-in. A fan-out of one inverter is assumed, and all pull-down transistors are minimal size.

The second problem is that propagation delay of a complementary CMOS gate deteriorates rapidly as a function of the fan-in. In fact, the *unloaded intrinsic delay* of the gate is, at worst, a *quadratic function of the fan-in*.

- The large number of transistors ($2N$) increases the overall capacitance of the gate. For an N-input gate, the *intrinsic capacitance* increases linearly with the fan-in. Consider, for instance, the NAND gate of Figure 6-11. Given the linear increase in the number of PMOS devices connected to the output node, we expect the low-to-high delay of the gate to increase linearly with fan-in—while the capacitance goes up linearly, the pull-up resistance remains unchanged.
- The series connection of transistors in either the PUN or PDN of the gate causes an additional slowdown. We know that the *distributed RC network* in the PDN of Figure 6-11 comes with a delay that is quadratic in the number of elements in the chain. The high-to-low delay of the gate should hence be a quadratic function of the fan-in.

Figure 6-13 plots the (intrinsic) propagation delay of a NAND gate as a function of fan-in assuming a fixed fan-out of one inverter (NMOS: 0.5 μm and PMOS: 1.5 μm). As predicted, t_{pLH} is a linear function of fan-in, while the simultaneous increase in the pull-down resistance and the load capacitance cause an approximately quadratic relationship for t_{pHL}. Gates with a *fan-in* greater than or equal to 4 become excessively slow and must be avoided.

Design Techniques for Large Fan-in

The designer has a number of techniques at his disposition to reduce the delay of large fan-in circuits:

- **Transistor Sizing** The most obvious solution is to increase the transistor sizes. This lowers the resistance of devices in series and lowers the time constants. However, increasing the transistor sizes results in larger parasitic capacitors, which not only affect the *propagation delay* of the gate in question, but

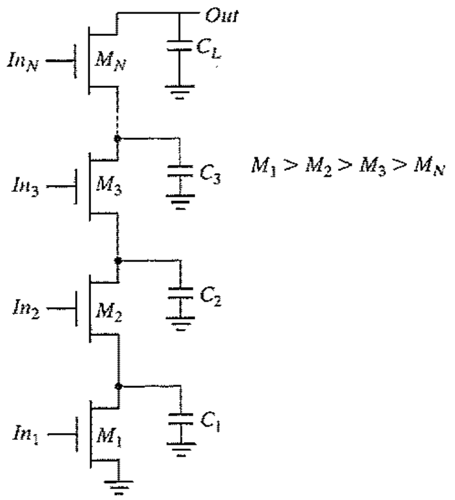


Figure 6-14 Progressive sizing of transistors in large transistor chains copes with the extra load of internal capacitances.

also present a larger load to the preceding gate. This technique should therefore be used with caution. If the load capacitance is dominated by the intrinsic capacitance of the gate, widening the device only creates a "self-loading" effect, and the *propagation delay* is unaffected. Sizing is only effective when the load is dominated by the fan-out. A more comprehensive approach toward sizing transistors in complex CMOS combinational networks is discussed in the next section.

- **Progressive Transistor Sizing** An alternate approach to uniform sizing (in which each transistor is scaled up uniformly), is to use progressive transistor sizing (Figure 6-14). Referring back to Eq. (6.3), we see that the resistance of M_1 (R_1) appears N times in the delay equation, the resistance of M_2 (R_2) appears $N - 1$ times, etc. From the equation, it is clear that R_1 should be made the smallest, R_2 the next smallest, etc. Consequently, a progressive scaling of the transistors is beneficial: $M_1 > M_2 > M_3 > M_N$. This approach reduces the dominant resistance, while keeping the increase in capacitance within bounds. For an excellent treatment on the optimal sizing of transistors in a complex network, we refer the interested reader to [Shoji88, pp. 131–143]. You should be aware, however, of one important pitfall of this approach. While progressive resizing of transistors is relatively easy in a schematic diagram, it is not as simple in a real layout. Very often, design-rule considerations force the designer to push the transistors apart, which causes the internal capacitance to grow. This may offset all the gains of the resizing!
- **Input Reordering** Some signals in complex combinational logic blocks might be more critical than others. Not all inputs of a gate arrive at the same time (due, for instance, to the propagation delays of the preceding logical gates). An input signal to a gate is called *critical* if it is the last signal of all inputs to assume a stable value. The path through the logic which determines the ultimate speed of the structure is called the *critical path*.

Putting the critical-path transistors closer to the output of the gate can result in a speed up, as demonstrated in Figure 6-15. Signal In_1 is assumed to be a critical signal. Suppose further that In_2 and In_3 are high, and that In_1 undergoes a $0 \rightarrow 1$ transition. Assume also that C_L is initially charged high. In case (a), no path to GND exists until M_1 is turned on, which, unfortunately, is the last event to happen. The delay between the arrival of In_1 and the output is therefore determined by the time it takes to discharge C_L, C_1 and C_2. In the second case, C_1 and C_2 are already

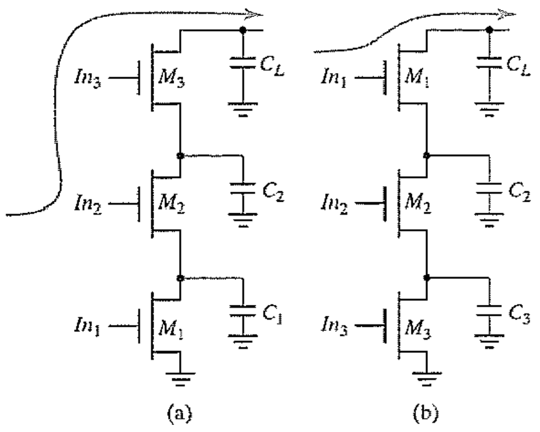


Figure 6-15 Influence of transistor ordering on delay. Signal In_1 is the critical signal.

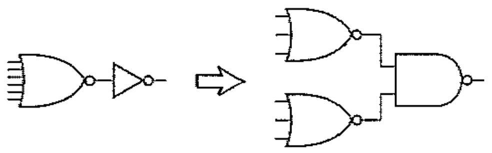


Figure 6-16 Logic restructuring can reduce the gate fan-in.

discharged when In_1 changes. Only C_L still has to be discharged, resulting in a smaller delay.

- **Logic Restructuring** Manipulating the logic equations can reduce the fan-in requirements and thus reduce the gate delay, as illustrated in Figure 6-16. The quadratic dependency of the gate delay on *fan-in* makes the six-input NOR gate extremely slow. Partitioning the NOR gate into two three-input gates results in a significant speedup, which by far offsets the extra delay incurred by turning the inverter into a two-input NAND gate. ■

Optimizing Performance in Combinational Networks

Earlier, we established that minimization of the propagation delay of a gate in isolation is a purely academic effort. The sizing of devices should happen in its proper context. In Chapter 5, we developed a methodology to do so for inverters. We also found that an optimal fan-out for a chain of inverters driving a load C_L is $(C_L/C_{in})^{1/N}$, where N is the number of stages in the chain, and C_{in} the input capacitance of the first gate in the chain. If we have an opportunity to select the number of stages, we found out that we would like to keep the fan-out per stage around 4. Can this result be extended to determine the size of any combinational path for minimal delay? By extending our previous approach to address complex logic networks, we find out that this is indeed possible [Sutherland99].[2]

[2]The approach introduced in this section is commonly called logical effort, and was formally introduced in [Sutherland99], which presents an extensive treatment of the topic. The treatment offered here represents only a glance over of the overall approach.

Table 6-4 Estimates of intrinsic delay factors of various logic types, assuming simple layout styles, and a fixed PMOS–NMOS ratio.

Gate type	p
Inverter	1
n-input NAND	n
n-input NOR	n
n-way multiplexer	$2n$
XOR, NXOR	$n2^{n-1}$

To do so, we modify the basic delay equation of the inverter that we introduced in Chapter 5, namely,

$$t_p = t_{p0}\left(1 + \frac{C_{ext}}{\gamma C_g}\right) = t_{p0}(1 + f/\gamma) \tag{6.5}$$

to

$$t_p = t_{p0}(p + gf/\gamma) \tag{6.6}$$

with t_{p0} still representing the intrinsic delay of an inverter and f the *effective fan-out*, defined as the ratio between the external load and the input capacitance of the gate. In this context, f is also called the *electrical effort*, and p represents the ratio of the intrinsic (or unloaded) delays of the complex gate and the simple inverter, and is a function of gate topology, as well as layout style. The more involved structure of the multiple-input gate causes its intrinsic delay to be higher than that of an inverter. Table 6-4 enumerates the values of p for some standard gates, assuming simple layout styles, and ignoring second-order effects such as internal node capacitances.

The factor g is called the *logical effort*, and represents the fact that, for a given load, complex gates have to work harder than an inverter to produce a similar response. In other words, the logical effort of a logic gate tells how much worse it is at producing output current than an inverter, given that each of its inputs may present only the same input capacitance as the inverter. Equivalently, logical effort is how much more input capacitance a gate presents to deliver the same output current as an inverter. Logical effort is a useful parameter, because it depends only on circuit topology. The logical efforts of some common logic gates are given in Table 6-5.

Table 6-5 Logic efforts of common logic gates, assuming a PMOS–NMOS ratio of 2.

Gate Type	1	2	3	n
	Number of Inputs			
Inverter	1			
NAND		4/3	5/3	(n + 2)/3
NOR		5/3	7/3	(2n + 1)/3
Multiplexer		2	2	2
XOR		4	12	

Example 6.5 Logical Effort of Complex Gates

Consider the gates shown in Figure 6-17. Assuming PMOS–NMOS ratio of 2, the input capacitance of a minimum-sized symmetrical inverter equals three times the gate capacitance of a minimum-sized NMOS (called C_{unit}). We size the two-input NAND and NOR such that their equivalent resistances equal the resistance of the inverter (using the techniques described earlier). This increases the input capacitance of the two-input NAND to 4 C_{unit}, or 4/3 the capacitance of the inverter. The input capacitance of the two-input NOR is 5/3 that of the inverter. Equivalently, for the same input capacitance, the NAND and NOR gate have 4/3 and 5/3 less driving strength than the inverter. This affects the delay component that corresponds to the load, increasing it by this same factor, called the *logical effort*. Hence, $g_{NAND} = 4/3$, and $g_{NOR} = 5/3$.

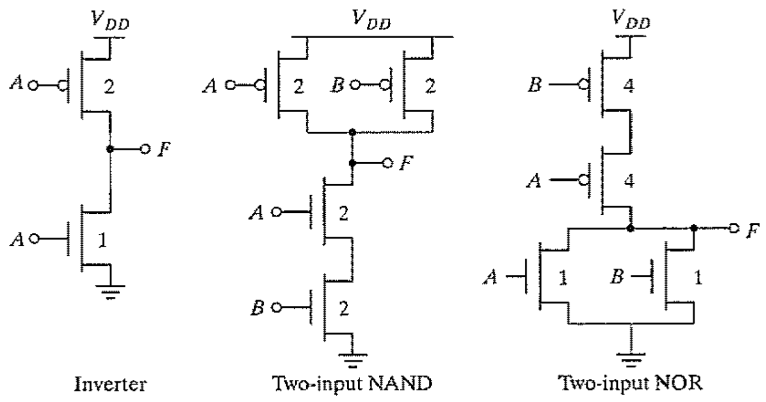


Figure 6-17 Logical effort of two-input NAND and NOR gates.

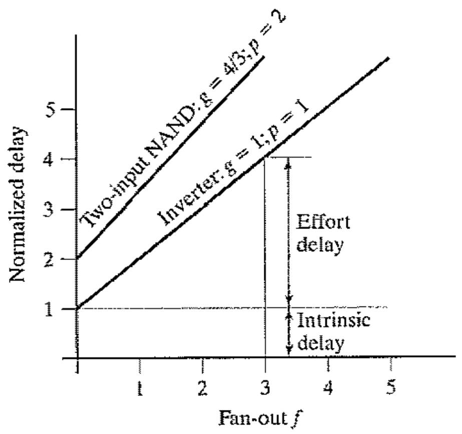


Figure 6-18 Delay as a function of fan-out for an inverter and a two-input NAND.

The delay model of a logic gate, as represented in Eq. (6.6), is a simple linear relationship. Figure 6-18 shows this relationship graphically: the delay is plotted as a function of the fan-out for an inverter and for a two-input NAND gate. The slope of the line is the logical effort of the gate; its intercept is the intrinsic delay. The graph shows that we can adjust the delay by adjusting the effective fan-out (by transistor sizing) or by choosing a logic gate with a different logical effort. Observe also that fan-out and logical effort contribute to the delay in a similar way. We call the product of the two $h = fg$, the *gate effort*.

The total delay of a path through a combinational logic block can now be expressed as

$$t_p = \sum_{j=1}^{N} t_{p,j} = t_{p0} \sum_{j=1}^{N} \left(p_j + \frac{f_j g_j}{\gamma}\right) \tag{6.7}$$

We use a similar procedure as we did for the inverter chain in Chapter 5 to determine the minimum delay of the path. By finding $N-1$ partial derivatives and setting them to zero, we find that **each stage should bear the same gate effort**:

$$f_1 g_1 = f_2 g_2 = \ldots = f_N g_N \tag{6.8}$$

The logical effort along a path in the network compounds by multiplying the logical efforts of all the gates along the path, yielding the *path logical effort* G:

$$G = \prod_{1}^{N} g_i \tag{6.9}$$

We also can define a *path effective fan-out* (or *electrical effort*) F, which relates the load capacitance of the last gate in the path to the input capacitance of the first gate:

$$F = \frac{C_L}{C_{g1}} \tag{6.10}$$

To relate F to the effective fan-outs of the individual gates, we must introduce another factor to account for the logical fan-out within the network. When fan-out occurs at the output of a node, some of the available drive current is directed along the path we are analyzing, and some is directed off the path. We define the *branching effort* b of a logical gate on a given path to be

$$b = \frac{C_{\text{on-path}} + C_{\text{off-path}}}{C_{\text{on-path}}} \tag{6.11}$$

where $C_{\text{on-path}}$ is the load capacitance of the gate along the path we are analyzing and $C_{\text{off-path}}$ is the capacitance of the connections that lead off the path. Note that the branching effort is, if the path does not branch (as in a chain of gates). The *path branching effort* is defined as the product of the branching efforts at each of the stages along the path, or

$$B = \prod_{1}^{N} b_i \tag{6.12}$$

The path electrical effort can now be related to the electrical and branching efforts of the individual stages:

$$F = \prod_{1}^{N} \frac{f_i}{b_i} = \frac{\prod f_i}{B} \tag{6.13}$$

Finally, the total path effort H can be defined. Using Eq. (6.13), we write

$$H = \prod_{1}^{N} h_i = \prod_{1}^{N} g_i f_i = GFB \tag{6.14}$$

From here on, the analysis proceeds along the same lines as the inverter chain. The gate effort that minimizes the path delay is

$$h = \sqrt[N]{H} \tag{6.15}$$

and the minimum delay through the path is

$$D = t_{p0}\left(\sum_{j=1}^{N} p_j + \frac{N(\sqrt[N]{H})}{\gamma}\right) \tag{6.16}$$

Note that the path intrinsic delay is a function of the types of logic gates in the path and is not affected by the sizing. The size factors of the individual gates in the chain s_i can then be derived by working from front to end (or vice versa). We assume that a unit-size gate has a driving capability equal to a minimum-size inverter. Based on the definition of the logical effort, this means that its input capacitance is g times larger than that of the reference inverter, which equals C_{ref}. With s_1 the sizing factor of the first gate in the chain, the input capacitance of the chain C_{g1}

equals $g_1 s_1 C_{ref}$. Including the branching effort, we know that the input capacitance of gate 2 is (f_1/b_1) larger, or

$$g_2 s_2 C_{ref} = \left(\frac{f_1}{b_1}\right) g_1 s_1 C_{ref} \tag{6.17}$$

For gate i in the chain, this yields

$$s_i = \left(\frac{g_1 s_1}{g_i}\right) \prod_{j=1}^{i-1} \left(\frac{f_j}{b_j}\right) \tag{6.18}$$

Example 6.6 Sizing Combinational Logic for Minimum Delay

Consider the logic network of Figure 6-19, which may represent the critical path of a more complex logic block. The output of the network is loaded with a capacitance which is five times larger than the input capacitance of the first gate, which is a minimum-sized inverter. The effective fan-out of the path thus equals $F = C_L/C_{g1} = 5$. Using the entries in Table 6-5, we find the path logical effort as follows:

$$G = 1 \times \frac{5}{3} \times \frac{5}{3} \times 1 = \frac{25}{9}$$

Since there is no branching, $B = 1$. Hence, $H = GFB = 125/9$, and the optimal stage effort h is $\sqrt[4]{H} = 1.93$. Taking into account the gate types, we derive the following fan-out factors: $f_1 = 1.93$; $f_2 = 1.93 \times (3/5) = 1.16$; $f_3 = 1.16$; $f_4 = 1.93$. Notice that the inverters are assigned larger than the more complex gates because they are better at driving loads.

Finally, we derive the gate sizes (with respect to the minimum-sized versions) using Eq. (6.18). This leads to the following values: $a = f_1 g_1/g_2 = 1.16$; $b = f_1 f_2 g_1/g_3 = 1.34$; and $c = f_1 f_2 f_3 g_1/g_4 = 2.60$.

These calculations do not have to be very precise. As discussed in Chapter 5, sizing a gate too large or too small by a factor of 1.5 still results in circuits within 5% of minimum delay. Therefore, the "back of the envelope" hand calculations using this technique are quite effective.

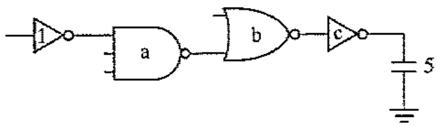


Figure 6-19 Critical path of combinational network.

Problem 6.2 Sizing an Inverter Network

Revisit Problem 5.5, but this time around use the branching-effort approach to produce the solution.

Power Consumption in CMOS Logic Gates

The sources of power consumption in a complementary CMOS inverter were discussed in detail in Chapter 5. Many of these issues apply directly to complex CMOS gates. The power dissipation is a strong function of transistor sizing (which affects physical capacitance,) input and output rise–fall times (which determine the short-circuit power,) device thresholds and temperature (which impact leakage power,) and switching activity. The dynamic power dissipation is given by $\alpha_{0\rightarrow 1}\, C_L\, V_{DD}^2\, f$. Making a gate more complex mostly affects the *switching activity* $\alpha_{0\rightarrow 1}$, which has two components: a static component that is only a function of the topology of the logic network, and a dynamic one that results from the timing behavior of the circuit. (The latter factor is also called glitching.)

Logic Function The transition activity is a strong function of the logic function being implemented. For static CMOS gates with statistically independent inputs, the static transition probability is the probability p_0 that the output will be in the *zero* state in one cycle, multiplied by the probability p_1 that the output will be in the *one* state in the next cycle:

$$\alpha_{0\rightarrow 1} = p_0 \cdot p_1 = p_0 \cdot (1 - p_0) \tag{6.19}$$

Assuming that the inputs are independent and uniformly distributed, any N-input static gate has a transition probability given by

$$\alpha_{0\rightarrow 1} = \frac{N_0}{2^N} \cdot \frac{N_1}{2^N} = \frac{N_0 \cdot (2^N - N_0)}{2^{2N}} \tag{6.20}$$

where N_0 is the number of *zero* entries, and N_1 is the number of *one* entries in the output column of the truth table of the function. To illustrate, consider a static two-input NOR gate whose truth table is shown in Table 6-6. Assume that only one input transition is possible during a clock cycle and that the inputs to the NOR gate have a uniform input distribution (in other words, the four possible states for inputs A and B—00, 01, 10, 11—are equally likely).

Table 6-6 Truth table of a two-input NOR gate.

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

From Table 6-6 and Eq. (6.20), the output transition probability of a two-input static CMOS NOR gate can be derived:

$$\alpha_{0\to 1} = \frac{N_0 \cdot (2^N - N)}{2^{2N}} = \frac{3 \cdot (2^2 - 3)}{2^{2 \cdot 2}} = \frac{3}{16} \tag{6.21}$$

Problem 6.3 ***N*-Input XOR Gate**

Assuming the inputs to an *N*-input XOR gate are uncorrelated and uniformly distributed, derive the expression for the switching activity factor.

Signal Statistics The switching activity of a logic gate is a strong function of the input signal statistics. Using a uniform input distribution to compute activity is not a good technique, since the propagation through logic gates can significantly modify the signal statistics. For example, consider once again a two-input static NOR gate, and let p_a and p_b be the probabilities that the inputs *A* and *B* are *one*. Assume further that the inputs are not correlated. The probability that the output node is 1 is given by

$$p_1 = (1 - p_a)(1 - p_b) \tag{6.22}$$

Therefore, the probability of a transition from 0 to 1 is

$$\alpha_{0\to 1} = p_0 p_1 = (1 - (1 - p_a)(1 - p_b))(1 - p_a)(1 - p_b) \tag{6.23}$$

Figure 6-20 shows the transition probability as a function of p_a and p_b. Observe how this graph degrades into the simple inverter case when one of the input probabilities is set to 0. From

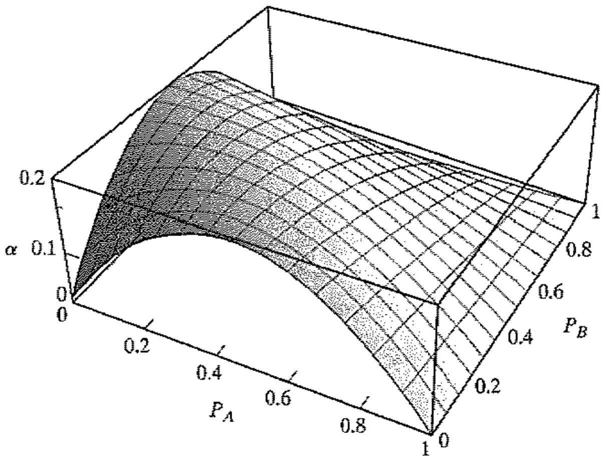


Figure 6-20 Transition activity of a two-input NOR gate as a function of the input probabilities (p_A, p_B).

this plot, it is clear that understanding the signal statistics and their impact on switching events can be used to significantly impact the power dissipation.

Problem 6.4 Power Dissipation of Basic Logic Gates

Derive the $0 \rightarrow 1$ output transition probabilities for the basic logic gates (AND, OR, XOR). The results to be obtained are given in Table 6-7.

Table 6-7 Output transition probabilities for static logic gates.

	$\alpha_{0\rightarrow 1}$
AND	$(1-p_A p_B)p_A p_B$
OR	$(1-p_A)(1-p_B)[1-(1-p_A)(1-p_B)]$
XOR	$[1-(p_A+p_B-2p_A p_B)](p_A+p_B-2p_A p_B)$

Intersignal Correlations The evaluation of the switching activity is further complicated by the fact that signals exhibit correlation in space and time. Even if the primary inputs to a logic network are uncorrelated, the signals become correlated or "colored," as they propagate through the logic network. This is best illustrated with a simple example. Consider first the circuit shown in Figure 6-21a, and assume that the primary inputs A and B are uncorrelated and uniformly distributed. Node C has a **1** (**0**) probability of 1/2, and a $0 \rightarrow 1$ transition probability of 1/4. The probability that the node Z undergoes a power consuming transition is then determined using the AND-gate expression of Table 6-7:

$$p_{0\rightarrow 1} = (1 - p_a p_b)\, p_a p_b = (1 - 1/2 \cdot 1/2)\ 1/2 \cdot 1/2 = 3/16 \tag{6.24}$$

The computation of the probabilities is straightforward: signal and transition probabilities are evaluated in an ordered fashion, progressing from the input to the output node. This approach, however, has two major limitations: (1) it does not deal with circuits with feedback as found in sequential circuits, and (2) it assumes that the signal probabilities at the input of each gate are independent. This is rarely the case in actual circuits, where reconvergent fan-out often causes intersignal dependencies. For instance, the inputs to the AND gate in Figure 6-21b (C and B) are interdependent because both are a function of A. The approach to computing

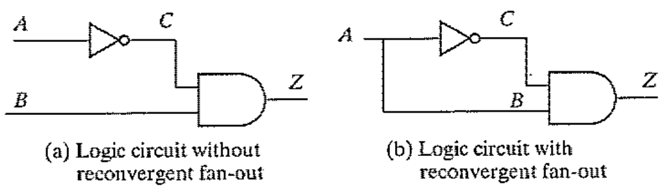

A
C
B
Z
A
C
B
Z
(a) Logic circuit without reconvergent fan-out
(b) Logic circuit with reconvergent fan-out

Figure 6-21 Example illustrating the effect of signal correlations.

probabilities that we presented previously fails under these circumstances. Traversing from inputs to outputs yields a transition probability of 3/16 for node Z, similar to the previous analysis. This value clearly is false, as logic transformations show that the network can be reduced to $Z = C \cdot B = A \cdot \overline{A} = 0$, and thus no transition will ever take place.

To get the precise results in the progressive analysis approach, its is essential to take signal interdependencies into account. This can be accomplished with the aid of conditional probabilities. For an AND gate, Z equals 1 if and only if B and C are equal to 1. Thus,

$$p_Z = p(Z = 1) = p(B = 1, C = 1) \tag{6.25}$$

where $p(B = 1, C = 1)$ represents the probability that B and C are equal to 1 simultaneously. If B and C are independent, $p(B = 1, C = 1)$ can be decomposed into $p(B = 1) \cdot p(C = 1)$, and this yields the expression for the AND gate derived earlier: $p_Z = p(B = 1) \cdot p(C = 1) = p_B\, p_C$. If a dependency between the two exists (as is the case in Figure 6-21b), a conditional probability has to be employed, such as the following:

$$p_Z = p(C = 1|B = 1) \cdot p(B = 1) \tag{6.26}$$

The first factor in Eq. (6.26) represents the probability that $C = 1$ given that $B = 1$. The extra condition is necessary because C is dependent upon B. Inspection of the network shows that this probability is equal to 0, since C and B are logical inversions of each other, resulting in the signal probability for Z, $p_Z = 0$.

Deriving those expressions in a structured way for large networks with reconvergent fan-out is complex, especially when the networks contain feedback loops. Computer support is therefore essential. To be meaningful, the analysis program has to process a typical sequence of input signals, because the power dissipation is a strong function of statistics of those signals.

Dynamic or Glitching Transitions When analyzing the transition probabilities of complex, multistage logic networks in the preceding section, we ignored the fact that the gates have a non-zero propagation delay. In reality, the finite propagation delay from one logic block to the next can cause spurious transitions known as *glitches or dynamic hazards* to occur: a node can exhibit multiple transitions in a single clock cycle before settling to the correct logic level.

A typical example of the effect of glitching is shown in Figure 6-22, which displays the simulated response of a chain of NAND gates for all inputs going simultaneously from 0 to 1. Initially, all the outputs are 1 since one of the inputs was 0. For this particular transition, all the odd bits must transition to 0, while the even bits remain at the value of 1. However, due to the finite propagation delay, the even output bits at the higher bit positions start to discharge, and the voltage drops. When the correct input ripples through the network, the output goes high. The glitch on the even bits causes extra power dissipation beyond what is required to strictly implement the logic function. Although the glitches in this example are only partial (i.e., not from rail to rail), they contribute significantly to the power dissipation. Long chains of gates often occur in important structures such as adders and multipliers, and the glitching component can easily dominate the overall power consumption.

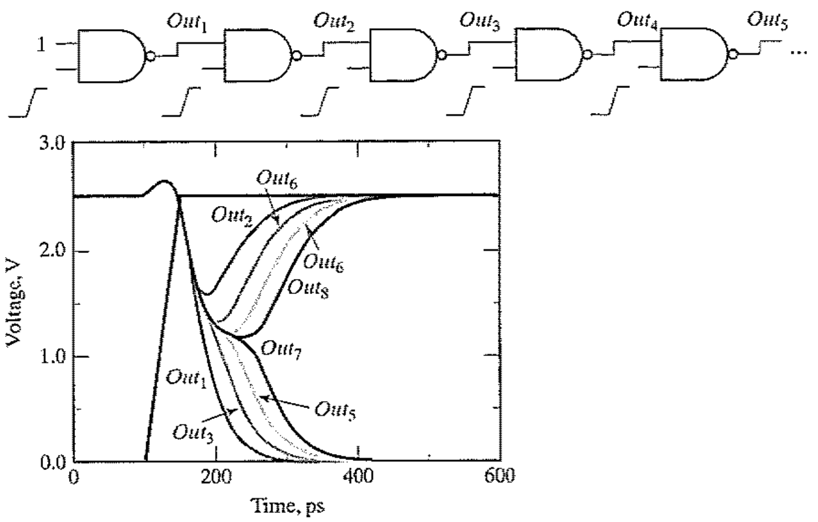


Figure 6-22 Glitching in a chain of NAND gates.

Design Techniques to Reduce Switching Activity

The dynamic power of a logic gate can be reduced by minimizing the physical capacitance and the switching activity. The physical capacitance can be minimized in a number ways, including circuit style selection, transistor sizing, placement and routing, and architectural optimizations. The switching activity, on the other hand, can be minimized at all levels of the design abstraction, and is the focus of this section. Logic structures can be optimized to minimize both the fundamental transitions required to implement a given function and the spurious transitions.

1. **Logic Restructuring** Changing the topology of a logic network may reduce its power dissipation. Consider, for example, two alternative implementations of $F = A \cdot B \cdot C \cdot D$, as shown in Figure 6-23. Ignore glitching and assume that all primary inputs (A,B,C,D) are uncorrelated and uniformly distributed (this is, $p_{1\ (a,b,c,d)} = 0.5$). Using the expressions from Table 6-7, the activity can be computed for the two topologies, as shown in Table 6-8. The results indicate that the chain implementation has an overall lower switching activity than the tree implementation for random inputs. However, as mentioned before, it is also important to consider the timing behavior to

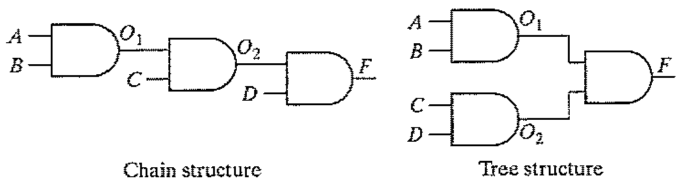


Figure 6-23 Simple example to demonstrate the influence of circuit topology on activity.

Table 6-8 Probabilities for tree and chain topologies.

	O_1	O_2	F
p_1 (chain)	1/4	1/8	1/16
$p_0 = 1-p_1$ (chain)	3/4	7/8	15/16
$p_{0\text{->}1}$ (chain)	3/16	7/64	15/256
p_1 (tree)	1/4	1/4	1/16
$p_0 = 1-p_1$ (tree)	3/4	3/4	15/16
$p_{0\text{->}1}$ (tree)	3/16	3/16	15/256

accurately make power trade-offs. In this example, the tree topology experiences (virtually) no glitching activity since the signal paths are balanced to all the gates.

2. **Input ordering** Consider the two static logic circuits of Figure 6-24. The probabilities that A, B, and C are equal to **1** are listed in the Figure. Since both circuits implement identical logic functionality, it is clear that the activity at the output node Z is equal in both cases. The difference is in the activity at the intermediate node. In the first circuit, this activity equals $(1 - 0.5 \times 0.2)(0.5 \times 0.2) = 0.09$. In the second case, the probability that a $0 \rightarrow 1$ transition occurs equals $(1 - 0.2 \times 0.1)(0.2 \times 0.1) = 0.0196$, a substantially lower value. From this, we learn that it is beneficial to postpone the introduction of signals with a high transition rate (i.e., signals with a signal probability close to 0.5). A simple reordering of the input signals is often sufficient to accomplish that goal.

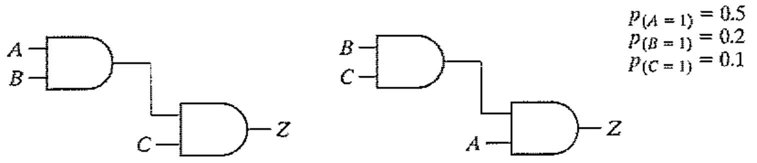
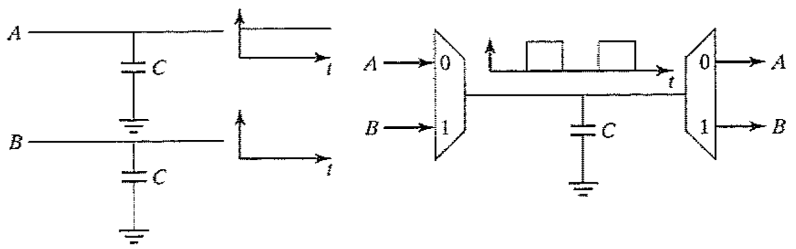


Figure 6-24 Reordering of inputs affects the circuit activity.

3. **Time-multiplexing resources** Time-multiplexing a single hardware resource—such as a logic unit or a bus—over a number of functions is a technique often used to minimize the implementation area. Unfortunately, the minimum area solution does not always result in the lowest switching activity. For example, consider the transmission of two input bits (A and B) using either dedicated resources or a time-multiplexed approach, as shown in Figure 6-25. To the first order, ignoring the multiplexer overhead, it would seem that the degree of time multiplexing should not affect the switched capacitance, since the time-multiplexed solution has half the physical capacitance switched at twice the frequency (for a fixed throughput).

   If the data being transmitted are random, it will make no difference which architecture is used. However, if the data signals have some distinct properties (such as temporal correlation), the power dissipation of the time-multiplexed solution can be significantly higher. Suppose, for instance, that A is always (or mostly) 1, and B is (mostly) 0. In the parallel solution, the switched capacitance is very low since there are very few transitions on the data bits. However, in the time-multiplexed solution,

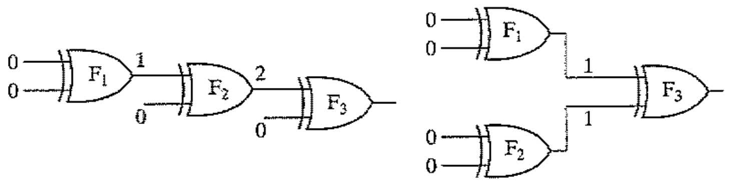


(a) Parallel data transmission (b) Serial data transmission

Figure 6-25 Parallel versus time-multiplexed data busses.

the bus toggles between 0 and 1. Care must be taken in digital systems to avoid time-multiplexing data streams with very distinct data characteristics.

4. **Glitch Reduction by balancing signal paths** The occurrence of glitching in a circuit is mainly due to a mismatch in the path lengths in the network. If all input signals of a gate change simultaneously, no glitching occurs. On the other hand, if input signals change at different times, a dynamic hazard might develop. Such a mismatch in signal timing is typically the result of different path lengths with respect to the primary inputs of the network. This is illustrated in Figure 6-26. Assume that the XOR gate has a unit delay. The first network (a) suffers from glitching as a result of the wide disparity between the arrival times of the input signals for a gate. For example, for gate F_3, one input settles at time 0, while the second one only arrives at time 2. Redesigning the network so that all arrival times are identical can dramatically reduce the number of superfluous transitions (network b).



(a) Network sensitive to glitching (b) Glitch-free network

Figure 6-26 Glitching is influenced by matching of signal path lengths. The annotated numbers indicate the signal arrival times.

■

Summary

The CMOS logic style described in the previous section is highly robust and scalable with technology, but requires $2N$ transistors to implement an N-input logic gate. Also, the load capacitance is significant, since each gate drives two devices (a PMOS and an NMOS) per *fan-out*. This has opened the door for alternative logic families that either are simpler or faster.

6.2.2 Ratioed Logic

Concept

Ratioed logic is an attempt to reduce the number of transistors required to implement a given logic function, often at the cost of reduced robustness and extra power dissipation. The purpose

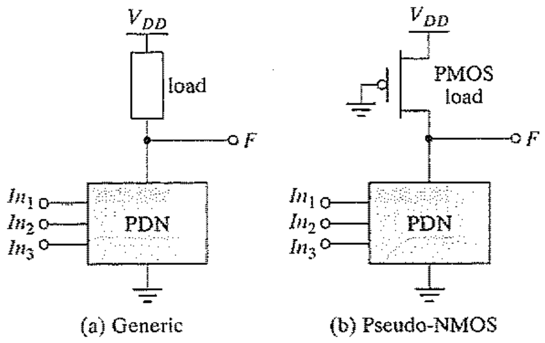


Figure 6-27 Ratioed logic gate.

of the PUN in complementary CMOS is to provide a conditional path between V_{DD} and the output when the PDN is turned *off*. In ratioed logic, the entire PUN is replaced with a single unconditional load device that pulls up the output for a high output as in Figure 6-27a. Instead of a combination of active pull-down and pull-up networks, such a gate consists of an NMOS pull-down network that realizes the *logic function*, and a simple *load device*. Figure 6-27b shows an example of ratioed logic, which uses a grounded PMOS load and is referred to as a pseudo-NMOS gate.

The clear advantage of a pseudo-NMOS gate is the reduced number of transistors ($N + 1$, versus $2N$ for complementary CMOS). The nominal high output voltage (V_{OH}) for this gate is V_{DD} since the pull-down devices are turned *off* when the output is pulled high (assuming that V_{OL} is below V_{Tn}). On the other hand, the **nominal low output voltage is not 0 V**, since there is contention between the devices in the PDN and the grounded PMOS load device. This results in reduced noise margins and, more importantly, static power dissipation. The sizing of the load device relative to the pull-down devices can be used to trade off parameters such as *noise margin*, *propagation delay*, and *power dissipation*. Since the voltage swing on the output and the overall functionality of the gate depend on the ratio of the NMOS and PMOS sizes, the circuit is called *ratioed*. This is in contrast to the *ratioless* logic styles, such as complementary CMOS, where the low and high levels do not depend on transistor sizes.

Computing the dc-transfer characteristic of the pseudo-NMOS proceeds along paths similar to those used for its complementary CMOS counterpart. The value of V_{OL} is obtained by equating the currents through the driver and load devices for $V_{in} = V_{DD}$. At this operation point, it is reasonable to assume that the NMOS device resides in linear mode (since, ideally, the output should be close to 0V), while the PMOS load is saturated:

$$k_n\left((V_{DD} - V_{Tn})V_{OL} - \frac{V_{OL}^2}{2}\right) + k_p\left((-V_{DD} - V_{Tp}) \cdot V_{DSATp} - \frac{V_{DSATp}^2}{2}\right) = 0 \qquad (6.27)$$

Assuming that V_{OL} is small relative to the gate drive ($V_{DD} - V_T$), and that V_{Tn} is equal to V_{Tp} in magnitude, V_{OL} can be approximated as

$$V_{OL} \approx \frac{k_p(V_{DD}+V_{Tp}) \cdot V_{DSATp}}{k_n(V_{DD}-V_{Tn})} \approx \frac{\mu_p \cdot W_p}{\mu_n \cdot W_n} \cdot V_{DSATp} \tag{6.28}$$

In order to make V_{OL} as small as possible, the PMOS device should be sized much smaller than the NMOS pull-down devices. Unfortunately, this has a negative impact on the *propagation delay* for charging up the output node since the current provided by the PMOS device is limited.

A major disadvantage of the pseudo-NMOS gate is the static power that is dissipated when the output is low through the direct current path that exists between V_{DD} and GND. The static power consumption in the low-output mode is easily derived:

$$P_{low} = V_{DD} I_{low} \approx V_{DD} \cdot \left| k_p\left((-V_{DD}-V_{Tp}) \cdot V_{DSATp} - \frac{V_{DSATp}^2}{2}\right) \right| \tag{6.29}$$

Example 6.7 Pseudo-NMOS Inverter

Consider a simple pseudo-NMOS inverter (where the PDN network in Figure 6-27 degenerates to a single transistor) with an NMOS size of 0.5 μm/0.25 μm. In this example, we study the effect of sizing the PMOS device to demonstrate the impact on various parameters. The *W–L* ratio of the grounded PMOS is varied over values from 4, 2, 1, 0.5 to 0.25. Devices with a *W–L* < 1 are constructed by making the length greater than the width. The voltage transfer curve for the different sizes is plotted in Figure 6-28.

Table 6-9 summarizes the nominal output voltage (V_{OL}), static power dissipation, and the low-to-high propagation delay. The low-to-high delay is measured as the time it takes to reach 1.25 V from V_{OL} (which is not 0V for this inverter)—by definition. The trade-off between the static and dynamic properties is apparent. A larger pull-up device not only improves performance, but also increases static power dissipation and lowers noise margins by increasing V_{OL}.

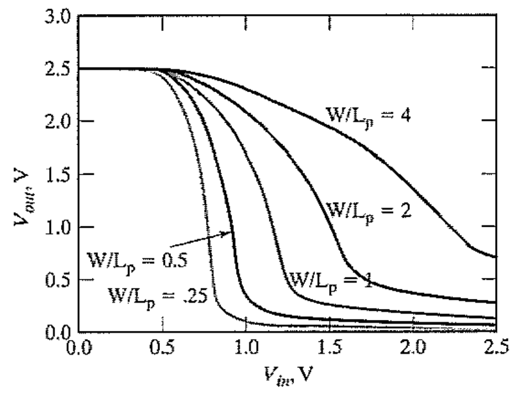


Figure 6-28 Voltage-transfer curves of the pseudo-NMOS inverter as a function of the PMOS size.

Table 6-9 Performance of a pseudo-NMOS inverter.

Size	V_{OL}	Static Power Dissipation	t_{plh}
4	0.693 V	564 μW	14 ps
2	0.273 V	298 μW	56 ps
1	0.133 V	160 μW	123 ps
0.5	0.064 V	80 μW	268 ps
0.25	0.031 V	41 μW	569 ps

Notice that the simple first-order model to predict V_{OL} is quite effective. For a PMOS W–L of 4, V_{OL} is given by (30/115) (4) (0.63V) = 0.66V.

The static power dissipation of pseudo-NMOS limits its use. When area is most important however, its reduced transistor count compared with complementary CMOS is quite attractive. Pseudo-NMOS thus still finds occasional use in large fan-in circuits. Figure 6-29 shows the schematics of pseudo-NMOS NOR and NAND gates.

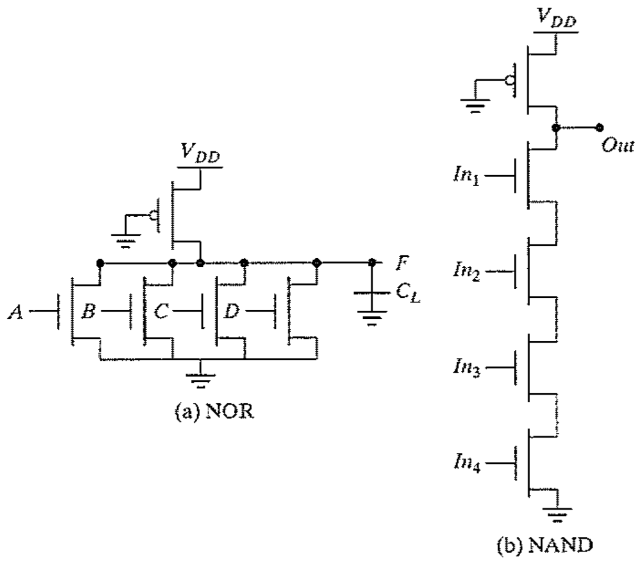


Figure 6-29 Four-input pseudo-NMOS NOR and NAND gates.

Problem 6.5 NAND versus NOR in Pseudo-NMOS

Given the choice between NOR or NAND logic, which one would you prefer for implementation in pseudo-NMOS?

How to Build Even Better Loads

It is possible to create a ratioed logic style that completely eliminates static currents and provides rail-to-rail swing. Such a gate combines two concepts: *differential logic* and *positive feedback*. A differential gate requires that each input is provided in complementary format, and it produces complementary outputs in turn. The feedback mechanism ensures that the load device is turned off when not needed. An example of such a logic family, called *Differential Cascode Voltage Switch Logic* (or DCVSL), is presented conceptually in Figure 6-30a [Heller84].

The pull-down networks PDN1 and PDN2 use NMOS devices and are mutually exclusive—that is, when PDN1 conducts, PDN2 is off, and when PDN1 is off, PDN2 conducts—such that the required logic function and its inverse are simultaneously implemented. Assume now that, for a given set of inputs, PDN1 conducts while PDN2 does not, and that *Out* and $\overline{Out}$ are initially high and low, respectively. Turning *on PDN1,* causes *Out* to be pulled down, although there is still contention between M_1 and PDN1. $\overline{Out}$ is in a high impedance state, as M_2 and PDN2 are both turned *off*. PDN1 must be strong enough to bring *Out* below $V_{DD} - |V_{Tp}|$, the point at which M_2 turns *on* and starts charging $\overline{Out}$ to V_{DD}, eventually turning off M_1. This in turn enables *Out* to discharge all the way to GND. Figure 6-30b shows an example of an XOR–XNOR gate. Notice that it is possible to share transistors among the two pull-down networks, which reduces the implementation overhead.

The resulting circuit exhibits a rail-to-rail swing, and the static power dissipation is eliminated: in steady state, none of the stacked pull-down networks and load devices are

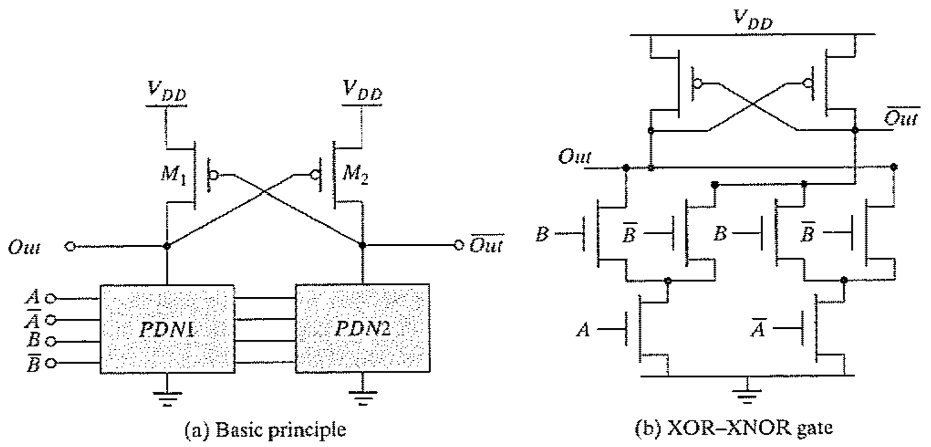
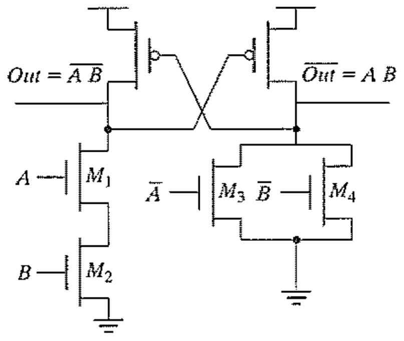


Figure 6-30 DCVSL logic gate.

simultaneously conducting. However, the circuit is still ratioed since the sizing of the PMOS devices relative to the pull-down devices is critical to functionality, not just performance. In addition to the problem of increased design complexity, this circuit style has a power-dissipation problem that is due to cross-over currents. During the transition, there is a period of time when PMOS and PDN are turned on simultaneously, producing a short circuit path.

Example 6.8 DCVSL Transient Response

An example transient response is shown in Figure 6.31 for an AND/NAND gate in DCVSL. Notice that as *Out* is pulled down to $V_{DD} - |V_{Tp}|$, $\overline{Out}$ starts to charge up to V_{DD} quickly. The delay from the input to *Out* is 197 ps and to $\overline{Out}$ is 321 ps. A static CMOS AND gate (NAND followed by an inverter) has a delay of 200 ps.



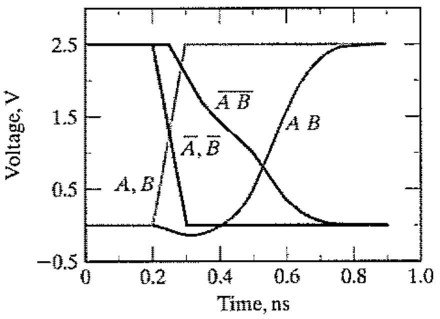


Figure 6-31 Transient response of a simple AND/NAND DCVSL gate. M_1 and M_2 1 μm/0.25 μm, M_3 and M_4 are 0.5 μm/0.25 μm and the cross-coupled PMOS devices are 1.5 μm/0.25 μm.

Design Consideration—Single-Ended versus Differential

The DCVSL gate provides *differential* (or *complementary*) *outputs*. Both the output signal (V_{out}) and its inverted value ($\overline{V}_{out}$) are simultaneously available. This is a distinct advantage, because it eliminates the need for an extra inverter to produce the complementary signal. It has been observed that a differential implementation of a complex function may reduce the number of gates required by a factor of two! The number of gates in the critical timing path is often reduced as well. Finally, the approach prevents some of the time-differential problems introduced by additional inverters. For example, in logic design, it often happens that both a signal and its complement are needed simultaneously. When the complementary signal is generated using an inverter, the inverted signal is delayed with respect to the original (Figure 6-32a). This causes timing problems, especially in very high-speed designs. Logic families with differential output capability avoid this problem to a major extent, if not completely (Figure 6-32b).

With all these positive properties, why not always use differential logic? The reason is that the differential nature virtually doubles the number of wires that have to be routed, often leading to unwieldy designs on top of the additional implementation overhead in the individual gates. The dynamic power dissipation also is high.

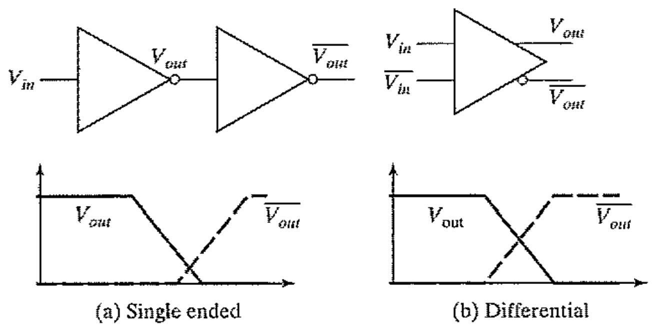


Figure 6-32 Advantage of over single-ended (a) differential (b) gate.

6.2.3 Pass-Transistor Logic

Pass-Transistor Basics

A popular and widely used alternative to complementary CMOS is *pass-transistor logic*, which attempts to reduce the number of transistors required to implement logic by allowing the primary inputs to drive gate terminals as well as source–drain terminals [Radhakrishnan85]. This is in contrast to logic families that we have studied so far, which only allow primary inputs to drive the gate terminals of MOSFETS.

Figure 6-33 shows an implementation of the AND function constructed that way, using only NMOS transistors. In this gate, if the B input is high, the top transistor is turned on and copies the input A to the output F. When B is low, the bottom pass-transistor is turned on and passes a 0. The switch driven by $\overline{B}$ seems to be redundant at first glance. Its presence is essential to ensure that the gate is static—a low-impedance path must exist to the supply rails under all circumstances (in this particular case, when B is low).

The promise of this approach is that fewer transistors are required to implement a given function. For example, the implementation of the AND gate in Figure 6-33 requires 4 transistors (including the inverter required to invert B), while a complementary CMOS implementation would require 6 transistors. The reduced number of devices has the additional advantage of lower capacitance.

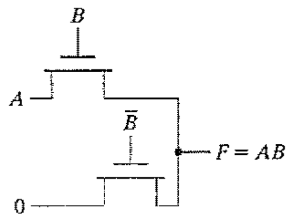


Figure 6-33 Pass-transistor implementation of an AND gate.

Unfortunately, as discussed earlier, an NMOS device is effective at passing a 0, but it is poor at pulling a node to V_{DD}. When the pass-transistor pulls a node high, the output only charges up to $V_{DD} - V_{Tn}$. In fact, the situation is worsened by the fact that the devices experience body effect, because a significant source-to-body voltage is present when pulling high. Consider the case in which the pass-transistor is charging up a node with the gate and drain terminals set at V_{DD}. Let the source of the NMOS pass-transistor be labeled x. The node x will charge up to $V_{DD} - V_{Tn}(V_x)$. We obtain

$$V_x = V_{DD} - (V_{Tn0} + \gamma(\sqrt{|2\phi_f| + V_x} - \sqrt{|2\phi_f|})) \tag{6.30}$$

Example 6.9 Voltage Swing for Pass-Transistors Circuits

The transient response of Figure 6-34 shows an NMOS charging up a capacitor. The drain voltage of the NMOS is at V_{DD}, and its gate voltage is being ramped from 0 V to V_{DD}. Assume that node x is initially at 0 V. We observe that the output initially charges up quickly, but the tail end of the transient is slow. The current drive of the transistor (gate-to-source voltage) is reduced significantly as the output approaches $V_{DD} - V_{Tn}$, and the current available to charge up node x is reduced drastically. Manual calculation using Eq. (6.30) results in an output voltage of 1.8 V, which is close to the simulated value.

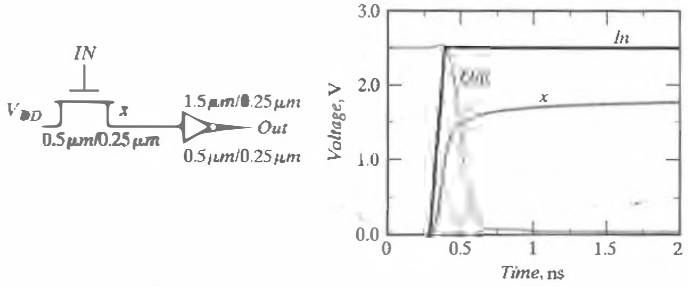


Figure 6-34 Transient response of charging up a node using an N device. Notice the slow tail after an initial quick response. $V_{DD} = 2.5$ V.

WARNING: The preceding example demonstrates that **pass-transistor gates cannot be cascaded by connecting the output of a pass gate to the gate input of another pass-transistor.** This is illustrated in Figure 6-35a, where the output of M_1 (node x) drives the gate of another MOS device. Node x can charge up to $V_{DD} - V_{Tn1}$. If node C has a rail-to-rail swing, node Y only charges up to the voltage on node $x - V_{Tn2}$, which works out to $V_{DD} - V_{Tn1} - V_{Tn2}$. Figure 6-35b, on the other hand, has the output of M_1 (x) driving the junction of M_2, and there is only one threshold drop. This is the proper way of cascading pass gates.

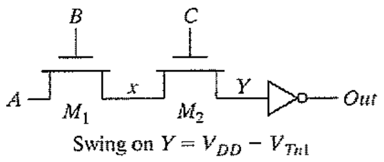
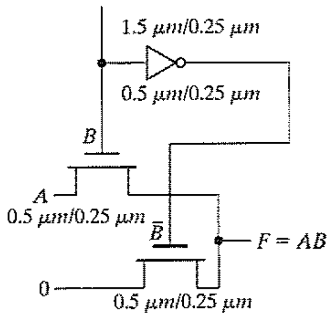


Figure 6-35 Pass-transistor output (drain–source) terminal should not drive other gate terminals to avoid multiple threshold drops.

Example 6.10 VTC of the Pass-Transistor AND Gate

The voltage transfer curve of a pass-transistor gate shows little resemblance to complementary CMOS. Consider the AND gate shown in Figure 6-36. Similar to complementary CMOS, the VTC of pass-transistor logic is data dependent. For the case when $B = V_{DD}$, the top pass-transistor is turned *on*, while the bottom one is turned *off*. In this case, the output just follows the input A until the input is high enough to turn *off* the top pass-transistor (i.e., reaches $V_{DD} - V_{Tn}$). Next, consider the case in which $A = V_{DD}$, and B makes a transition from $0 \to 1$. Since the inverter has a threshold of $V_{DD}/2$, the bottom pass-transistor is turned *on* until then and the output remains close to zero. Once the bottom pass-transistor turns *off*, the output follows the input B minus a threshold drop. A similar behavior is observed when both inputs A and B transition from $0 \to 1$.

Observe that a pure pass-transistor gate is not regenerative. A gradual signal degradation will be observed after passing through a number of subsequent stages. This can be remedied by the occasional insertion of a CMOS inverter. With the inclusion of an inverter in the signal path, the VTC resembles one of the CMOS gates.



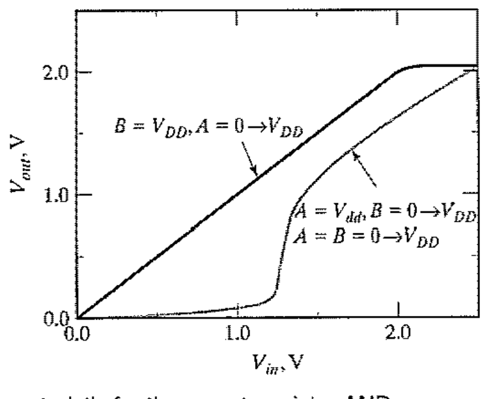


Figure 6-36 Voltage transfer characteristic for the pass-transistor AND gate of Figure 6-33.

Pass-transistors require lower switching energy to charge up a node, due to the reduced voltage swing. For the pass-transistor circuit in Figure 6-34, assume that the drain voltage is at

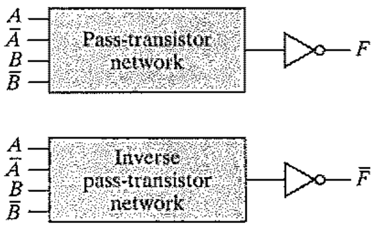
V_{DD} and the gate voltage transitions to V_{DD}. The output node charges from 0V to $V_{DD} - V_{Tn}$ (assuming that node x was initially at 0V), and the energy drawn from the power supply for charging the output of a pass-transistor is given by

$$E_{0\rightarrow 1} = \int_0^T P(t)dt = V_{DD}\int_0^T i_{supply}(t)dt$$
$$= V_{DD}\int_0^{(V_{DD}-V_{Tn})} C_L dV_{out} = C_L \cdot V_{DD} \cdot (V_{DD} - V_{Tn}) \quad (6.31)$$

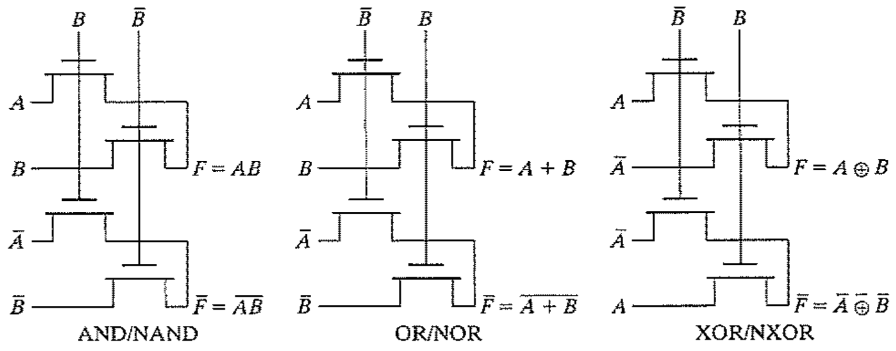
While the circuit exhibits lower switching power, it may also consume static power when the output is high—the reduced voltage level may be insufficient to turn off the PMOS transistor of the subsequent CMOS inverter.

Differential Pass-Transistor Logic

For high performance design, a differential pass-transistor logic family, called *CPL* or *DPL*, is commonly used. The basic idea (similar to DCVSL) is to accept true and complementary inputs and produce true and complementary outputs. Several CPL gates (AND/NAND, OR/NOR, and XOR/NXOR) are shown in Figure 6-37. These gates possess some interesting properties:



(a) Basic concept



(b) Example pass-transistor networks

Figure 6-37 Complementary pass-transistor logic (CPL).

- Since the circuits are *differential*, complementary data inputs and outputs are always available. Although generating the differential signals requires extra circuitry, the differential style has the advantage that some complex gates such as XORs and adders can be realized efficiently with a small number of transistors. Furthermore, the availability of both polarities of every signal eliminates the need for extra inverters, as is often the case in static CMOS or pseudo-NMOS.
- CPL belongs to the class of *static* gates, because the output-defining nodes are always connected to either V_{DD} or GND through a low-resistance path. This is advantageous for the noise resilience.
- The design is very modular. In effect, all gates use exactly the same topology. Only the inputs are permutated. This makes the design of a library of gates very simple. More complex gates can be built by cascading the standard pass-transistor modules.

Example 6.11 Four-Input NAND in CPL

Consider the implementation of a four-input AND/NAND gate using CPL. Based on the associativity of the boolean AND operation $[A \cdot B \cdot C \cdot D = (A \cdot B) \cdot (C \cdot D)]$, a two-stage approach has been adopted to implement the gate (Figure 6-38). The total number of transistors in the gate (including the final buffer) is 14. This is substantially higher than previously discussed gates.[3] This factor, combined with the complicated routing requirements, makes this circuit style not particularly efficient for this gate. One should, however, be aware of the fact that the structure simultaneously implements the AND and the NAND functions, which might reduce the transistor count of the overall circuit.

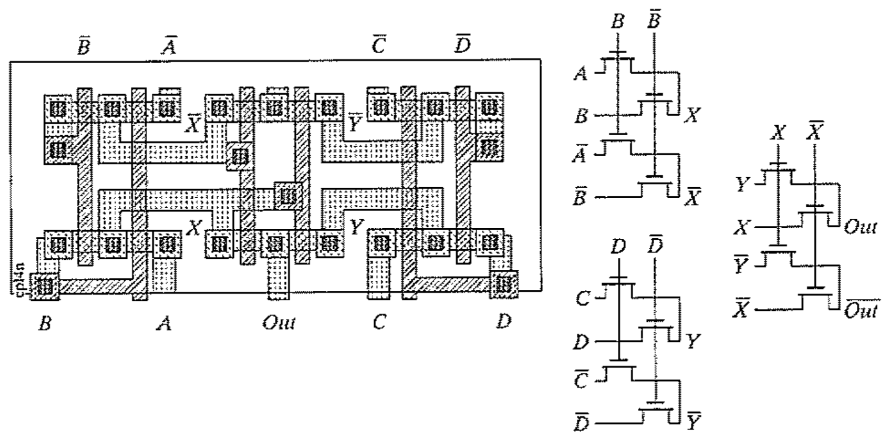


Figure 6-38 Layout and schematics of four-input NAND gate using CPL. The final inverter stage is omitted.

[3]This particular circuit configuration is only acceptable when zero-threshold pass-transistors are used. If not, it directly violates the concepts introduced in Figure 6-35.

In sum, CPL is a conceptually simple and modular logic style. Its applicability depends strongly on the logic function to be implemented. The availability of a simple XOR and the ease of implementing multiplexers makes it attractive for structures such as adders and multipliers. Some extremely fast and efficient implementations have been reported in that application domain [Yano90]. When considering CPL, the designer should not ignore the implicit routing overhead of the complementary signals, which is apparent in the layout of Figure 6-38.

Robust and Efficient Pass-Transistor Design

Unfortunately, differential pass-transistor logic, like single-ended pass-transistor logic, suffers from static power dissipation and reduced noise margins, since the high input to the signal-restoring inverter only charges up to $V_{DD} - V_{Tn}$. There are several solutions proposed to deal with this problem, outlined as follows:

Solution 1: Level Restoration A common solution to the voltage drop problem is the use of a *level restorer*, which is a single PMOS configured in a feedback path (see Figure 6-39). The gate of the PMOS device is connected to the output of the inverter its drain is connected to the input of the inverter and the source is connected to V_{DD}. Assume that node X is at 0V (*out* is at V_{DD} and the M_r is turned *off*) with $B = V_{DD}$ and $A = 0$. If input A makes a 0 to V_{DD} transition, M_n only charges up node X to $V_{DD} - V_{Tn}$. This is, however, enough to switch the output of the inverter low, turning on the feedback device M_r and pulling node X all the way to V_{DD}. This eliminates any static power dissipation in the inverter. Furthermore, no static current path can exist through the level restorer and the pass-transistor, since the restorer is only active when A is high. In sum, this circuit has the advantage that all voltage levels are either at GND or V_{DD}, and no static power is consumed.

While this solution is appealing in terms of eliminating static power dissipation, it adds complexity since the circuit is ratioed. The problem arises during the transition of node X from high to low (seeFigure 6-40). The pass-transistor network attempts to pull down node X, while

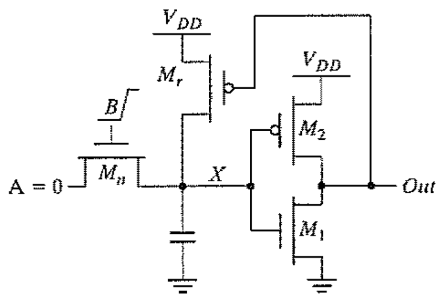


Figure 6-39 Transistor-sizing problem in level-restoring circuits.

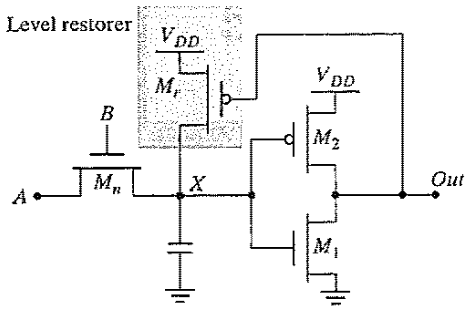


Figure 6-40 Level-restoring circuit.

the level restorer pulls X to V_{DD}. Therefore, the pull-down network, represented by M_n, must be stronger than the pull-up device Mr to switch node X (and the output). Careful transistor sizing is necessary to make the circuit function correctly. Assume the notation R_1 to denote the equivalent on-resistance of transistor M_1, R_2 for M_2, and R_r for M_r. When R_r is too small, it is impossible to bring the voltage at node X below the switching threshold of the inverter. Hence, the inverter output never switches to V_{DD}, and the gate is locked in a single state. The problem can be resolved by sizing transistors M_n and M_r such that the voltage at node X drops below the threshold of the inverter V_M, which is a function of R_1 and R_2. This condition is sufficient to guarantee the switching of the output voltage V_{out} to V_{DD} and the turning off of the level-restoring transistor.

Example 6.12 Sizing of a Level Restorer

Analyzing the circuit as a whole is nontrivial, because the restoring transistor acts as a feedback device. One way to simplify the circuit for manual analysis is to open the feedback loop and to ground the gate of the restoring transistor when determining the switching point (this is a reasonable assumption, as the feedback only becomes active once the inverter starts to switch). Hence, M_r and M_n form a configuration that resembles pseudo-NMOS with M_r the load transistor, and M_n acting as a pull-down network to GND. Assume that the inverter M_1, M_2 is sized to have its switching threshold at $V_{DD}/2$ (NMOS: 0.5 µm/0.25 µm and PMOS: 1.5 µm/0.25 µm). Therefore, node X must be pulled below $V_{DD}/2$ to switch the inverter and to shut off M_r.

This is confirmed in Figure 6-41, which shows the transient response as the size of the level restorer is varied, while keeping the size of M_n fixed (0.5 µm/0.25 µm). As the simulation indicates, for sizes above 1.5 µm/0.25 µm, node X cannot be brought below the switching threshold of the inverter, and can't switch the output.

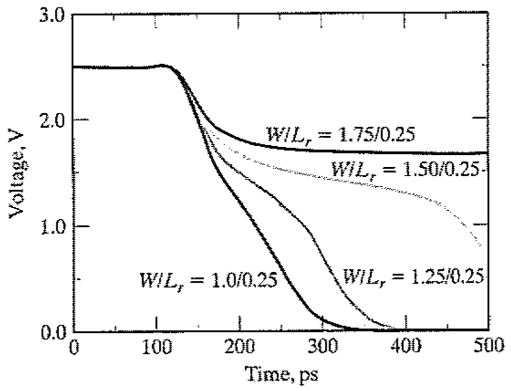


Figure 6-41 Transient response of the circuit in Figure 6-39. A level restorer that is too large results in incorrect evaluation.

Another concern is the influence of the level restorer on the switching speed of the device. Adding the restoring device increases the capacitance at the internal node X, slowing down the gate. In addiction, the rise time of the gate is affected negatively. The level restoring transistor M_r fights the decrease in voltage at node X before being switched off. On the other hand, the level restorer reduces the fall time, since the PMOS transistor, once turned on, accelerates the pull-up action.

Problem 6.6 Device Sizing in Pass-Transistors

For the circuit shown in Figure 6-39, assume that the pull-down device consists of six pass-transistors in series each with a device size of 0.5 µm/0.25 µm (replacing transistor M_n). Determine the maximum W–L size for the level restorer transistor for correct functionality.

A modification of the level restorer concept is shown in Figure 6-42. It is applicable in differential networks and is known as *swing-restored pass-transistor logic*. Instead of a simple inverter at the output of the pass-transistor network, two back-to-back inverters configured in a cross-coupled fashion are used for level restoration and performance improvement. Inputs are fed to both the gate and source–drain terminals, as in the case of conventional pass-transistor networks. Figure 6-42 shows a simple XOR/XNOR gate of three variables A, B, and C. The complementary network can be optimized by sharing transistors between the true and complementary outputs. This logic family comes with a major caveat: When cascading gates, buffers may have to be included in between the gates. If not, contention between the level-restoring devices of the cascaded gates negatively impacts the performance.

Solution 2: Multiple-Threshold Transistors A technology solution to the voltage-drop problem associated with pass-transistor logic is the use of multiple-threshold devices. Using *zero-*

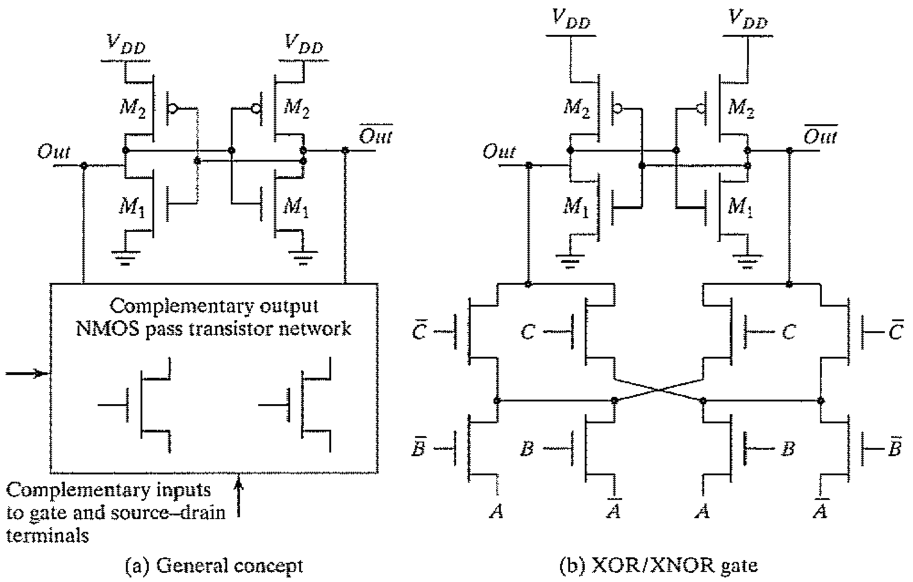


Figure 6-42 Swing-restored pass-transistor logic [Landman91, Parameswar96].

threshold devices for the NMOS pass-transistors eliminates most of the threshold drop, and passes a signal close to V_{DD}. All devices other than the pass-transistors (i.e., the inverters) are implemented using standard high-threshold devices. The use of multiple-threshold transistors is becoming more common, and involves simple modifications to existing process flows. Observe that even if the device implants were carefully calibrated to yield thresholds of exactly zero, the body effect of the device still would prevent a full swing to V_{DD}.

The use of zero-threshold transistors has some negative impact on the power consumption due to the subthreshold currents flowing through the pass-transistors, even if V_{GS} is below V_T. This is demonstrated in Figure 6-43, which points out a potential sneak dc-current path. While these leakage paths are not critical when the device is switching constantly, they do pose a significant energy overhead when the circuit is in the idle state.

Solution 3: Transmission-Gate Logic The most widely used solution to deal with the voltage-drop problem is the use of *transmission gates*.[4] This technique builds on the complementary properties of NMOS and PMOS transistors: NMOS devices pass a strong 0, but a weak 1, while PMOS transistors pass a strong 1 but a weak 0. The ideal approach is to use an NMOS to pull down and a PMOS to pull up. The transmission gate combines the best of both device flavors by placing an NMOS device in parallel with a PMOS device as in Figure 6-44a. The control

[4]The transmission gate is only one of the possible solutions. Other styles of pass-transistor networks that combine NMOS and PMOS transistors have been devised. Double pass-transistor logic (DPL) is an example of such [Bernstein98, pp. 84].

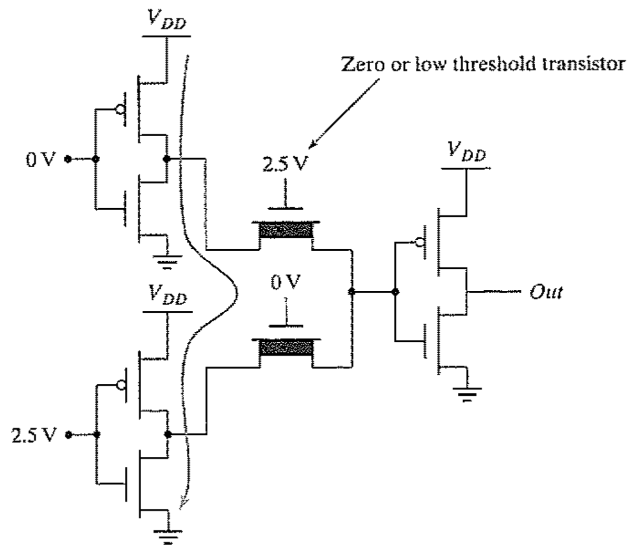


Figure 6-43 Static power consumption when using zero-threshold pass-transistors.

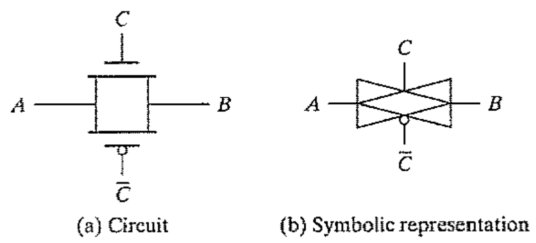


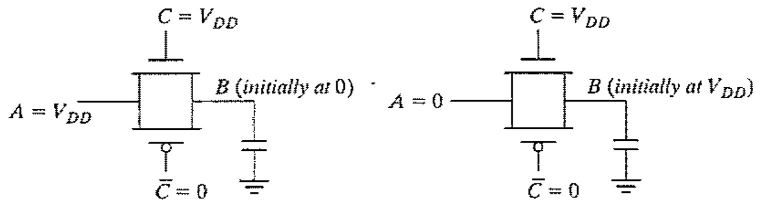
Figure 6-44 CMOS transmission gate.

signals to the transmission gate (C and $\overline{C}$) are complementary. The transmission gate acts as a bidirectional switch controlled by the gate signal C. When $C = 1$, both MOSFETs are on, allowing the signal to pass through the gate. In short,

$$A = B \quad \text{if} \quad C = 1 \tag{6.32}$$

On the other hand, $C = 0$ places both transistors in cutoff, creating an open circuit between nodes A and B. Figure 6-44b shows a commonly used transmission-gate symbol.

Consider the case of charging node B to V_{DD} for the transmission-gate circuit in Figure 6-45a. Node A is set at V_{DD}, and the transmission gate is enabled ($C = 1$ and $\overline{C} = 0$). If only the NMOS pass device were present, node B would only charge up to $V_{DD} - V_{Tn}$, at which point the NMOS device would turn off. However, since the PMOS device is present



(a) Charging node B (b) Discharging node B

Figure 6-45 Transmission gates enable rail-to-rail switching.

and is "on" ($V_{GSp} = -V_{DD}$), the output charges all the way up to V_{DD}. Figure 6-45b shows the opposite case—that is, discharging node B to 0. B is initially at V_{DD} when node A is driven low. The PMOS transistor by itself can only pull-down node B to V_{Tp} at which point it turns *off*. The parallel NMOS device stays turned on, however (since its $V_{GSn} = V_{DD}$), and pulls node B all the way to GND. Although the transmission gate requires two transistors and more control signals, it enables rail-to-rail swing.

Transmission gates can be used to build some complex gates very efficiently. Figure 6-46 shows an example of a simple inverting two-input multiplexer. This gate either selects input A or B on the basis of the value of the control signal S, which is equivalent to implementing the following Boolean function:

$$\bar{F} = (A \cdot S + B \cdot \bar{S}) \tag{6.33}$$

A complementary implementation of the gate requires eight transistors instead of six.

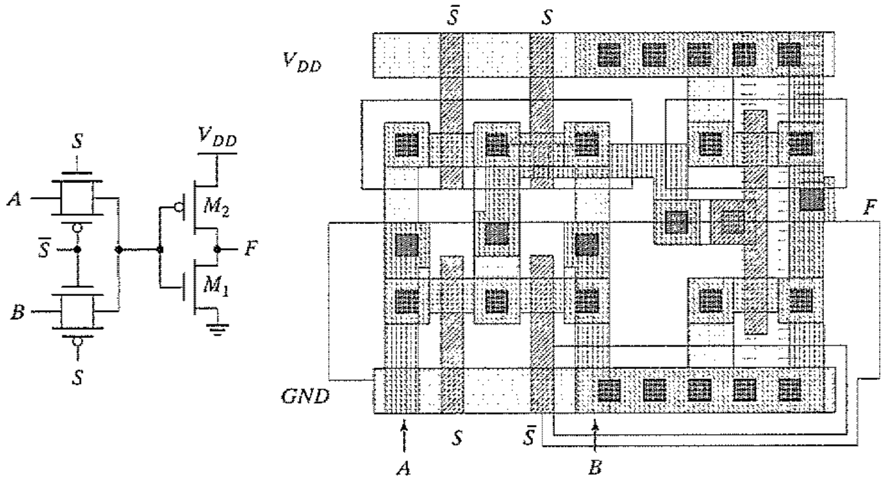


Figure 6-46 Transmission-gate multiplexer and its layout.

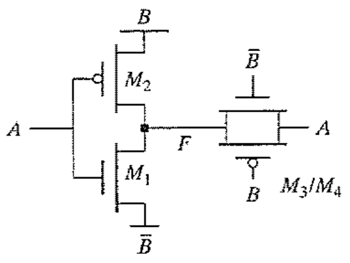


Figure 6-47 Transmission-gate XOR.

Another example of the effective use of transmission gates is the popular XOR circuit shown in Figure 6-47. The complete implementation of this gate requires only 6 transistors (including the inverter used for the generation of $\overline{B}$), compared with the 12 transistors required for a complementary implementation. To understand the operation of this circuit, we need only analyze the $B = 0$ and $B = 1$ cases separately. For $B = 1$, transistors M_1 and M_2 act as an inverter, while the transmission gate M_3/M_4 is off; hence, $F = \overline{A}B$. In the opposite case, M_1 and M_2 are disabled, and the transmission gate is operational, or $F = A\overline{B}$. The combination of both leads to the XOR function. Notice that regardless of the values of A and B, node F always has a connection to either V_{DD} or GND and thus is a low-impedance node. When designing static-pass-transistor networks, it is essential to adhere to the low-impedance rule under all circumstances. Other examples in which transmission-gate logic is effectively used are fast adder circuits and registers.

Performance of Pass-Transistor and Transmission-Gate Logic

The pass-transistor and the transmission gate are, unfortunately, not ideal switches, and they have a series resistance associated with them. To quantify the resistance, consider the circuit in Figure 6-48, which involves charging a node from 0 V to V_{DD}. In this discussion, we use the large-signal definition of resistance, which involves dividing the voltage across the switch by the drain current. The effective resistance of the switch is modeled as a parallel connection of the resistances R_n and R_p of the NMOS and PMOS devices, defined as $(V_{DD} - V_{out})/I_{Dn}$ and $(V_{DD} - V_{out})/(-I_{Dp})$, respectively. The currents through the devices obviously are dependent on the value of V_{out} and the operating mode of the transistors. During the low-to-high transition, the pass-transistors traverse through a number of operation modes. For low values of V_{out}, the NMOS device is saturated and the resistance is approximated as

$$R_p = \frac{V_{out} - V_{DD}}{I_{Dp}} = \frac{V_{out} - V_{DD}}{k_p \cdot \left((-V_{DD} - V_{Tp})(V_{out} - V_{DD}) - \frac{(V_{out} - V_{DD})^2}{2}\right)} \tag{6.34}$$
$$\approx \frac{1}{k_p(-V_{DD} - V_{Tp})}$$

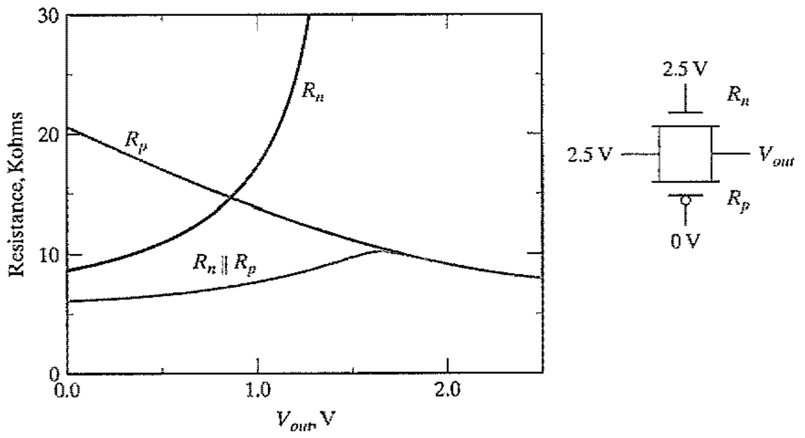


Figure 6-48 Simulated equivalent resistance of transmission gate for low-to-high transition (for $(W\text{–}L)_n = (W\text{–}L)_p = 0.5\ \mu m/0.25\ \mu m$). A similar response for overall resistance is obtained for the high-to-low transition.

The resistance goes up for increasing values of V_{out} and approaches infinity when V_{out} reaches $V_{DD} - V_{Tn}$ and the device shuts off. Similarly, we can analyze the behavior of the PMOS transistor. When V_{out} is small, the PMOS is saturated, but it enters the linear mode of operation for V_{out} approaching V_{DD}. This gives the following approximated resistance:

$$R_p = \frac{V_{out} - V_{DD}}{I_{Dp}} = \frac{V_{out} - V_{DD}}{k_p \cdot \left((-V_{DD} - V_{Tp})(V_{out} - V_{DD}) - \frac{(V_{out} - V_{DD})^2}{2}\right)} \tag{6.35}$$
$$\approx \frac{1}{k_p(-V_{DD} - V_{Tp})}$$

The simulated value of $R_{eq} = R_p \| R_n$ as a function of V_{out} is plotted in Figure 6-48. It can be observed that R_{eq} is relatively constant ($\approx 8\ k\Omega$ in this particular case). The same is true in other design instances (for example, when discharging C_L). When analyzing transmission-gate networks, the simplifying assumption that the switch has a constant resistive value is therefore acceptable.

Problem 6.7 Equivalent Resistance during Discharge

Determine the equivalent resistance by simulation for the high-to-low transition of a transmission gate. (In other words, produce a plot similar to the one presented in Figure 6-48).

An important consideration is the delay associated with a chain of transmission gates. Figure 6-49 shows a chain of n transmission gates. Such a configuration often occurs in circuits such as adders or deep multiplexors. Assume that all transmission gates are turned on and a step

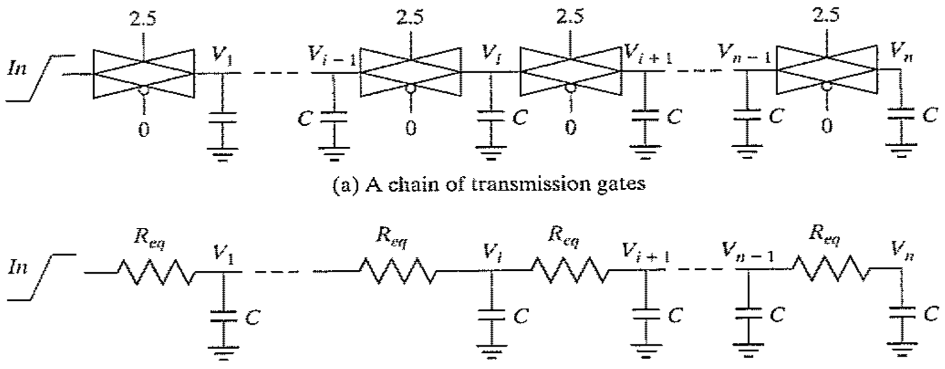


Figure 6-49 Speed optimization in transmission-gate networks.

is applied at the input. To analyze the propagation delay of this network, the transmission gates are replaced by their equivalent resistances R_{eq}. This produces the network of Figure 6-49b.

The delay of a network of n transmission gates in sequence can be estimated by using the Elmore approximation (see Chapter 4):

$$t_p(V_n) = 0.69\sum_{k=0}^{n} CR_{eq}k = 0.69CR_{eq}\frac{n(n+1)}{2} \tag{6.36}$$

This means that the propagation delay is proportional to n^2 and increases rapidly with the number of switches in the chain.

Example 6.13 Delay of Transmission-Gate Chain

Consider 16 cascaded minimum-sized transmission gates, each with an average resistance of 8 kΩ. The node capacitance consists of the capacitance of two NMOS and PMOS devices (junction and gate). Since the gate inputs are assumed to be fixed, there is no Miller multiplication. The capacitance can be calculated to be approximately 3.6 fF for the low-to-high transition. The delay is given by

$$t_p = 0.69 \cdot CR_{eq}\frac{n(n+1)}{2} = 0.69 \cdot (3.6\text{ fF})(8\text{ K}\Omega)\left(\frac{16(16+1)}{2}\right) \approx 2.7\text{ ns} \tag{6.37}$$

The transient response for this particular example is shown in Figure 6-50. The simulated delay is 2.7 ns. It is remarkable that a simple RC model predicts the delay so accurately. It is also clear that the use of long pass-transistor chains causes significant delay degradation.

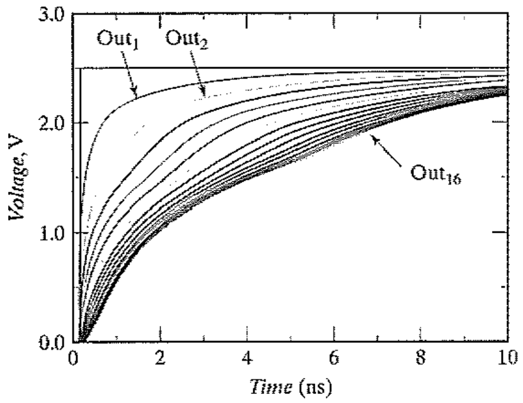


Figure 6-50 Speed optimization in transmission-gate networks.

The most common approach for dealing with the long delay is to break the chain and insert buffers every m switches (Figure 6-51). Assuming a propagation delay t_{buf} for each buffer, the overall propagation delay of the transmission gate–buffer network is then computed as follows:

$$\begin{aligned} t_p &= 0.69\left[\frac{n}{m}CR_{eq}\frac{m(m+1)}{2}\right] + \left(\frac{n}{m}-1\right)t_{buf} \\ &= 0.69\left[CR_{eq}\frac{n(m+1)}{2}\right] + \left(\frac{n}{m}-1\right)t_{buf} \end{aligned} \tag{6.38}$$

The resulting delay exhibits only a linear dependence on the number of switches n, in contrast to the unbuffered circuit, which is quadratic in n. The optimal number of switches m_{opt} between buffers can be found by setting the derivative $\frac{\partial t_p}{\partial m}$ to 0, which yields

$$m_{opt} = 1.7\sqrt{\frac{t_{pbuf}}{CR_{eq}}} \tag{6.39}$$

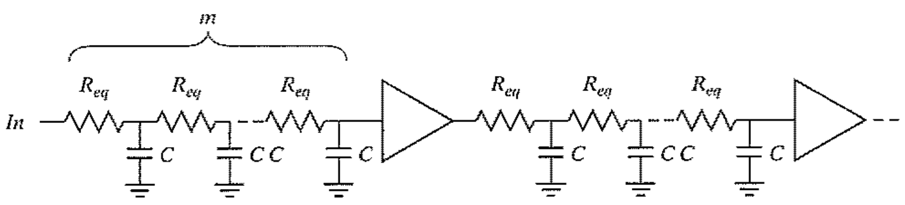


Figure 6-51 Breaking up long transmission-gate chains by inserting buffers.

Obviously, the number of switches per segment grows with increasing values of t_{buf}. In current technologies, m_{opt} typically equals 3 or 4. The presented analysis ignores that tp_{buf} itself is a function of the load m. A more accurate analysis taking this factor into account is presented in Chapter 9.

Example 6.14 Transmission-Gate Chain

Consider the same 16-transmission-gate chain. The buffers shown in Figure 6-51 can be implemented as inverters (instead of two cascaded inverters). In some cases, it might be necessary to add an extra inverter to produce the correct polarity. Assuming that each inverter is sized such that the NMOS is 0.5 μm/0.25 μm and PMOS is 0.5 μm /0.25 μm, Eq. (6.39) predicts that an inverter must be inserted every 3 transmission gates. The simulated delay when placing an inverter every two transmission gates is 154 ps; for every three transmission gates, the delay is 154 ps; and for four transmission gates, it is 164 ps. The insertion of buffering inverters reduces the delay by a factor of almost 2.

CAUTION: Although many of the circuit styles discussed in the previous sections sound very interesting, and might be superior to static CMOS in many respects, none has the *robustness and ease of design* of complementary CMOS. Therefore, use them sparingly and with caution. For designs that have no extreme area, complexity, or speed constraints, complementary CMOS is the recommended design style.

6.3 Dynamic CMOS Design

It was noted earlier that static CMOS logic with a fan-in of N requires $2N$ devices. A variety of approaches were presented to reduce the number of transistors required to implement a given logic function including pseudo-NMOS, pass-transistor logic, etc. The pseudo-NMOS logic style requires only $N + 1$ transistors to implement an N input logic gate, but unfortunately it has static power dissipation. In this section, an alternate logic style called *dynamic logic* is presented that obtains a similar result, while avoiding static power consumption. With the addition of a clock input, it uses a sequence of *precharge* and conditional *evaluation* phases.

6.3.1 Dynamic Logic: Basic Principles

The basic construction of an (n-type) dynamic logic gate is shown in Figure 6-52a. The PDN (pull-down network) is constructed exactly as in complementary CMOS. The operation of this circuit is divided into two major phases—*precharge* and *evaluation*—with the mode of operation determined by the *clock signal CLK*.

Precharge

When $CLK = 0$, the output node Out is precharged to V_{DD} by the PMOS transistor M_p. During that time, the evaluate NMOS transistor M_e is off, so that the pull-down path is disabled. The

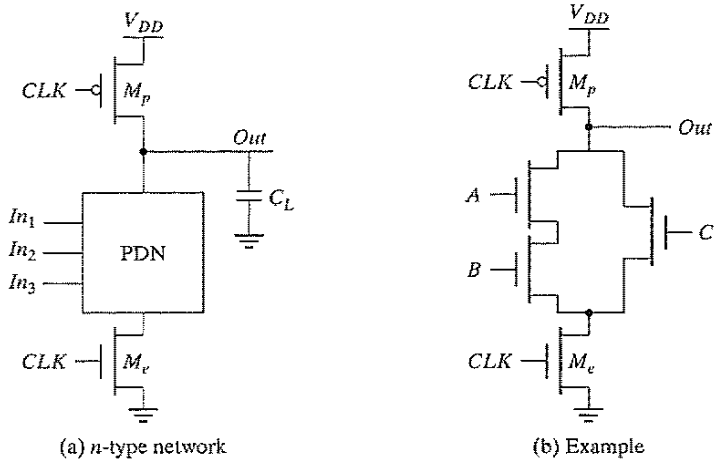


Figure 6-52 Basic concepts of a dynamic gate.

evaluation FET eliminates any static power that would be consumed during the precharge period (i.e., static current would flow between the supplies if both the pull-down and the precharge device were turned on simultaneously).

Evaluation

For $CLK = 1$, the precharge transistor M_p is off, and the evaluation transistor M_e is turned on. The output is conditionally discharged based on the input values and the pull-down topology. If the inputs are such that the PDN conducts, then a low resistance path exists between *Out* and GND, and the output is discharged to GND. If the PDN is turned off, the precharged value remains stored on the output capacitance C_L, which is a combination of junction capacitances, the wiring capacitance, and the input capacitance of the fan-out gates. During the evaluation phase, the only possible path between the output node and a supply rail is to GND. Consequently, once *Out* is discharged, it cannot be charged again until the next precharge operation. **The inputs to the gate can thus make at most one transition during evaluation.** Notice that the output can be in the *high-impedance state* during the evaluation period if the pull-down network is turned off. This behavior is fundamentally different from the static counterpart that always has a low resistance path between the output and one of the power rails.

As an example, consider the circuit shown in Figure 6-52b. During the precharge phase ($CLK = 0$), the output is precharged to V_{DD} regardless of the input values, because the evaluation device is turned off. During evaluation ($CLK = 1$), a conducting path is created between *Out* and GND if (and only if) $A \cdot B + C$ is TRUE. Otherwise, the output remains at the precharged state of V_{DD}. The following function is thus realized:

$$Out = \overline{CLK} + \overline{(A \cdot B + C)} \cdot CLK \tag{6.40}$$

A number of important properties can be derived for the dynamic logic gate:

- The logic function is implemented by the NMOS pull-down network. The construction of the PDN proceeds just as it does for static CMOS.
- The *number of transistors* (for complex gates) is substantially lower than in the static case: $N + 2$ versus $2N$.
- It is *nonratioed*. The sizing of the PMOS precharge device is not important for realizing proper functionality of the gate. The size of the precharge device can be made large to improve the low-to-high transition time (of course, at a cost to the high-to-low transition time). There is, however, a trade-off with power dissipation, since a larger precharge device directly increases clock-power dissipation.
- It only consumes *dynamic power*. Ideally, no static current path ever exists between V_{DD} and GND. The overall power dissipation, however, can be significantly higher compared with a static logic gate.
- The logic gates have *faster switching speeds*, for two main reasons. The first (obvious) reason is due to the reduced load capacitance attributed to the lower number of transistors per gate and the single-transistor load per *fan-in*. This translates in a *reduced logical effort*. For instance, the logical effort of a two-input dynamic NOR gate equals 2/3, which is substantially smaller than the 5/3 of its static CMOS counterpart. The second reason is that the dynamic gate does not have short circuit current, and all the current provided by the pull-down devices goes towards discharging the load capacitance.

The low and high output levels of V_{OL} and V_{OH} are easily identified as GND and V_{DD}, and they are not dependent on the transistor sizes. The other VTC parameters are dramatically different from static gates. Noise margins and switching thresholds have been defined as static quantities that are not a function of time. To be functional, a dynamic gate requires a periodic sequence of precharges and evaluations. Pure static analysis, therefore, does not apply. During the evaluation period, the pull-down network of a dynamic inverter starts to conduct when the input signal exceeds the threshold voltage (V_{Tn}) of the NMOS pull-down transistor. Therefore, it is reasonable to assume that the switching threshold (V_M) as well as V_{IH} and V_{IL} are equal to V_{Tn}. This translates to a low value for the NM_L.

Design Consideration

It is also possible to implement dynamic logic using the dual approach, where the output node is connected by a predischarge NMOS transistor to GND, and the evaluation PUN network is implemented in PMOS. The operation is similar: During precharge, the output node is discharged to GND; during evaluation, the output is conditionally charged to V_{DD}. This p-type dynamic gate has the disadvantage of being slower than the n-type because of the lower current drive of the PMOS transistors. ■

6.3.2 Speed and Power Dissipation of Dynamic Logic

The main advantages of dynamic logic are increased speed and reduced implementation area. Fewer devices to implement a given logic function implies that the overall load capacitance is much smaller. The analysis of the switching behavior of the gate has some interesting peculiarities to it. After the precharge phase, the output is high. For a low input signal, no additional switching occurs. As a result, $t_{pLH} = 0$! The high-to-low transition, on the other hand, requires the discharging of the output capacitance through the pull-down network. Therefore, t_{pHL} is proportional to C_L and the current-sinking capabilities of the pull-down network. The presence of the evaluation transistor slows the gate somewhat, as it presents an extra series resistance. Omitting this transistor, while functionally not forbidden, may result in static power dissipation and potentially a performance loss.

The preceding analysis is somewhat unfair because it ignores the influence of the precharge time on the switching speed of the gate. The precharge time is determined by the time it takes to charge C_L through the PMOS precharge transistor. During this time, the logic in the gate cannot be utilized. Very often, however, the overall digital system can be designed in such a way that the precharge time coincides with other system functions. For instance, the precharge of the arithmetic unit in a microprocessor could coincide with the instruction decode. The designer has to be aware of this "dead zone" in the use of dynamic logic and thus should carefully consider the pros and cons of its usage, taking the overall system requirements into account.

Example 6.15 A Four-Input Dynamic NAND Gate

Figure 6-53 shows the design of a four-input NAND example designed using the dynamic-circuit style. Due to the dynamic nature of the gate, the derivation of the voltage-transfer

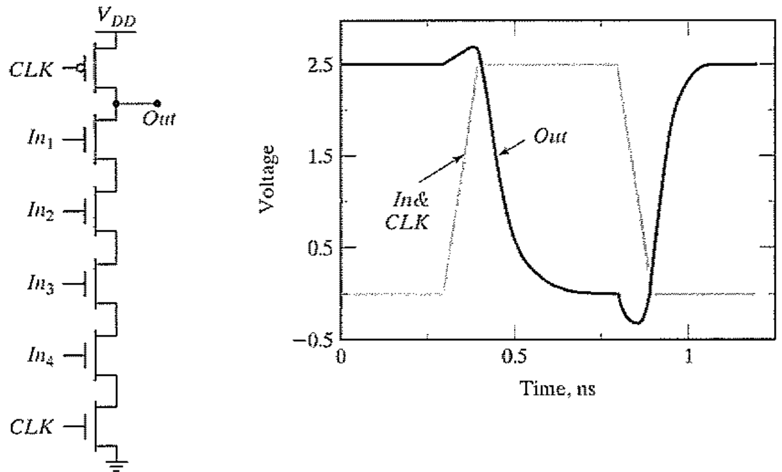


Figure 6-53 Schematic and transient response of a four-input dynamic NAND gate.

characteristic diverges from the traditional approach. As discussed earlier, we assume that the switching threshold of the gate equals the threshold of the NMOS pull-down transistor. This results in asymmetrical noise margins, as shown in Table 6-10.

Table 6-10 The dc and ac parameters of a four-input dynamic NAND.

Transistors	V_{OH}	V_{OL}	V_M	NM_H	NM_L	t_{pHL}	t_{pLH}	t_{pre}
6	2.5 V	0 V	V_{TN}	$2.5 - V_{TN}$	V_{TN}	110 ps	0 ps	83 ps

The dynamic behavior of the gate is simulated with SPICE. It is assumed that all inputs are set high when the clock goes high. On the rising edge of the clock, the output node is discharged. The resulting transient response is plotted in Figure 6-53, and the propagation delays are summarized in Table 6-10. The duration of the precharge cycle can be adjusted by changing the size of the PMOS precharge transistor. Making the PMOS too large should be avoided, however, as it both slows down the gate and increases the capacitive load on the clock line. For large designs, the latter factor might become a major design concern because the clock load can become excessive and hard to drive.

As mentioned earlier, the static gate parameters are time dependent. To illustrate this, consider a four-input NAND gate with all the partial inputs tied together, and are making a low-to-high transition. Figure 6-54 shows a transient simulation of the output voltage for three different input transitions—from 0 to 0.45 V, 0.5 V and 0.55 V, respectively. In the preceding discussion, we have defined the switching threshold of the dynamic gate as the device threshold. However, notice that the amount by which the output voltage drops is a strong function of the input voltage and the *available evaluation time*. The noise voltage needed to corrupt the signal has to be larger if the evaluation time is short. In other words, the switching threshold is truly time dependent.

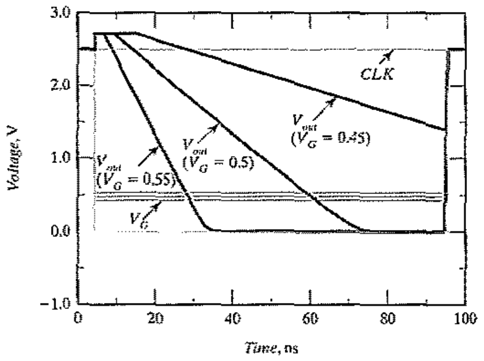


Figure 6-54 Effect of an input glitch on the output.
The switching threshold depends on the time for evaluation.
A larger glitch is acceptable if the evaluation phase is shorter.

It would appear that dynamic logic presents a significant advantage from a power perspective. There are three reasons for this. First, the physical capacitance is lower since dynamic logic uses fewer transistors to implement a given function. Also, the load seen for each fan-out is one transistor instead of two. Second, dynamic logic gates *by construction* can have at most one transition per clock cycle. Glitching (or dynamic hazards) does not occur in dynamic logic. Finally, dynamic gates do not exhibit short-circuit power since the pull-up path is not turned on when the gate is evaluating.

While these arguments generally are true, they are offset by other considerations: (1) the clock power of dynamic logic can be significant, particularly since the clock node has a guaranteed transition on every single clock cycle; (2) the number of transistors is greater than the minimal set required for implementing the logic; (3) short-circuit power may exist when leakage-combatting devices are added (as will be discussed further); and (4), most importantly, dynamic logic generally displays a higher switching activity due to the periodic *precharge* and *discharge* operations. Earlier, the transition probability for a static gate was shown to be $p_0 p_1 = p_0 (1 - p_0)$. For dynamic logic, the output transition probability does not depend on the state (history) of the inputs, but rather on the signal probabilities. For an n-tree dynamic gate, the output makes a $0 \rightarrow 1$ transition during the precharge phase only if the output was discharged during the preceding evaluate phase. Hence, the $0 \rightarrow 1$ transition probability for an n-type dynamic gate is given by

$$\alpha_{0 \rightarrow 1} = p_0 \tag{6.41}$$

where p_0 is the probability that the output is zero. This number is always greater than or equal to $p_0 p_1$. For uniformly distributed inputs, the transition probability for an N-input gate is

$$\alpha_{0 \rightarrow 1} = \frac{N_0}{2^N} \tag{6.42}$$

where N_0 is the number of zero entries in the truth table of the logic function.

Example 6.16 Activity Estimation in Dynamic Logic

To illustrate the increased activity for a dynamic gate, consider again a two-input NOR gate. An n-tree dynamic implementation is shown in Figure 6-55, along with its static counterpart. For equally probable inputs, there is a 75% probability that the output node of the dynamic gate discharges immediately after the precharge phase, implying that the activity for such a gate equals 0.75 (i.e., $P_{NOR} = 0.75\ C_L V_{dd}^2 f_{clk}$). The corresponding activity is a lot smaller, 3/16, for a static implementation. For a dynamic NAND gate, the transition probability is 1/4 (since there is a 25% probability the output will be discharged) while it is 3/16 for a static implementation. Although these examples illustrate that the switching activity of dynamic logic is generally higher, it should be noted that dynamic logic has lower physical capacitance. Both factors must be accounted for when analyzing dynamic power dissipation.

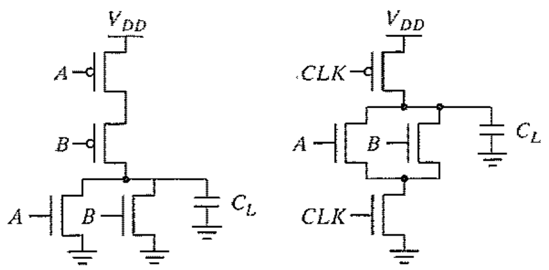


Figure 6-55 Static NOR versus *n*-type dynamic NOR.

Problem 6.8 Activity Computation

For the four-input dynamic NAND gate, compute the activity factor with the following assumption for the inputs: They are independent, and $p_{A=1} = 0.2, p_{B=1} = 0.3, p_{C=1} = 0.5$, and $p_{D=1} = 0.4$.

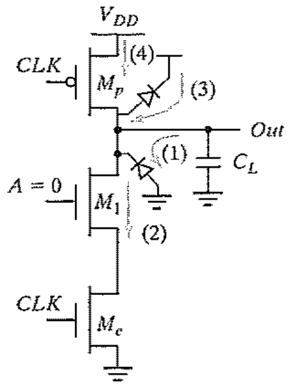
6.3.3 Signal Integrity Issues in Dynamic Design

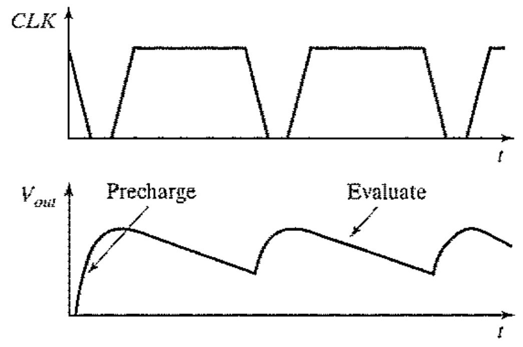
Dynamic logic clearly can result in high-performance solutions compared to static circuits. However, there are several important considerations that must be taken into account if one wants dynamic circuits to function properly. These include charge leakage, charge sharing, capacitive coupling, and clock feedthrough. These issues are discussed in some detail in this section.

Charge Leakage

The operation of a dynamic gate relies on the dynamic storage of the output value on a capacitor. If the pull-down network is *off*, ideally, the output should remain at the precharged state of V_{DD} during the evaluation phase. However, this charge gradually leaks away due to leakage currents, eventually resulting in a malfunctioning of the gate. Figure 6-56a shows the sources of leakage for the basic dynamic inverter circuit.

Source 1 and 2 are the *reverse-biased diode* and *subthreshold leakage* of the NMOS pull-down device M_1, respectively. The charge stored on C_L will slowly leak away through these leakage channels, causing a degradation in the high level (Figure 6-56b). Dynamic circuits therefore require a minimal clock rate, which is typically on the order of a few kHz. This makes the usage of dynamic techniques unattractive for low-performance products such as watches, or processors that use conditional clocks (where there are no guarantees on minimum clock rates). Note that the PMOS precharge device also contributes some leakage current due to the reverse bias diode (source 3) and the subthreshold conduction (source 4). To some extent, the leakage current of the PMOS counteracts the leakage of the pull-down path. As a result, the output voltage is going to be set by the resistive divider composed of the pull-down and pull-up paths.





(a) Leakage sources (b) Effect on waveforms

Figure 6-56 Leakag e issues in dynamic circuits.

Example 6.17 Leakage in Dynamic Circuits

Consider the simple inverter with all devices set at 0.5 μm/0.25 μm. Assume that the input is low during the evaluation period. Ideally, the output should remain at the precharged state of V_{DD}. However, as seen from Figure 6-57, the output voltage drops. Once the output drops below the switching threshold of the fan-out logic gate, the output is interpreted as a low voltage. Notice that the output settles to an intermediate voltage, due to the leakage current provided by the PMOS pull-up.

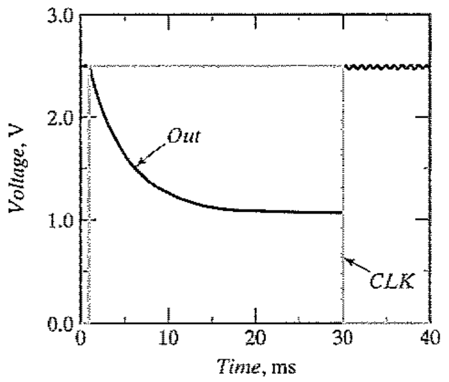


Figure 6-57 Impact of charge leakage. The output settles to an intermediate voltage determined by a resistive divider of the pull-down and pull-up devices.

Leakage is caused by the high-impedance state of the output node during the evaluate mode, when the pull-down path is turned off. The leakage problem may be counteracted by reducing the output impedance on the output node during evaluation. This often is done by

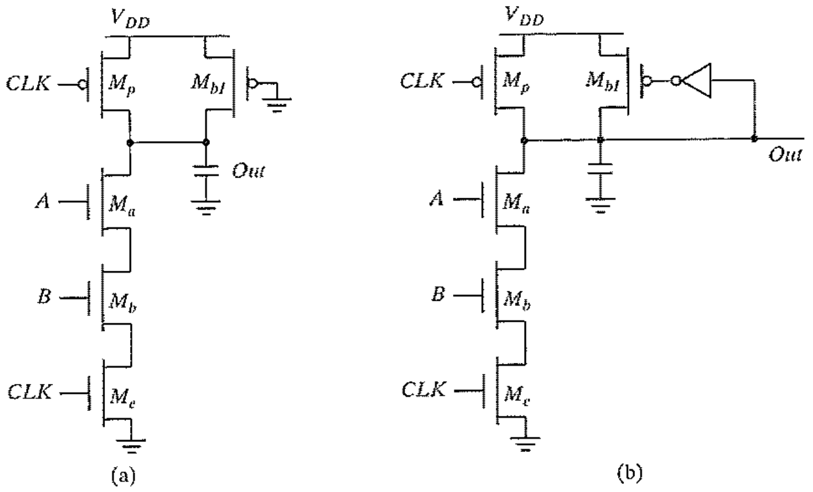


Figure 6-58 Static bleeders compensate for the charge leakage.

adding a *bleeder transistor*, as shown in Figure 6-58a. The only function of the bleeder—an NMOS style pull-up device—is to compensate for the charge lost due to the pull-down leakage paths. To avoid the ratio problems associated with this style of circuit and the associated static power consumption, the bleeder resistance is made high (in other words, the device is kept small). This allows the (strong) pull-down devices to lower the *Out* node substantially below the switching threshold of the next gate. Often, the bleeder is implemented in a feedback configuration to eliminate the static power dissipation altogether (Figure 6-58b).

Charge Sharing

Another important concern in dynamic logic is the impact of charge sharing. Consider the circuit in Figure 6-59. During the precharge phase, the output node is precharged to V_{DD}. Assume that all inputs are set to 0 during precharge, and that the capacitance C_a is discharged. Assume further that input B remains at 0 during evaluation, while input A makes a $0 \rightarrow 1$ transition, turning transistor M_a on. The charge stored originally on capacitor C_L is redistributed over C_L and C_a. This causes a drop in the output voltage, which cannot be recovered due to the dynamic nature of the circuit.

The influence on the output voltage is readily calculated. Under the assumptions given previously, the following initial conditions are valid: $V_{out}(t = 0) = V_{DD}$ and $V_X(t = 0) = 0$. As a result, two possible scenarios must be considered:

1. $\Delta V_{out} < V_{Tn}$. In this case, the final value of V_X equals $V_{DD} - V_{Tn}(V_X)$. Charge conservation then yields

$$C_L V_{DD} = C_L V_{out}(\text{final}) + C_a[V_{DD} - V_{Tn}(V_X)]$$

or

$$\Delta V_{out} = V_{out}(\text{final}) + (-V_{DD}) = -\frac{C_a}{C_L}[V_{DD} - V_{Tn}(V_X)] \tag{6.43}$$

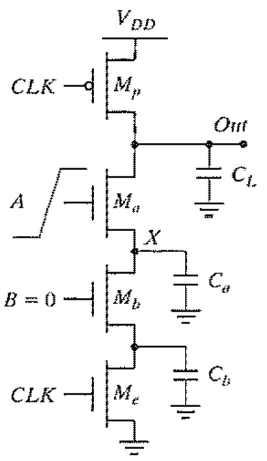


Figure 6-59 Charge sharing in dynamic networks.

2. $\Delta V_{out} > V_{Tn}$. V_{out} and V_X then reach the same value:

$$\Delta V_{out} = -V_{DD}\left(\frac{C_a}{C_a + C_L}\right) \tag{6.44}$$

We determine which of these scenarios is valid by the capacitance ratio. The boundary condition between the two cases can be determined by setting ΔV_{out} equal to V_{Tn} in Eq. (6.44), yielding

$$\frac{C_a}{C_L} = \frac{V_{Tn}}{V_{DD} - V_{Tn}} \tag{6.45}$$

Case 1 holds when the (C_a/C_L) ratio is smaller than the condition defined in Eq. (6.45). If not, Eq. (6.44) is valid. Overall, it is desirable to keep the value of ΔV_{out} below $|V_{Tp}|$. The output of the dynamic gate might be connected to a static inverter, in which case the low level of V_{out} would cause static power consumption. One major concern is a circuit malfunction if the output voltage is brought below the switching threshold of the gate it drives.

Example 6.18 Charge Sharing

Let us consider the impact of charge sharing on the dynamic logic gate shown in Figure 6-60, which implements a three-input EXOR function $y = A \oplus B \oplus C$. The first question to be resolved is what conditions cause the worst case voltage drop on node y. For simplicity, ignore the load inverter, and assume that all inputs are low during the precharge operation and that all isolated internal nodes (V_a, V_b, V_c, and V_d) are initially at 0 V.

Inspection of the truth table for this particular logic function shows that the output stays high for 4 out of 8 cases. The worst case change in output is obtained by exposing the maximum amount of internal capacitance to the output node during the evaluation

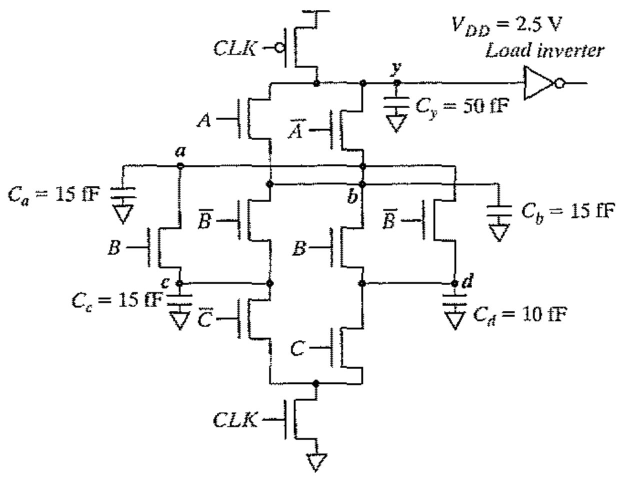


Figure 6-60 Example illustrating the charge-sharing effect in dynamic logic.

period. This happens for $\overline{A}\ B\ C$ or $A\ \overline{B}\ C$. The voltage change can then be obtained by equating the initial charge with the final charge as done with equation Eq. (6.44), yielding a worst case change of 30/(30 + 50) * 2.5 V = 0.94 V. To ensure that the circuit functions correctly, the switching threshold of the connecting inverter should be placed below 2.5 − 0.94 = 1.56 V.

The most common and effective approach to deal with the charge redistribution is to also precharge critical internal nodes, as shown in Figure 6-61. Since the internal nodes are charged

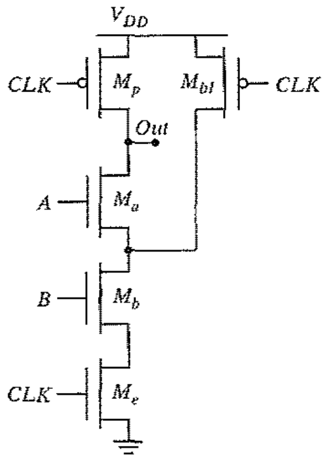


Figure 6-61 Dealing with charge sharing by precharging internal nodes. An NMOS precharge transistor may also be used, but this requires an inverted clock.

to V_{DD} during precharge, charge sharing does not occur. This solution obviously comes at the cost of increased area and capacitance.

Capacitive Coupling

The relatively high impedance of the output node makes the circuit very sensitive to crosstalk effects. A wire routed over or next to a dynamic node may couple capacitively and destroy the state of the floating node. Another equally important form of capacitive coupling is *backgate (or output-to-input) coupling.* Consider the circuit shown in Figure 6-62a, in which a dynamic two-input NAND gate drives a static NAND gate. A transition in the input *In* of the static gate may cause the output of the gate (Out_2) to go low. This output transition couples capacitively to the other input of the gate (the dynamic node Out_1) through the gate–source and gate–drain capacitances of transistor M_4. A simulation of this effect is shown in Figure 6-62b. It demonstrates how the coupling causes the output of the dynamic gate Out_1 to drop significantly. This further causes the output of the static NAND gate not to drop all the way down to 0 V and a small amount of static power to be dissipated. If the voltage drop is large enough, the circuit can evaluate incorrectly, and the NAND output may not go low. When designing and laying out dynamic circuits, special care is needed to minimize capacitive coupling.

Clock Feedthrough

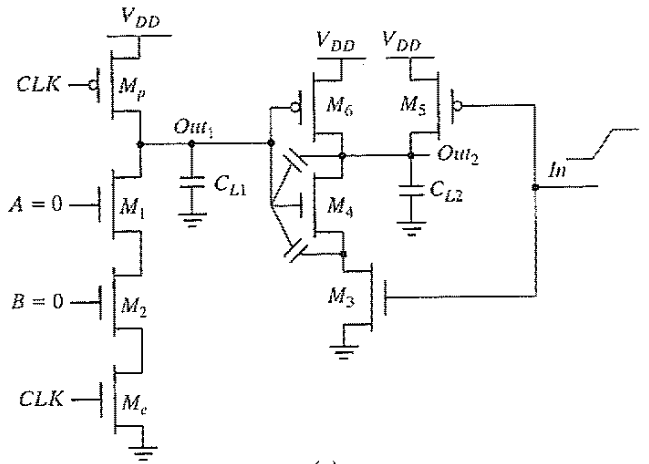
A special case of capacitive coupling is clock feedthrough, an effect caused by the capacitive coupling between the clock input of the precharge device and the dynamic output node. The coupling capacitance consists of the gate-to-drain capacitance of the precharge device, and includes both the overlap and channel capacitances. This capacitive coupling causes the output of the dynamic node to rise above V_{DD} on the low-to-high transition of the clock, assuming that the pull-down network is turned off. Subsequently, the fast rising and falling edges of the clock couple onto the signal node, as is quite apparent in the simulation of Figure 6-62b.

The danger of clock feedthrough is that it may cause the normally reverse-biased junction diodes of the precharge transistor to become forward biased. This causes electron injection into the substrate, which can be collected by a nearby high-impedance node in the **1** state, eventually resulting in faulty operation. CMOS latchup might be another result of this injection. For all purposes, high-speed dynamic circuits should be carefully simulated to ensure that clock feedthrough effects stay within bounds.

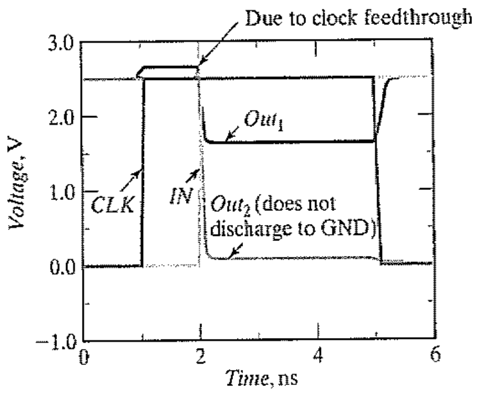
All of the preceding considerations demonstrate that the design of dynamic circuits is rather tricky and requires extreme care. It should therefore be attempted only when high performance is required, or high quality design-automation tools are available.

6.3.4 Cascading Dynamic Gates

Besides the signal integrity issues, there is one major catch that complicates the design of dynamic circuits: Straightforward cascading of dynamic gates to create multilevel logic structures does not work. The problem is best illustrated with two cascaded *n*-type dynamic



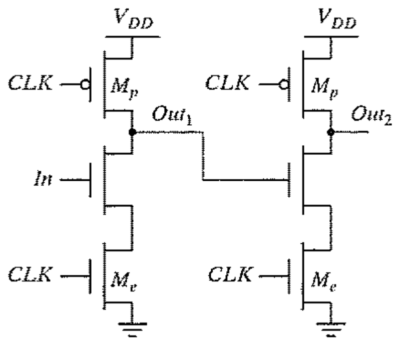
(a)



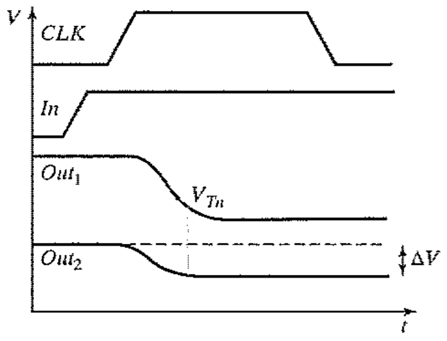
(b)

Figure 6-62 Example demonstrating the effect of backgate coupling: (a) circuit schematics; (b) simulation results.

inverters, shown in Figure 6-63a. During the precharge phase (i.e., *CLK* = 0), the outputs of both inverters are precharged to V_{DD}. Assume that the primary input *In* makes a 0 → 1 transition (Figure 6-63b). On the rising edge of the clock, output Out_1 starts to discharge. The second output should remain in the precharged state of V_{DD} as its expected value is 1 (Out_1 transitions to 0 during evaluation). However, there is a finite propagation delay for the input to discharge Out_1 to GND. Therefore, the second output also starts to discharge. As long as Out_1 exceeds the switching threshold of the second gate, which approximately equals V_{Tn}, a conducting path exists between Out_2 and GND, and precious charge is lost at Out_2. The conducting path is only disabled once Out_1 reaches V_{Tn}, and turns off the NMOS pull-down transistor. This leaves Out_2 at an intermediate voltage level. The correct level will not be recovered, because dynamic gates rely on capacitive storage, in contrast to static gates, which have dc restoration. The charge loss leads to reduced noise margins and potential malfunctioning.



(a)



(b)

Figure 6-63 Cascade of dynamic *n*-type blocks.

The cascading problem arises because the outputs of each gate—and thus the inputs to the next stages—are precharged to 1. This may cause inadvertent discharge in the beginning of the evaluation cycle. Setting all the inputs to 0 during precharge addresses that concern. When doing so, all transistors in the pull-down network are turned off after precharge, and no inadvertent discharging of the storage capacitors can occur during evaluation. In other words, correct operation is guaranteed as long as **the inputs can only make a single 0 → 1 transition during the evaluation period.**[5] Transistors are turned on only when needed—and at most, once per cycle. A number of design styles complying with this rule have been conceived, but the two most important ones are discussed next.

Domino Logic

Concept A domino logic module [Krambeck82] consists of an *n*-type dynamic logic block followed by a static inverter (Figure 6-64). During precharge, the output of the *n*-type dynamic gate is charged up to V_{DD}, and the output of the inverter is set to 0. During evaluation, the dynamic gate conditionally discharges, and the output of the inverter makes a conditional transition from 0 → 1. If one assumes that all the inputs of a domino gate are outputs of other domino gates,[6] then it is ensured that all inputs are set to 0 at the end of the precharge phase, and that the only transitions during evaluation are 0 → 1 transitions. Hence, the formulated rule is obeyed. The introduction of the static inverter has the additional advantage that the fan-out of the gate is driven by a static inverter with a low-impedance output, which increases noise immunity. Also, the buffer reduces the capacitance of the dynamic output node by separating internal and load capacitances. Finally, the inverter can be used to drive a bleeder device to combat leakage and charge redistribution, as shown in the second stage of Figure 6-64.

[5]This ignores the impact of charge distribution and leakage effects, discussed earlier.
[6]It is required that all other inputs that do not fall under this classification (for instance, primary inputs) stay constant during evaluation.

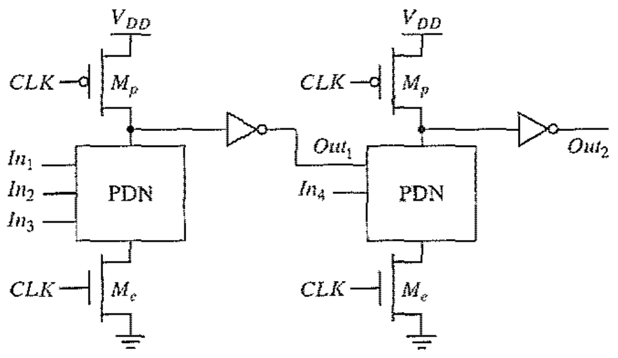


Figure 6-64 Domino CMOS logic.

Consider now the operation of a chain of domino gates. During precharge, all inputs are set to 0. During evaluation, the output of the first domino block either stays at 0 or makes a $0 \rightarrow 1$ transition, affecting the second gate. This effect might ripple through the whole chain, one after the other, similar to a line of falling dominoes—hence the name. Domino CMOS has the following properties:

- Since each dynamic gate has a static inverter, only noninverting logic can be implemented. Although there are ways to deal with this, as discussed in a subsequent section, this is a major limiting factor, and pure domino design has thus become rare.
- Very high speeds can be achieved: only a rising edge delay exists, while t_{pHL} equals zero. The inverter can be sized to match the *fan-out*, which is already much smaller than in the complimentary static CMOS case, as only a single gate capacitance has to be accounted for per fan-out gate.

Since the inputs to a domino gate are low during precharge, it is tempting to eliminate the evaluation transistor because this reduces clock load and increases pull-down drive. However, eliminating the evaluation device extends the precharge cycle—the precharge now has to ripple through the logic network as well. Consider the logic network shown in Figure 6-65, where the evaluation devices have been eliminated. If the primary input In_1 is 1 during evaluation, the output of each dynamic gate evaluates to 0, and the output of each static inverter is 1. On the falling edge of the clock, the precharge operation is started. Assume further that In_1 makes a high-to-low transition. The input to the second gate is initially high, and it takes two gate delays before In_2 is driven low. During that time, the second gate cannot precharge its output, as the pull-down network is fighting the precharge device. Similarly, the third gate has to wait until the second gate precharges before it can start precharging, etc. Therefore, the time taken to precharge the logic circuit is equal to its critical path. Another important negative is the extra power dissipation when both pull-up and pull-down devices are on. Therefore, it is good practice to always utilize evaluation devices.

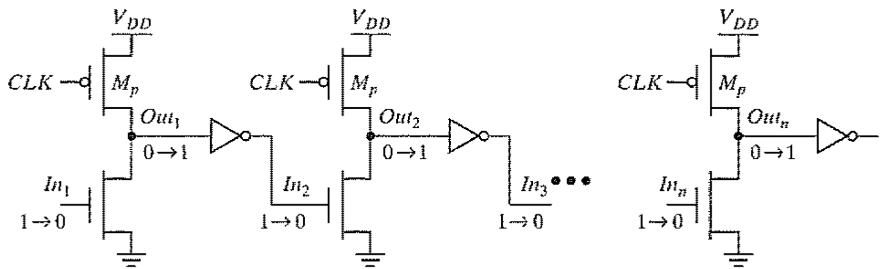


Figure 6-65 Effect of ripple precharge when the evaluation transistor is removed. The circuit also exhibits static power dissipation.

Dealing with the Noninverting Property of Domino Logic A major limitation in domino logic is that only noninverting logic can be implemented. This requirement has limited the widespread use of pure domino logic. There are several ways to deal with it, though. Figure 6-66 shows one approach to the problem—reorganizing the logic using simple boolean transforms such as De Morgan's Law. Unfortunately, this sort of optimization is not always possible, and more general schemes may have to be used.

A general (but expensive) approach to solving the problem is the use of differential logic. *Dual-rail domino* is similar in concept to the DCVSL structure discussed earlier, but it uses a precharged load instead of a static cross-coupled PMOS load. Figure 6-67 shows the circuit schematic of an AND/NAND differential logic gate. Note that all inputs come from other differential domino gates. They are low during the precharge phase, while making a conditional $0 \to 1$ transition during evaluation. Using differential domino, it is possible to implement any arbitrary function. This comes at the expense of an increased power dissipation, since a transition is guaranteed every single clock cycle regardless of the input values—either O or $\overline{O}$ must make a $0 \to 1$ transition. The function of transistors M_{f1} and M_{f2} is to keep the circuit static when the clock is high for extended periods of time (*bleeder*). Notice that this circuit is not ratioed, even in the presence of the PMOS pull-up devices! Due to its high performance, this differential approach is very popular, and is used in several commercial microprocessors.

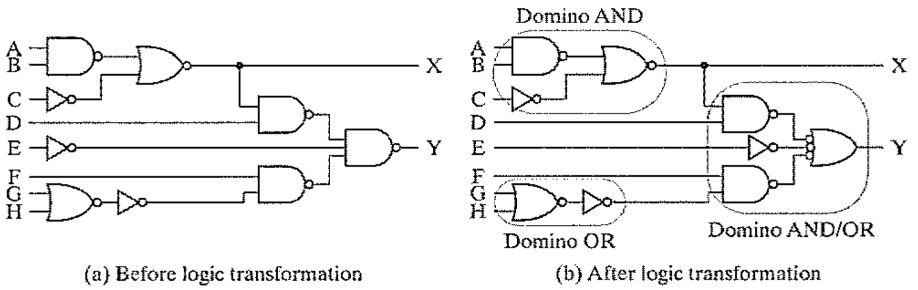

A
B
C
D
E
F
G
H
X
Y
Domino AND
Domino OR
Domino AND/OR
(a) Before logic transformation
(b) After logic transformation

Figure 6-66 Restructuring logic to enable implementation by using noninverting domino logic.

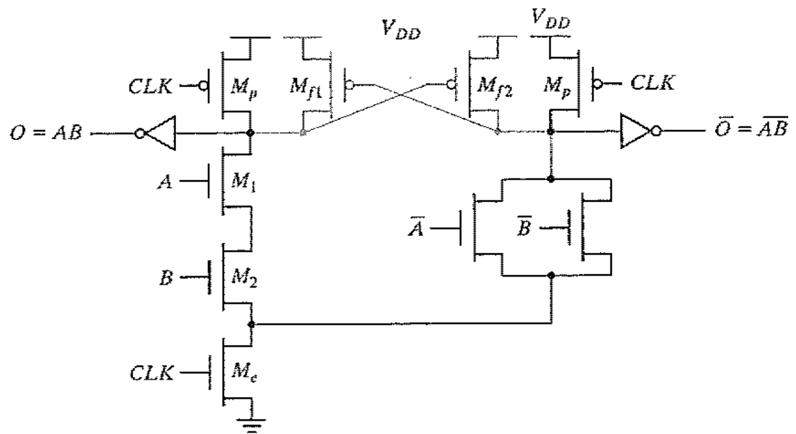


Figure 6-67 Simple dual rail (differential) domino logic gate.

Optimization of Domino Logic Gates Several optimizations can be performed on domino logic gates. The most obvious performance optimization involves the sizing of the transistors in the static inverter. With the inclusion of the evaluation devices in domino circuits, all gates precharge in parallel, and the precharge operation takes only two gate delays—charging the output of the dynamic gate to V_{DD}, and driving the inverter output low. The critical path during evaluation goes through the pull-down path of the dynamic gate and through the PMOS pull-up transistor of the static inverter. Therefore, to speed up the circuit during evaluation, the beta ratio of the static inverter should be made high so that its switching threshold is close to V_{DD}. This can be accomplished by using a small (minimum-sized) NMOS and a large PMOS device. The minimum-sized NMOS only affects the precharge time, which is generally limited due to the parallel precharging of all gates. The only disadvantage of using a large beta ratio is a reduction in noise margin. Hence, a designer should consider reduced noise margin and performance impact simultaneously during the device sizing.

Numerous variations of domino logic have been proposed [Bernstein98]. One optimization that reduces area is *multiple-output domino logic*. The basic concept is illustrated in Figure 6-68. It exploits the fact that certain outputs are subsets of other outputs to generate a number of logical functions in a single gate. In this example, $O3 = C + D$ is used in all three outputs, and thus it is implemented at the bottom of the pull-down network. Since $O2$ equals $B \cdot O3$, it can reuse the logic for $O3$. Notice that the internal nodes have to be precharged to V_{DD} to produce the correct results. Given that the internal nodes precharge to V_{DD}, the number of devices driving precharge devices is not reduced. However, the number of evaluation transistors is drastically reduced because they are amortized over multiple outputs. Additionally, this approach results in a reduction of the fan-out factor, again due to the reuse of transistors over multiple functions.

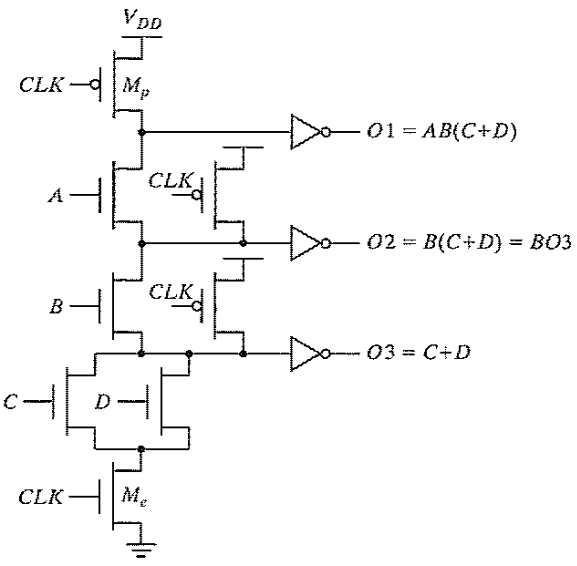


Figure 6-68 Multiple-output domino.

Compound domino (Figure 6-69) represents another optimization of the generic domino gate, once again minimizing the number of transistors. Instead of each dynamic gate driving a static inverter, it is possible to combine the outputs of multiple dynamic gates with the aid of a complex static CMOS gate, as shown in Figure 6-69. The outputs of three dynamic structures (implementing $O1 = \overline{A\ B\ C}$, $O2 = \overline{D\ E\ F}$ and $O3 = \overline{G\ H}$) are combined using a single complex CMOS static gate that implements $O = \overline{(O1 + O2)\ O3}$. The total logic function realized this way is $O = A\ B\ C\ D\ E\ F + GH$.

Compound domino is a useful tool for constructing complex dynamic logic gates. Large dynamic stacks are replaced by parallel structures with small fan-in and complex CMOS gates. For example, a large *fan-in* domino AND can be implemented as a set of parallel dynamic NAND structures with lower *fan-in,* combined with a static NOR gate. One important consideration in Compound domino is the problem associated with backgate coupling. Care must be taken to ensure that the dynamic nodes are not affected by the coupling between the output of the static gates and the output of dynamic nodes.

np-CMOS

An alternative approach to cascading dynamic logic is provided by *np*-CMOS, which uses two flavors (*n*-tree and *p*-tree) of dynamic logic, and avoids the extra static inverter in the critical path that comes with domino logic. In a *p*-tree logic gate, PMOS devices are used to build a pull-up logic network, including a PMOS evaluation transistor ([Gonçalvez83, Friedman84, Lee86]).

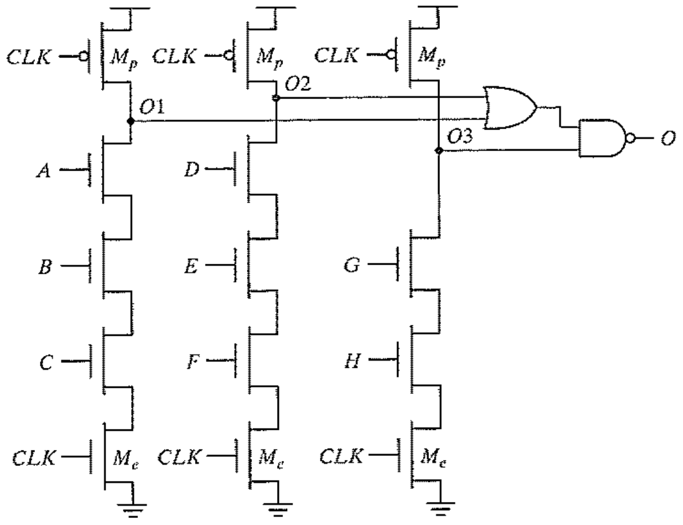


Figure 6-69 Compound domino logic uses complex static gates at the output of the dynamic gates.

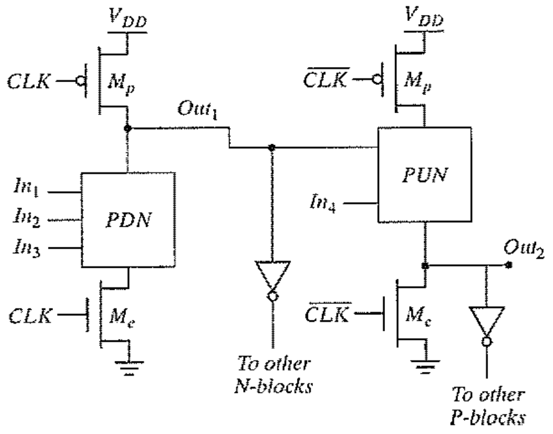


Figure 6-70 The *np*-CMOS logic circuit style.

(see Figure 6-70). The NMOS predischarge transistor drives the output low during precharge The output conditionally makes a $0 \rightarrow 1$ transition during evaluation depending on its inputs.

np-CMOS logic exploits the duality between *n*-tree and *p*-tree logic gates to eliminate the cascading problem. If the *n*-tree gates are controlled by *CLK*, and *p*-tree gates are controlled using $\overline{CLK}$, *n*-tree gates can directly drive *p*-tree gates, and vice versa. Similar to domino, *n*-tree outputs must go through an inverter when connecting to another *n*-tree gate. During the precharge phase ($CLK = 0$), the output of the n-tree gate, Out_1, is charged to V_{DD}, while the output of the p-tree gate, Out_2, is predischarged to 0 V. Since the n-tree gate connects PMOS pull-up devices, the PUN of the p-tree is turned off at that time. During evaluation, the output of the n-tree gate can only make a $1 \rightarrow 0$ transition, conditionally turning on some transistors in the p-tree. This ensures that no accidental discharge of Out_2 can occur. Similarly, n-tree blocks can follow p-tree gates without any problems, because the inputs to the n-gate are precharged to 0. A disadvantage of the np-CMOS logic style is that the p-tree blocks are slower than the n-tree modules, due to the lower current drive of the PMOS transistors in the logic network. Equalizing the propagation delays requires extra area. Also, the lack of buffers requires that dynamic nodes are routed between gates.

6.4 Perspectives

6.4.1 How to Choose a Logic Style?

In the preceding sections, we have discussed several gate-implementation approaches using the CMOS technology. Each of the circuit styles has its advantages and disadvantages. Which one to select depends upon the primary requirement: ease of design, robustness, area, speed, or power dissipation. No single style optimizes all these measures at the same time. Even more, the approach of choice may vary from logic function to logic function.

The static approach has the advantage of being robust in the presence of noise. This makes the design process rather trouble free and amenable to a high degree of automation. It is clearly the best general-purpose logic design style. This ease of design does come at a cost: For complex gates with a large fan-in, complementary CMOS becomes expensive in terms of area and performance. Alternative static logic styles have therefore been devised. Pseudo-NMOS is simple and fast at the expense of a reduced noise margin and static power dissipation. Pass-transistor logic is attractive for the implementation of a number of specific circuits, such as multiplexers and XOR-dominated logic like adders.

Dynamic logic, on the other hand, makes it possible to implement fast and small complex gates. This comes at a price, however. Parasitic effects such as charge sharing make the design process a precarious job. Charge leakage forces a periodic refresh, which puts a lower bound on the operating frequency of the circuit.

The current trend is towards an increased use of complementary static CMOS. This tendency is inspired by the increased use of design-automation tools at the logic design level. These tools emphasize optimization at the logic level, rather than at the circuit level, and they put a premium on robustness. Another argument is that static CMOS is more amenable to voltage scaling than some of the other approaches discussed in this chapter.

6.4.2 Designing Logic for Reduced Supply Voltages

In Chapter 3, we projected that the supply voltage for CMOS processes will continue to drop over the coming decade, and may go as low as 0.6 V by 2010. To maintain performance under

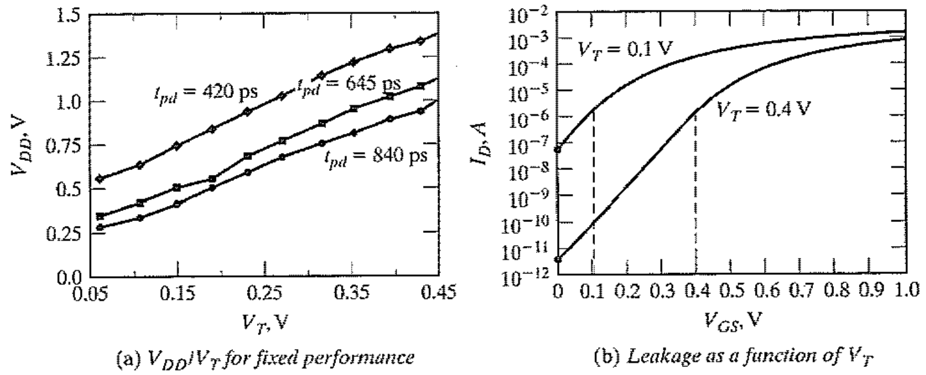


Figure 6-71 Voltage Scaling (V_{DD}/V_T on delay and leakage).

those conditions, it is essential that the device thresholds scale as well. Figure 6-71a shows a plot of the (V_T, V_{DD}) ratio required to maintain a given performance level (assuming that other device characteristics remain identical).

This trade-off is not without penalty. Reducing the threshold voltage increases the subthreshold leakage current exponentially, as we derived in Eq. (3.39) (repeated here for the sake of clarity):

$$I_{leakage} = I_S 10^{\frac{V_{GS}-V_{th}}{S}}\left(1 - 10^{-\frac{nV_{DS}}{S}}\right) \tag{6.46}$$

In Eq. (6.46), S is the *slope factor* of the device. The subthreshold leakage of an inverter is the current of the NMOS for V_{in} = 0 V and V_{out} = V_{DD} (or the PMOS current for V_{in} = V_{DD} and V_{out} = 0). The exponential increase in inverter leakage for decreasing thresholds is illustrated in Figure 6-71b.

These leakage currents are a concern particularly for designs that feature intermittent computational activity separated by long periods of inactivity. For example, the processor in a cellular phone remains in idle mode for a majority of the time. While the processor is in idle mode, ideally, the system should consume zero or near-zero power. This is only possible if leakage is low—that is, the devices have a high threshold voltage. This is in contrast to the scaling scenario that we just depicted, where high performance under low supply voltage means reduced thresholds. To satisfy the contradicting requirements of high performance during active periods and low leakage during standby, several process modifications or leakage-control techniques have been introduced in CMOS processes. Most processes with feature sizes at or below 0.18 μm CMOS support devices with different thresholds—typically a device with low threshold for high-performance circuits, and a transistor with high threshold for leakage control. Another approach gaining popularity is the dynamic control of the threshold voltage of a device by exploiting the body effect of the transistor. Use of this approach to control individual devices requires a dual-well process (see Figure 2-2).

Clever circuit design can also help reduce the leakage current, which is a function of the circuit topology and the value of the inputs applied to the gate. Since V_T depends on body bias (V_{BS}), the subthreshold leakage of an MOS transistor depends not only on the gate drive (V_{GS}), but also on the body bias. In an inverter with $In = 0$, the subthreshold leakage of the inverter is set by the NMOS transistor with its $V_{GS} = V_{BS} = 0$ V. In more complex CMOS gates, such as the two-input NAND gate of Figure 6-72, the leakage current depends on the input vector. The subthreshold leakage current of this gate is the least when $A = B = 0$. Under these conditions, the intermediate node X settles to

$$V_X \approx V_{th} \ln(1 + n) \tag{6.47}$$

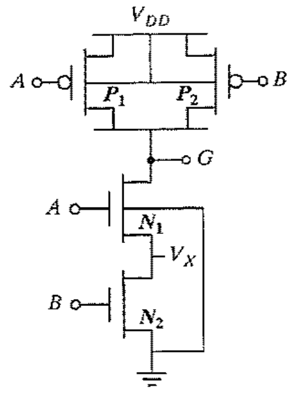
The leakage current of the gate is then determined by the topmost NMOS transistor with $V_{GS} = V_{BS} = -V_X$. Clearly, the subthreshold leakage under this condition is smaller than that of the inverter. This reduction due to stacked transistors is called the *stack effect*. The Table in Figure 6-72 analyzes the leakage components for the two-input NAND gate under different input conditions.

The reality is even better. In short-channel MOS transistors, the subthreshold leakage current depends not only on the gate drive (V_{GS}) and the body bias (V_{BS}), but also on the drain voltage (V_{DS}). The threshold voltage of a short-channel MOS transistor decreases with increasing V_{DS} due to *drain-induced barrier lowering* (DIBL). Typical values for DIBL can range from a 20- to a 150-mV change in V_T per voltage change in V_{DS}. Because of this, the impact of the stack effect is even more significant for short-channel transistors. The intermediate voltage reduces the drain–source voltage of the topmost device, increases its threshold, and thus lowers its leakage.

Example 6.19 Stack Effect in Two-Input NAND Gate

Consider again the two-input NAND gate of Figure 6-72a, when both $N_1$and N_2 are *off* ($A = B = 0$). From the simulated load lines shown in Figure 6-72c, we see that V_X settles to approximately 100 mV in steady state. The steady-state subthreshold leakage in the gate is therefore due to $V_{GS} = V_{BS} = -100$ mV and $V_{DS} = V_{DD} - 100$ mV, which is 20 times smaller than the leakage of a stand-alone NMOS transistor with $V_{GS} = V_{BS} = 0$ mV and $V_{DS} = V_{DD}$ [Ye98].

In sum, the subthreshold leakage in complex stacked circuits can be significantly lower than in individual devices. Observe that the maximum leakage reduction occurs when all the transistors in the stack are *off*, and the intermediate node voltage reaches its steady–state value. Exploiting this effect requires a careful selection of the input signals to every gate during standby or sleep mode.



A	B	V_X	I_{SUB}
0	0	$V_{th} \ln(1+n)$	$I^N_{SUB}(V_{GS} = V_{BS} = -V_X)$
0	1	0	$I^N_{SUB}(V_{GS} = V_{BS} = 0)$
1	0	$V_{dd} - V_T$	$I^N_{SUB}(V_{GS} = V_{BS} = 0)$
1	1	0	$2\, I^P_{SUB}(V_{SG} = V_{SB} = 0)$

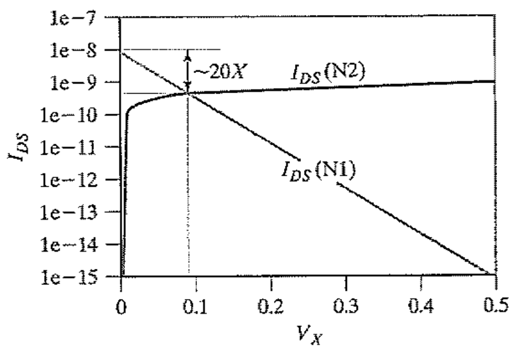


Figure 6-72 Subthreshold leakage reduction in a two-input NAND gate (a) due to stack effect for different input conditions (b). Figure (c) plots the simulated load lines of the gate for $A = B = 0$.

Problem 6.9 Computing V_X

Equation (6.47) calculates the intermediate node voltage for a two-input NAND with less than 10% error, for $A = B = 0$. Derive Eq. (6.47) assuming (1) V_T and I_S of N_1 and N_2 are approximately equal, (2) NMOS transistors are identically sized, and (3) $n < 1.5$.

6.5 Summary

In this chapter, we have extensively analyzed the behavior and performance of combinational CMOS digital circuits with regard to area, speed, and power. We summarize the major points as follows:

- *Static complementary* CMOS combines dual pull-down and pull-up networks, only one of which is enabled at any time.

- The performance of a CMOS gate is a strong function of the *fan-in*. Techniques to deal with fan-in include transistor sizing, input reordering, and partitioning. The speed is also a linear function of the fan-out. Extra buffering is needed for large fan-outs.
- The *ratioed logic* style consists of an active pull-down (-up) network connected to a load device. This results in a substantial reduction in gate complexity at the expense of static power consumption and an asymmetrical response. Careful transistor sizing is necessary to maintain sufficient noise margins. The most popular approaches in this class are the pseudo-NMOS techniques and differential DCVSL, which require complementary signals.
- *Pass-transistor logic* implements a logic gate as a simple switch network. This results in very simple implementations for some logic functions. Long cascades of switches are to be avoided due to a quadratic increase in delay with respect to the number of elements in the chain. NMOS-only pass-transistor logic produces even simpler structures, but might suffer from static power consumption and reduced noise margins. This problem can be addressed by adding a level-restoring transistor.
- The operation of *dynamic logic* is based on the storage of charge on a capacitive node and the conditional discharging of that node as a function of the inputs. This calls for a two-phase scheme, consisting of a precharge followed by an evaluation step. Dynamic logic trades off noise margin for performance. It is sensitive to parasitic effects such as leakage, charge redistribution, and clock feedthrough. Cascading dynamic gates can cause problems and thus should be addressed carefully.
- The *power consumption* of a logic network is strongly related to the switching activity of the network. This activity is a function of the input statistics, the network topology, and the logic style. Sources of power consumption such as glitches and short-circuit currents can be minimized by careful circuit design and transistor sizing.
- Threshold voltage scaling is required for *low-voltage operation*. Leakage control is critical for low-voltage operation.

6.6 To Probe Further

The topic of (C)MOS logic styles is treated extensively in the literature. Numerous texts have been devoted to the issue. Some of the most comprehensive treatments can be found in [Weste93] and [Chandrakasan01]. Regarding the intricacies of high-performance design, [Shoji96] and [Bernstein98] offer the most in-depth discussion of the optimization and analysis of digital MOS circuits. The topic of power minimization is relatively new, but comprehensive reference works are available in [Chandrakasan95], [Rabaey95], and [Pedram02].

Innovations in the MOS logic area are typically published in the proceedings of the ISSCC Conference and the VLSI circuits symposium, as well as the *IEEE Journal of Solid State Circuits* (especially the November issue).

References

[Bernstein98] K. Bernstein et al., *High-Speed CMOS Design Styles*, Kluwer Academic Publishers, 1998.

[Chandrakasan95] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[Chandrakasan01] A. Chandrakasan, W. Bowhill, and F. Fox, ed., *Design of High-Performance Microprocessor Circuits*, IEEE Press, 2001.

[Gonçalvez83] N. Gonçalvez and H. De Man, "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures," *IEEE Journal of Solid State Circuits*, vol. SC-18, no. 3, pp. 261–266, June 1983.

[Heller84] L. Heller et al., "Cascade Voltage Switch Logic: A Differential CMOS Logic Family," *Proc. IEEE ISSCC Conference*, pp. 16–17, February 1984.

[Krambeck82] R. Krambeck et al., "High-Speed Compact Circuits with CMOS," *IEEE Journal of Solid State Circuits*, vol. SC-17, no. 3, pp. 614–619, June 1982.

[Landman91] P. Landman and J. Rabaey, "Design for Low Power with Applications to Speech Coding," *Proc. International Micro-Electronics Conference*, Cairo, December 1991.

[Parameswar96] A. Parameswar, H. Hara, and T. Sakurai, "A Swing Restored Pass-Transistor Logic-Based Multiply and Accumulate Circuit for Multimedia Applications," *IEEE Journal of Solid State Circuits*, vol. SC-31, no. 6, pp. 805–809, June 1996.

[Pedram02] M. Pedram and J. Rabaey, ed., *Power-Aware Design Methodologies*, Kluwer, 2002.

[Rabaey95] J. Rabaey and M. Pedram, ed., *Low Power Design Methodologies*, Kluwer, 1995.

[Radhakrishnan85] D. Radhakrishnan, S. Whittaker, and G. Maki, "Formal Design Procedures for Pass-Transistor Switching Circuits," *IEEE Journal of Solid State Circuits*, vol. SC-20, no. 2, pp. 531–536, April 1985.

[Shoji88] M. Shoji, *CMOS Digital Circuit Technology*, Prentice Hall, 1988.

[Shoji96] M. Shoji, *High-Speed Digital Circuits*, Addison-Wesley, 1996.

[Sutherland99] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort*, Morgan Kaufmann, 1999.

[Weste93] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley, 1993.

[Yano90] K. Yano et al., "A 3.8 ns CMOS 16 × 16 b Multiplier Using Complimentary Pass-Transistor Logic," *IEEE Journal of Solid State Circuits*, vol. SC-25, no. 2, pp. 388–395, April 1990.

[Ye98] Y. Ye, S. Borkar, and V. De, "A New Technique for Standby Leakage Reduction in High-Performance Circuits," *Symposium on VLSI Circuits*, pp. 40–41, 1998.

Exercises

For the latest problem sets and design challenges in CMOS digital logic, log in to **http://bwrc.eecs.berkeley.edu/IcBook**.

DESIGN METHODOLOGY INSERT

C

How to Simulate Complex Logic Circuits

Timing- and Switch-Level Simulation

Logic and Functional Simulation

Behavioral Simulation

Register-Transfer Languages

While circuit simulation in the SPICE style proves to be an extremely valuable element of the designers tool box, it has one major deficiency. By taking into account all the peculiarities and second-order effects of the semiconductor devices, it tends to be time consuming. It rapidly becomes unwieldy when designing complex circuits, unless one is willing to spend days of computer time. Even though computers are always getting faster and simulators are getting better, circuits are getting complex even faster. The designer can address the complexity issue by giving up modeling accuracy and resorting to higher representation levels. A discussion of the different abstraction levels available to the designer and their impact on simulation accuracy is the topic of this insert.

The best way of differentiating among the myriad of simulation approaches and abstraction levels is to identify how the data and time variables are represented—as analog, continuous variables, as discrete signals, or as abstract data models.

C.1 Representing Digital Data as a Continuous Entity

Circuit Simulation and Derivatives

In Design Methodology Insert B, we established that a circuit simulator is "digitallyagnostic," meaning that it is, in essence, an analog simulator. Voltage, current, and time are treated as analog variables. This accurate modeling, combined with the nonlinearity of most of the devices leads to a high overhead in simulation time.

Substantial effort has been invested to decrease the computation time at the expense of generality. Consider an MOS digital circuit. Due to the excellent isolation property of the MOS gate, it is often possible to partition the circuit into a number of sections that have limited interaction. A possible approach is to solve each of these partitions individually over a given period, assuming that the inputs from other sections are known or constant. The resulting waveforms can then be iteratively refined. This *relaxation-based* approach has the advantage of being computationally more effective than the traditional technique by avoiding expensive matrix inversions, but it is restricted to MOS circuits [White87]. When the circuit contains feedback paths, the partitions can become large, and simulation performance degrades.

Another approach is to reduce the complexity of the transistor models used. For example, linearization of the model leads to a dramatic reduction in the computational complexity. Yet another approach is to employ a simplified table-lookup model. While this approach, by necessity, leads to a decreased accuracy of the waveforms, it still allows for a good estimation of timing parameters such as propagation delay and rise and fall times. This explains why these tools often are called *timing simulators*. The big advantage is in the execution speed, which can be one or two orders of magnitude higher than that of SPICE-like tools. Another advantage is that, in contrast to the tools that are discussed next, timing simulators still can incorporate second-order effects such as leakage, threshold drops, and signal glitches. Examples of an offering in this class is the NanoSim (formerly TimeMill/PowerMill) tool set from Synopsys [TimeMill]. As a point of reference, simulators in this class typically give up 5 to 10% in accuracy on timing parameters, with respect to full-blown circuit simulators.

C.2 Representing Data as a Discrete Entity

In digital circuits, we generally are not interested in the actual value of the voltage variable, but only in the digital value it represents. Therefore, it is possible to envision a simulator in which data signals are either in the 0 or 1 range. Signals that do no comply with either condition are denoted as X, or undefined.

This tertiary representation $\{0, 1, X\}$ is used extensively in simulators at both the device and gate level. By augmenting this set of allowable data values, we can obtain more detailed information, while retaining the capability of handling complex designs. Possible extensions are the Z-value for a tristate node in the high-impedance state, and R- and F-values for the rising and falling transients. Some commercially offered simulators provide as many as a dozen possible signal states.

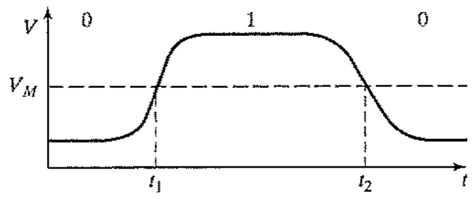


Figure C-1 Discretizing the time variable.

While substantial performance improvement is obtained by making the data representation space discrete, similar benefits can be obtained by making time a discrete variable as well. Consider the voltage waveform of Figure C-1, which represents the signal at the input of an inverter with a switching threshold V_M. It is reasonable to assume that the inverter output changes its value one propagation delay after its input crossed V_M. When one is not strictly interested in the exact shape of the signal waveforms, it is sufficient to evaluate the circuit only at the interesting time points, t_1 and t_2.

Similarly, the interesting points of the output waveform are situated at $t_1 + t_{pHL}$ and $t_2 + t_{pLH}$. A simulator that only evaluates a gate at the time an event happens at one of its inputs is called an *event-driven* simulator. The evaluation order is determined by putting projected events on a time queue and processing them in a time-ordered fashion. Suppose that the waveform of Figure C-1 acts as the input waveform to a gate. An event is scheduled to occur at time t_1. Upon processing that event, a new event is scheduled for the fan-out nodes at $t_1 + t_{pHL}$ and is put on the time queue. This event-driven approach is evidently more efficient than the time-step-driven approach of the circuit simulators. To take the impact of fan-out into account, the propagation delay of a circuit can be expressed in terms of an intrinsic delay (t_{in}) and a load-dependent factor (t_l), and it can differ over edge transitions:

$$t_{pLH} = t_{inLH} + t_{lLH} \times C_L \qquad \text{(C.1)}$$

The load C_L can be entered in absolute terms (in pF) or as a function of the number of fan-out gates. Observe how closely this equation resembles the *logical-effort* model we introduced in the preceding chapter.

While offering a substantial performance benefit, the preceding approach still has the disadvantage that events can happen any time. Another simplification could be to make the time even more discrete and allow events to happen only at integer multiples of a *unit time* variable. An example of such an approach is the *unit-delay* model, where each circuit has a single delay of one unit. Finally, the simplest model is the *zero-delay model,* in which gates are assumed to be free of delay. Under this paradigm, time proceeds from one clock event to the next, and all events are assumed to occur instantaneously upon arrival of a clocking event. These concepts can be applied on a number of abstraction levels, resulting in the simulation approaches discussed next.

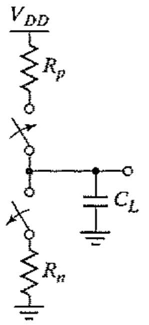


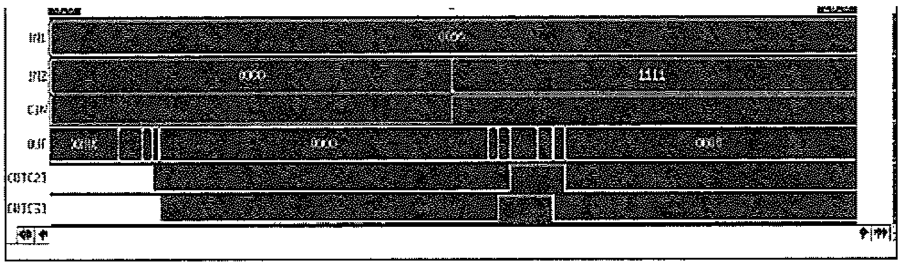
Figure C-2 Switch-level model of CMOS inverter.

Switch-Level Simulation

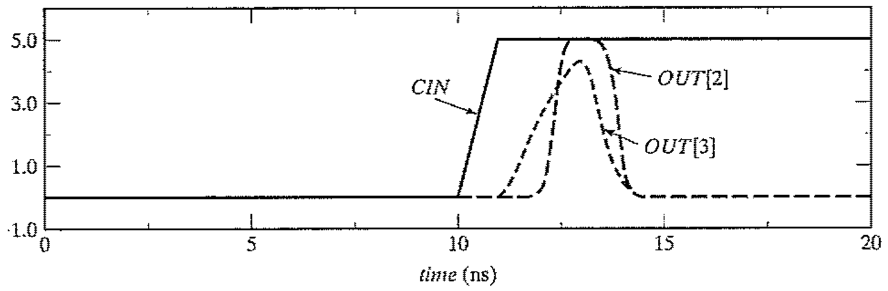
The nonlinear nature of semiconductor devices is one of the major impediments to higher simulation speeds. The switch-level model [Bryant81] overcomes this hurdle by approximating the transistor behavior with a linear resistance whose value is a function of the operating conditions. In the off-mode, the resistance is set to infinity, while in the on-mode, it is set to the average "on" resistance of the device (Figure C-2). The resulting network is a time-variant, linear network of resistors and capacitors that can be more efficiently analyzed. Evaluation of the resistor network determines the steady-state values of the signals and typically employs a $\{0, 1, X\}$ model. For instance, if the total resistance between a node and GND is substantially smaller than the resistance to V_{DD}, the node is set to the 0-state. The timing of the events can be resolved by analyzing the *RC* network. Simpler timing models such as the unit-delay model are also employed.

Example C.1 Switch versus Circuit-Level Simulation

A four-bit adder is simulated using the switch-level simulator IRSIM ([Salz89]). The simulation results are plotted in Figure C-3. Initially, all inputs (*IN*1 and *IN*2) and the carry-input *CIN* are set to 0. After 10 nsec, all inputs *IN*2 as well as *CIN* are set to 1. The display window plots the input signals, the output vector *OUT*[0–3], and the most significant output bits *OUT*[2] and *OUT*[3]. The output converges to the correct value 0000 after a transition period. Notice how the data assumes only 0 and 1 levels. The glitches in the output signals go rail to rail, although in reality they might represent only partial excursions. During transients, the signal is marked *X*, which means "undefined." To put this result in perspective, Figure C-3 plots the SPICE results for the same input vectors. Notice the partial glitches. Also, it shows that the IRSIM timing, which is based on an *RC* model, is relatively accurate and sufficient to get a first-order impression.



(a) IRSIM results.



(b) SPICE results.

Figure C-3 Comparison between circuit and switch-level simulations.

Gate-Level (or Logic) Simulation

Gate-level simulators use the same signal values as the switch-level tools, but the simulation primitives are gates instead of transistors. This approach enables the simulation of more complex circuits at the expense of detail and generality. For example, some common VLSI structures such as tristate busses and pass transistors are hard to deal with at this level. Since gate level is the preferred entry level for many designers, this simulation approach remained extremely popular until the introduction of logic synthesis tools, which moved the focus to the functional or behavioral abstraction layer. The interest in logic simulation was so great that special and expensive hardware accelerators were developed to expedite the simulation process (e.g., [Agrawal90]).

Functional Simulation

Functional simulation can be considered as a simple extension of logic simulation. The primitive elements of the input description can be of an arbitrary complexity. For instance, a simulation element can be a NAND gate, a multiplier, or an SRAM memory. The functionality of one of these complex units can be described using a modern programming language or a dedicated hardware description language. For instance, the THOR simulator uses the C programming language to determine the output values of a module as a function of its inputs [Thor88].

The *SystemC* language [SystemC] uses most of the syntax and semantics of C, but adds a number of constructs and data types to deal with the peculiarities of hardware design—such as the presence of concurrency. On the other hand, *VHDL (VHSIC Hardware Description Language)* [VHDL88] is a specially developed language for the description of hardware designs.

In the *structural mode*, VHDL describes a design as a connection of functional modules. Such a description often is called a *netlist*. For example, Figure C-4 shows a description of a 16-bit accumulator consisting of a register and adder.

The adder and register can in turn be described as a composition of components such as full-adder or register cells. An alternative approach is to use the *behavioral mode* of the language that describes the functionality of the module as a set of input/output relations regardless of the chosen implementation. As an example, Figure C-5 describes how the output of the adder is the two's-complement sum of its inputs.

```
entity accumulator is
    port ( -- definition of input and output terminals
        DI: in bit_vector(15 downto 0) -- a vector of 16 bit wide
        DO: inout bit_vector(15 downto 0);
        CLK: in bit
    );
end accumulator;

architecture structure of accumulator is
    component reg -- definition of register ports
        port (
            DI : in bit_vector(15 downto 0);
            DO : out bit_vector(15 downto 0);
            CLK : in bit
        );
    end component;
    component add -- definition of adder ports
        port (
            IN0 : in bit_vector(15 downto 0);
            IN1 : in bit_vector(15 downto 0);
            OUT0 : out bit_vector(15 downto 0)
        );
    end component;
-- definition of accumulator structure
signal X : bit_vector(15 downto 0);
begin
    add1 : add
        port map (DI, DO, X); -- defines port connectivity
    reg1 : reg
        port map (X, DO, CLK);
end structure;
```

Figure C-4 Functional description of an accumulator in VHDL.

```
entity add is
   port (
      IN0 : in bit_vector(15 downto 0);
      IN1 : in bit_vector(15 downto 0);
      OUT0 : out bit_vector(15 downto 0)
   );
end add;

architecture behavior of add is
begin
   process(IN0, IN1)
      variable C : bit_vector(16 downto 0);
      variable S : bit_vector(15 downto 0);
   begin
      loop1:
      for i in 0 to 15 loop
         S(i) := IN0(i) xor IN1(i) xor C(i);
         C(i+1):= IN0(i) and IN1(i) or C(i) and (IN0(i) or IN1(i));
      end loop loop1;
      OUT0 <= S;
   end process;
end behavior;
```

Figure C-5 Behavioral description of 16-bit adder.

The signal levels of the functional simulator are similar to the switch and logic levels. A variety of timing models can be used—for example, the designer can describe the delay between input and output signals as part of the behavioral description of a module. Most often the zero-delay model is employed, since it yields the highest simulation speed.

C.3 Using Higher-Level Data Models

When conceiving a digital system such as a compact disk player or an embedded microcontroller, the designer rarely thinks in terms of bits. Instead, she envisions data moving over busses as integer or floating-point words, and patterns transmitted over the instruction bus as members of an enumerated set of instruction words (such as {*ACC*, *RD*, *WR*, or *CLR*}). Modeling a discrete design at this level of abstraction has the distinct advantage of being more understandable, and it also results in a substantial benefit in simulation speed. Since a 64-bit bus is now handled as a single object, analyzing its value requires only one action instead of the 64 evaluations it formerly took to determine the current state of the bus at the logic level. The disadvantage of this approach is another sacrifice of timing accuracy. Since a bus is now considered to be a single entity, only one global delay can be annotated to it, while the delay of bus elements can vary from bit to bit at the logic level.

It also is common to distinguish between *functional* (or *structural)* and *behavioral* descriptions. In a functional-level specification, the description mirrors the intended hardware structure. Behavioral-level specifications only mimic the input/output functionality of a design. Hardware delay loses its meaning, and simulations are normally performed on a per clock-cycle (or higher) basis. For instance, the behavioral models of a microprocessor that are used to verify the completeness and the correctness of the instruction set are performed on a per instruction basis.

The most popular languages at this level of abstraction are the VHDL and VERILOG hardware-description languages. VHDL allows for the introduction of user-defined data types such as 16-bit, two's-complement words or enumerated instruction sets. Many designers tend to use traditional programming approaches such as C or C++ for their first-order behavioral models. This approach has the advantage of offering more flexibility, but it requires the user to define all data types and to essentially write the complete simulation scenario.

Example C.2 Behavioral-Level VHDL Description

To contrast the functional and behavioral description modes and the use of higher level data models, consider again the example of the accumulator (see Figure C-6). In this case, we use a fully behavioral description that employs integer data types to describe the module operation.

Figure C-7 shows the results of a simulation performed at this level of abstraction. Even for this small example, the simulation performance in terms of CPU time is three times better than what is obtained with the structural description of Figure C-4.

```
entity accumulator is
   port (
      DI : in integer;
      DO : inout integer := 0;
      CLK : in bit
   );
end accumulator;

architecture behavior of accumulator is
begin
   process(CLK)
   variable X : integer := 0; -- intermediate variable
   begin
      if CLK = '1' then
         X <= DO + DI;
         DO <= X;
      end if;
   end process;
end behavior;
```

Figure C-6 Accumulator for Example C.2.

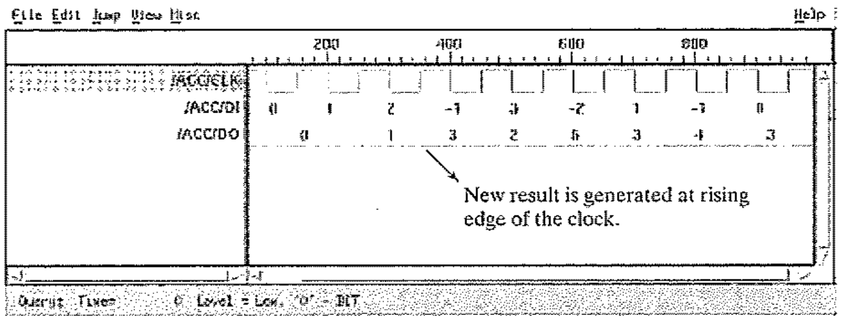


Figure C-7 Display of simulation results for accumulator example as obtained at the behavioral level. The WAVES display tool (and VHDL simulator) are part of the Synopsis VHDL tool suite (Courtesy of Synopsys.).

References

[Agrawal90] P. Agrawal and W. Dally, "A Hardware Logic Simulation System," *IEEE Trans. Computer-Aided Design*, CAD-9, no. 9, pp. 19–29, January 1990.

[Bryant81] R. Bryant, *A Switch-Level Simulation Model for Integrated Logic Circuits*, Ph. D. diss., MIT Laboratory for Computer Science, report MIT/LCS/TR-259, March 1981.

[Salz89] A. Salz and M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," *Proceedings of the 26th Design Automation Conference*, pp. 173–178, 1989.

[SystemC] *Everything You Wanted to Know about SystemC, http://www.systemc.org/*

[Thor88] R. Alverson et al., "THOR User's Manual," Technical Report CSL-TR-88-348 and 349, Stanford University, January 1988.

[TimeMill] *http://www.synopsys.com/products/mixedsignal/mixedsignal.html*

[VHDL88] VHDL Standards Committee, IEEE Standard VHDL Language Reference Manual, IEEE standard 1076–1077, 1978.

[White87] J. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits*, Kluwer Academic, 1987.

DESIGN METHODOLOGY INSERT

D

Layout Techniques for Complex Gates

Weinberger and standard-cell layout techniques

Euler graph approach

In Chapter 6, we discussed in detail how to construct the schematics of complex gates and how to size the transistors. The last step in the design process is to derive a layout for the gate or cell; in other words, we must determine the exact shape of the various polygons composing the gate layout. The composition of a layout is strongly influenced by the *interconnect approach.* How does the cell fit in the overall chip layout, and how does it communicate with neighboring cells? Keeping these questions in mind from the start results in denser designs with less parasitic capacitance.

Weinberger and Standard-Cell Layout Techniques

We now examine two important layout approaches, although many others can be envisioned. In the *Weinberger approach* [Weinberger67], the data wires (inputs and outputs) are routed (in metal) parallel to the supply rails and perpendicular to the diffusion areas, as illustrated in Figure D-1. Transistors are formed at the cross points of the polysilicon signal wires (connected to the horizontal metal wires) and the diffusion zones. The "over-the-cell" wiring approach makes the Weinberger technique particularly suited for bit-sliced datapaths. While it is still used on an occasional base, the Weinberger technique has lost its appeal over the years in favor of the standard-cell style.

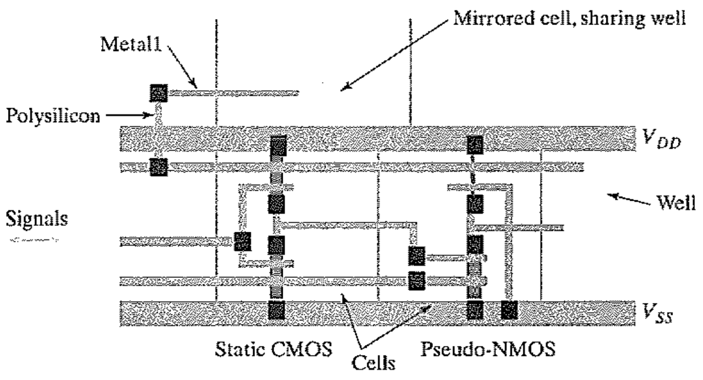


Figure D-1 The Weinberger approach for complex gate layout (using a single metal layer).

In the *standard-cell technique*, signals are routed in polysilicon perpendicular to the power distribution (Figure D-2). This approach tends to result in a dense layout for static CMOS gates, as the vertical polysilicon wire can serve as the input to both the NMOS and the PMOS transistors. An example of a cell implemented using the standard-cell approach is shown in Figure 6-12. Interconnections between cells generally are established in so-called routing channels, as demonstrated in Figure D-2. The standard-cell approach is very popular at present due to its high degree of automation. (For a detailed description of the design automation tools supporting the standard-cell approach, see Chapter 8.)

Layout Planning using the Euler Path Approach

The common use of this layout strategy makes it worth analyzing how a complex Boolean function can be mapped efficiently onto such a structure. For density reasons, it is desirable to realize the NMOS and PMOS transistors as an unbroken row of devices with abutting source–drain con-

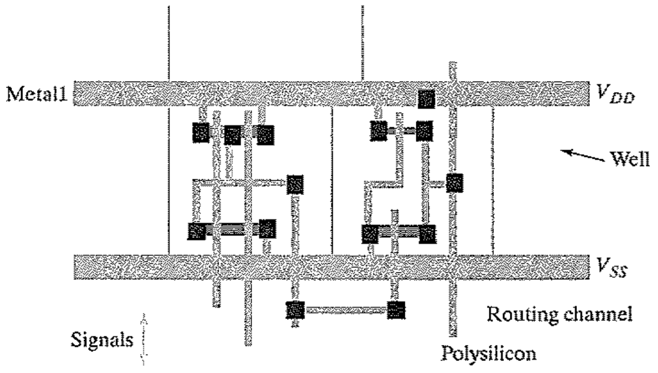
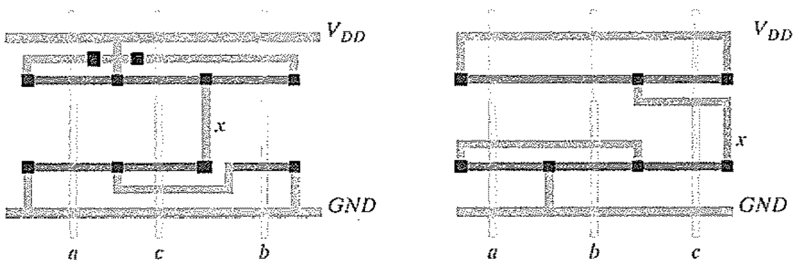


Figure D-2 The standard-cell approach for complex gate layout.



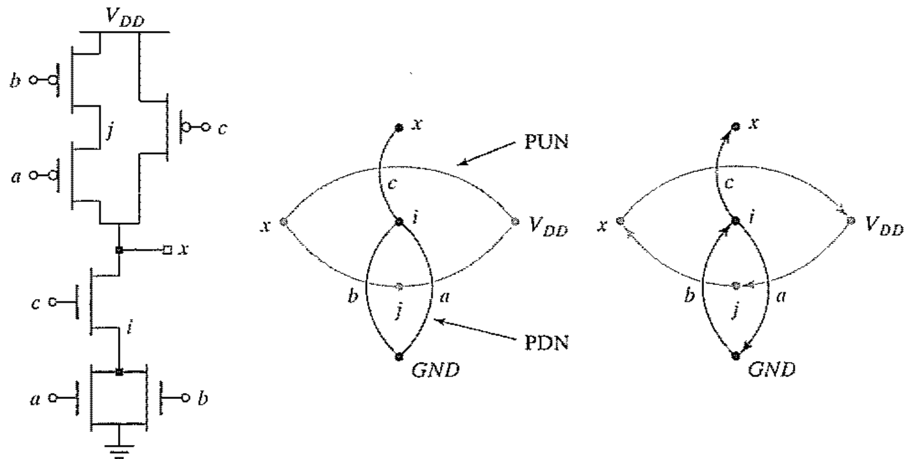
(a) Input order $\{a\ c\ b\}$ (b) Input order $\{a\ b\ c\}$

Figure D-3 Stick Diagram for $x = (a + b) \cdot c$.

nections, and with the gate connections of the corresponding NMOS and PMOS transistors aligned. This approach requires only a single strip of diffusion in both wells. To achieve this goal, a careful ordering of the input terminals is necessary. This is illustrated in Figure D-3, where the logical function $\bar{x} = (a + b) \cdot c$ is implemented. In the first version, the order $\{a\ c\ b\}$ is adopted. It can easily be seen that no solution will be found using only a single diffusion strip. A reordering of the terminals (for instance, using $\{a\ b\ c\}$), generates a feasible solution, as shown in Figure D-3b. Observe that the "layouts" in Figure D-3 do not represent actual mask geometries, but are rather symbolic diagrams of the gate topologies. Wires and transistors are represented as dimensionless objects, and positioning is relative, not absolute. Such conceptual representations are called *stick diagrams*, and often are used at the conception time of the gate, before determining the actual dimensions. We use stick diagrams whenever we want to discuss gate topologies or layout strategies.

Fortunately, a systematic approach has been developed to derive the permutation of the input terminals so that complex functions can be realized by uninterrupted diffusion strips that minimize the area [Uehara81]. The systematic nature of the technique also has the advantage that it is easily automated. It consists of the following two steps:

1. **Construction of *logic graph*.** The logic graph of a transistor network (or a switching function) is the graph of which the vertices are the nodes (signals) of the network, and the edges represent the transistors. Each edge is named for the signal controlling the corresponding transistor. Since the PUN and PDN networks of a static CMOS gate are dual, their corresponding graphs are dual as well—that is, a parallel connection is replaced by a series one and vice versa. This is demonstrated in Figure D-4, where the logic graphs for the PDN and PUN networks of the Boolean function $\bar{x} = (a + b) \cdot c$ are overlaid (notice that this approach can be used to derive dual networks).
2. **Identification of Euler paths.** An Euler path in a graph is defined as a path through all nodes in the graph such that each edge in the graph is only visited once. Identification of such a path is important, because an ordering of the inputs leading to an uninterrupted diffusion strip of NMOS (PMOS) transistors is possible only if there exists an Euler path in



(a) Schematic diagram (b) Logic graphs for PDN and PUN (c) Consistent Euler paths for PDN and PUN

Figure D-4 Schematic diagram, logic graph, and Euler paths for $x = (a + b) \cdot c$.

the logic graph of the PDN (PUN) network. The reasoning behind this finding is as follows:

To form an interrupted strip of diffusion, all transistors must be visited in sequence; that is, the drain of one device is the source of the next one. This is equivalent to traversing the logic graph along an Euler path. Be aware that Euler paths are not unique: many different solutions may exist.

The sequence of edges in the Euler path equals the ordering of the inputs in the gate layout. To obtain the same ordering in both the PUN and PDN networks, as is necessary if we want to use a single poly strip for every input signal, the Euler paths must be *consistent*—that is, they must have the same sequence.

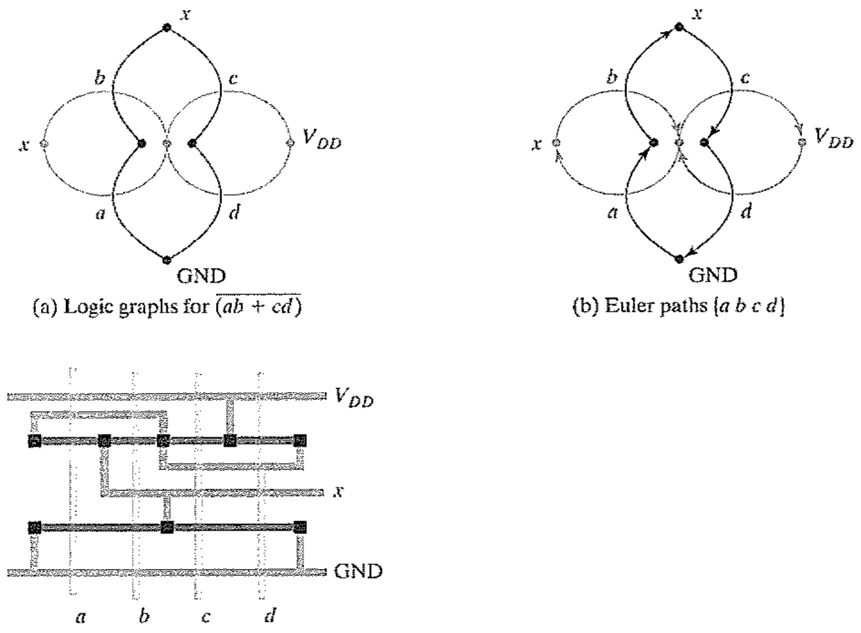
Consistent Euler paths for the example of Figure D-4a are shown in Figure D-4c. The layout associated with this solution is shown in Figure D-3b. An inspection of the logic diagram of the function shows that $\{a\ c\ b\}$ is an Euler path for the PUN, but not for the PDN. A single-diffusion-strip solution is, hence, nonexistent (Figure D-3a).

Example D.1 Derivation of Layout Topology of Complex Logic Gate

As an example, let us derive the layout topology of the following logical function:

$$x = \overline{ab + cd}$$

The logical function and one consistent Euler path are shown in Figure D-5a and Figure D-5b. The corresponding layout is shown in Figure D-5c.



(a) Logic graphs for $\overline{(ab + cd)}$

(b) Euler paths $\{a\ b\ c\ d\}$

(c) Stick diagram for ordering $\{a\ b\ c\ d\}$

Figure D-5 Deriving the layout topology for $x = (ab + cd)$.

The reader should be aware that the existence of consistent Euler paths depends on the way the Boolean expressions (and the corresponding logic graphs) are constructed. For example, no consistent Euler paths can be found for $\bar{x} = a + b \cdot c + d \cdot e$, but the function $\bar{x} = b \cdot c + a + d \cdot e$ has a simple solution (confirm that this is true by preserving the ordering of the function when constructing the logic graphs). A restructuring of the function is sometimes necessary before a set of consistent paths can be identified. This could lead to an exhaustive search over all possible path combinations. Fortunately, a simple algorithm to avoid this plight has been proposed [Uehara81]. A discussion of this is beyond our scope, however, and we refer the interested reader to that text.

Finally, it is worth mentioning that the layout strategies presented are not the only possibilities. For example, sometimes it might be more effective to provide multiple diffusion strips stacked vertically. In this case, a single polysilicon input line can serve as the input for multiple transistors. This might be beneficial for certain gate structures, such as the NXOR gate, and therefore, case-by-case analysis is recommended.

To Probe Further

A good overview of cell-generation techniques can be found in [Rubin87, pp. 116–128]. Some of the landmark papers in this area include the following:

[Clein00] D. Clein, *CMOS IC Layout*, Newnes, 2000.

[Rubin87] S. Rubin, *Computer Aids for VLSI Design*, Addison-Wesley, 1987.

[Uehara81] T. Uehara and W. Van Cleemput, "Optimal Layout of CMOS Functional Arrays," *IEEE Trans. on Computers*, vol. C-30, no. 5, pp. 305–311, May 1981.

[Weinberger67] A. Weinberger, "Large Scale Integration of MOS Complex Logic: A Layout Method," *IEEE Journal of Solid State Circuits*, vol. 2, no. 4, pp. 182–190, 1967.

CHAPTER

7

Designing Sequential Logic Circuits

Implementation techniques for registers, latches, flip-flops, oscillators, pulse generators, and Schmitt triggers

Static versus dynamic realization

Choosing clocking strategies

7.1 Introduction

As described earlier, combinational logic circuits have the property that the output of a logic block is only a function of the *current* input values, assuming that enough time has elapsed for the logic gates to settle. Still, virtually all useful systems require storage of state information, leading to another class of circuits called *sequential logic* circuits. In these circuits, the output depends not only on the *current* values of the inputs, but also on *preceding* input values. In other words, a sequential circuit remembers some of the past history of the system—it has memory.

Figure 7-1 shows a block diagram of a generic *finite-state machine* (FSM) that consists of combinational logic and registers, which hold the system state. The system depicted here belongs to the class of *synchronous* sequential systems, in which all registers are under control of a single global clock. The outputs of the FSM are a function of the current *Inputs* and the *Current State*. The *Next State* is determined based on the *Current State* and the current *Inputs* and is fed to the inputs of registers. On the rising edge of the clock, the *Next State* bits are copied to the outputs of the registers (after some propagation delay), and a new cycle begins. The register then ignores changes in the input signals until the next rising edge. In general, registers can be *positive edge triggered* (where the input data is copied on the rising edge of the clock) or *negative edge triggered* (where the input data is copied on the falling edge, as indicated by a small circle at the clock input).

This chapter discusses the CMOS implementation of the most important sequential building blocks. A variety of choices in sequential primitives and clocking methodologies exist; making the correct selection is getting increasingly important in modern digital circuits, and can

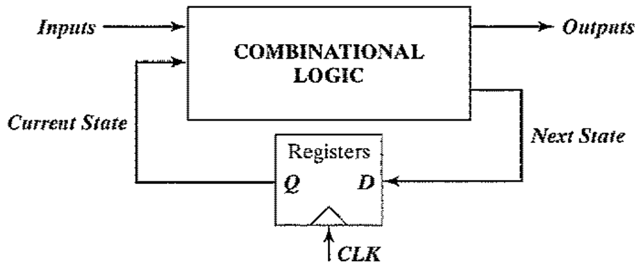


Figure 7-1 Block diagram of a finite-state machine, using *positive edge-triggered* registers.

have a great impact of performance, power, and/or design complexity. Before embarking on a detailed discussion of the various design options, a review of the relevant design metrics and a classification of the sequential elements is necessary.

7.1.1 Timing Metrics for Sequential Circuits

There are three important timing parameters associated with a register. They are shown in Figure 7-2. The *setup time* (t_{su}) is the time that the data inputs (D) must be valid before the clock transition (i.e., the $0 \rightarrow 1$ transition for a *positive edge-triggered* register). The *hold time* (t_{hold}) is the time the data input must remain valid after the clock edge. Assuming that the *setup* and *hold* times are met, the data at the D input is copied to the Q output after a worst case *propagation delay* (with reference to the clock edge) denoted by t_{c-q}.

Once we know the timing information for the registers and the combinational logic blocks, we can derive the system-level timing constraints (see Figure 7-1 for a simple system view). In synchronous sequential circuits, switching events take place concurrently in response to a clock stimulus. Results of operations await the next clock transitions before progressing to the next stage. In other words, the next cycle cannot begin unless all current computations have completed and the system has come to rest. The *clock period* T, at which the sequential circuit operates, must thus accommodate the longest delay of any stage in the network. Assume that the worst case propagation delay of the logic equals t_{plogic}, while its minimum delay—also called the *contamination delay*—is t_{cd}. The minimum clock period T required for proper operation of the sequential circuit is given by

$$T \geq t_{c-q} + t_{plogic} + t_{su} \tag{7.1}$$

The *hold time* of the register imposes an extra constraint for proper operation, namely

$$t_{cdregister} + t_{cdlogic} \geq t_{hold} \tag{7.2}$$

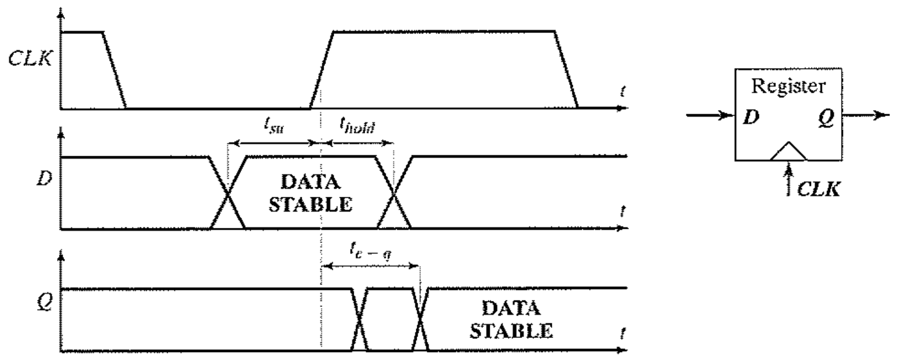


Figure 7-2 Definition of *setup time, hold time,* and *propagation delay* of a synchronous register.

where $t_{cdregister}$ is the minimum *propagation delay* (or *contamination delay*) of the register. This constraint ensures that the input data of the sequential elements is held long enough after the clock edge and is not modified too soon by the new wave of data coming in.

As seen from Eq. (7.1), it is important to minimize the values of the timing parameters associated with the register, as these directly affect the rate at which a sequential circuit can be clocked. In fact, modern high-performance systems are characterized by a very low logic depth, and the register *propagation delay* and *setup* times account for a significant portion of the clock period. For example, the DEC Alpha EV6 microprocessor [Gieseke97] has a maximum logic depth of 12 gates, and the register overhead stands for approximately 15% of the clock period. In general, the requirement of Eq. (7.2) is not difficult to meet, although it becomes an issue when there is little or no logic between registers.[1]

7.1.2 Classification of Memory Elements

Foreground versus Background Memory

At a high level, memory is classified into background and foreground memory. Memory that is embedded into logic is *foreground memory* and is most often organized as individual registers or register banks. Large amounts of centralized memory core are referred to as *background memory*. Background memory, discussed in Chapter 12, achieves higher area densities through efficient use of array structures and by trading off performance and robustness for size. In this chapter, we focus on foreground memories.

Static versus Dynamic Memory

Memories can be either static or dynamic. Static memories preserve the state as long as the power is turned on. They are built by using *positive feedback* or regeneration, where the circuit topology consists of intentional connections between the output and the input of a combinational circuit. Static memories are most useful when the register will not be updated for extended periods of time. Configuration data, loaded at power-up time, is a good example of static data. This condition also holds for most processors that use conditional clocking (i.e., gated clocks) where the clock is turned off for unused modules. In that case, there are no guarantees on how frequently the registers will be clocked, and static memories are needed to preserve the state information. Memory based on positive feedback falls under the class of elements called *multivibrator circuits*. The *bistable* element is its most popular representative, but other elements such as *monostable* and *astable* circuits also are frequently used.

Dynamic memories store data for a short period of time, perhaps milliseconds. They are based on the principle of temporary *charge storage* on parasitic capacitors associated with MOS devices. As with dynamic logic, discussed earlier, the capacitors have to be refreshed periodically to compensate for charge leakage. Dynamic memories tend to be simpler, resulting in significantly higher performance and lower power dissipation. They are most useful in datapath

[1]Or when the clocks at different registers are somewhat out of phase due to clock skew. We discuss this topic in Chapter 10.

circuits that require high performance levels and are periodically clocked. It is possible to use dynamic circuitry even when circuits are conditionally clocked, if the state can be discarded when a module goes into idle mode.

Latches versus Registers

A latch is an essential component in the construction of an *edge-triggered* register. It is a *level-sensitive* circuit that passes the D input to the Q output when the clock signal is high. This latch is said to be in *transparent* mode. When the clock is low, the input data sampled on the falling edge of the clock is held stable at the output for the entire phase, and the latch is in *hold* mode. The inputs must be stable for a short period around the falling edge of the clock to meet setup and hold requirements. A latch operating under these conditions is a *positive latch*. Similarly, a *negative latch* passes the D input to the Q output when the clock signal is low. Positive and negative latches are also called *transparent high* or *transparent low*, respectively. The signal waveforms for a positive and negative latch are shown in Figure 7-3. A wide variety of static and dynamic implementations exists for the realization of latches.

Contrary to *level-sensitive* latches, *edge-triggered* registers only sample the input on a clock transition—that is, $0 \rightarrow 1$ for a *positive edge-triggered* register, and $1 \rightarrow 0$ for a *negative edge-triggered* register. They are typically built to use the latch primitives of Figure 7-3. An often-recurring configuration is the *master–slave* structure, that cascades a positive and negative latch. Registers also can be constructed by using one-shot generators of the clock signal ("glitch" registers), or by using other specialized structures. Examples of these are shown later in this chapter.

The literature on sequential circuits has been plagued by ambiguous definitions for the different types of storage elements (i.e., register, flip-flop, and latch). To avoid confusion, we adhere strictly to the following set of definitions in this book:

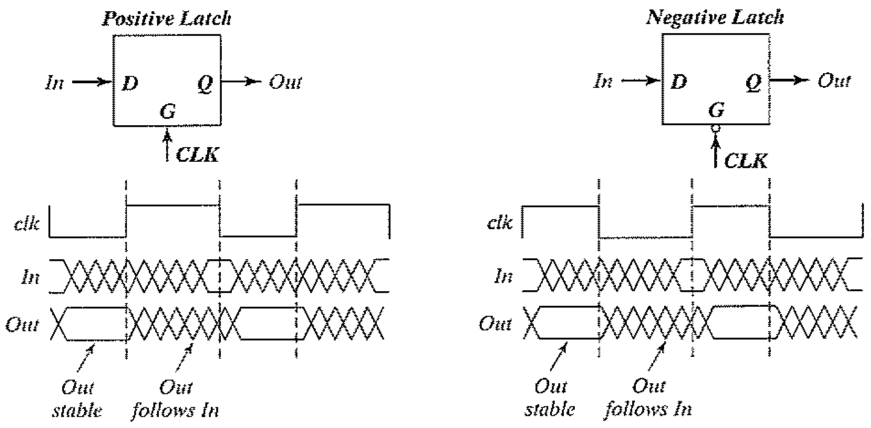


Figure 7-3 Timing of positive and negative latches.

- An *edge-triggered* storage element is called a *register*;
- A *latch* is a *level-sensitive* device;
- and any *bistable* component, formed by the cross coupling of gates, is called a *flip-flop*.[2]

7.2 Static Latches and Registers

7.2.1 The Bistability Principle

Static memories use positive feedback to create a *bistable circuit*—a circuit having two stable states that represent 0 and 1. The basic idea is shown in Figure 7-4a, which shows two inverters connected in cascade along with a voltage-transfer characteristic typical of such a circuit. Also plotted are the VTCs of the first inverter—that is, V_{o1} versus V_{i1}—and the second inverter (V_{o2} versus V_{o1}). The latter plot is rotated to accentuate that $V_{i2} = V_{o1}$. Assume now that the output of the second inverter V_{o2} is connected to the input of the first V_{i1}, as shown by the dotted lines in Figure 7-4a. The resulting circuit has only three possible operation points (A, B, and C), as demonstrated on the combined VTC. It is easy to prove the validity of the following important conjecture:

When the gain of the inverter in the transient region is larger than 1, A and B are the only stable operation points, and C is a metastable operation point.

Suppose that the cross-coupled inverter pair is biased at point C. A small deviation from this bias point, possibly caused by noise, is amplified and *regenerated* around the circuit loop.

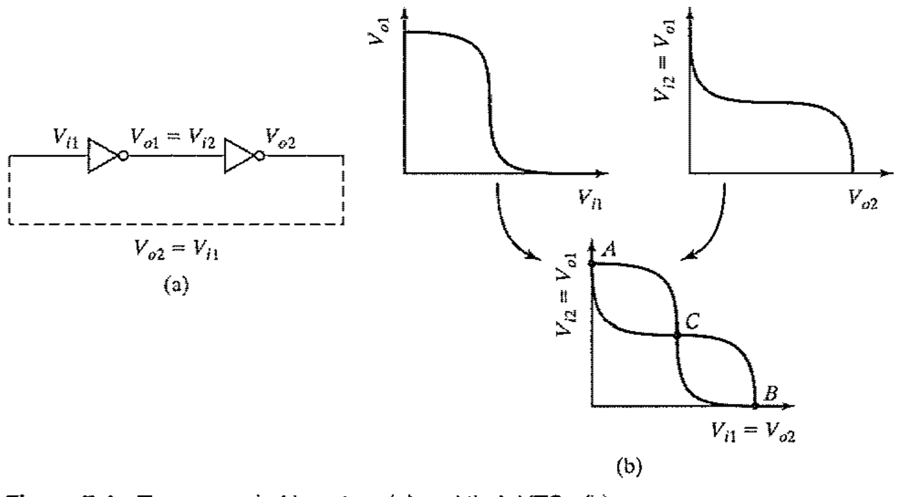


Figure 7-4 Two cascaded inverters (a) and their VTCs (b).

[2]An edge-triggered register is often referred to as a flip-flop as well. In this text, flip-flop is used to **uniquely** mean bistable element.

Dell Ex. 1025
Page 219

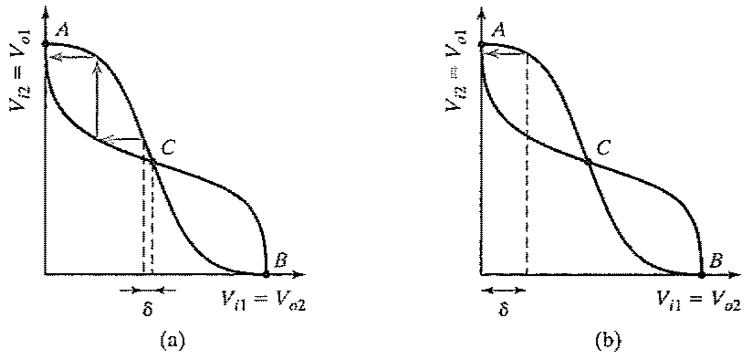


Figure 7-5 Metastable versus stable operation points.

This is a result of the gain around the loop being larger than 1. The effect is demonstrated in Figure 7-5a. A small deviation δ is applied to V_{i1} (biased in C). This deviation is amplified by the gain of the inverter. The enlarged divergence is applied to the second inverter and amplified once more. The bias point moves away from C until one of the operation points A or B is reached. In conclusion, C is an unstable operation point. Every deviation (even the smallest one) causes the operation point to run away from its original bias. The chance is indeed very small that the cross-coupled inverter pair is biased at C and stays there. Operation points with this property are termed *metastable.*

On the other hand, A and B are stable operation points, as demonstrated in Figure 7-5b. In these points, the **loop gain is much smaller than unity**. Even a rather large deviation from the operation point reduces in sizesand disappears.

Hence, the cross coupling of two inverters results in a *bistable* circuit—that is, a circuit with two stable states, each corresponding to a logic state. The circuit serves as a memory, storing either a 1 or a 0 (corresponding to positions A and B).

In order to change the stored value, we must be able to bring the circuit from state A to B and vice versa. Since the precondition for stability is that the loop gain G is smaller than unity, we can achieve this by making A (or B) temporarily unstable by increasing G to a value larger than 1. This is generally done by applying a trigger pulse at V_{i1} or V_{i2}. For example, assume that the system is in position A ($V_{i1} = 0$, $V_{i2} = 1$). Forcing V_{i1} to 1 causes both inverters to be on simultaneously for a short time and the loop gain G to be larger than 1. The positive feedback regenerates the effect of the trigger pulse, and the circuit moves to the other state (B, in this case). The width of the trigger pulse need be only a little larger than the total propagation delay around the circuit loop, which is twice the average propagation delay of the inverters.

In summary, a bistable circuit has two stable states. In absence of any triggering, the circuit remains in a single state (assuming that the power supply remains applied to the circuit) and thus remembers a value. Another common name for a bistable circuit is *flip-flop.* A flip-flop is useful only if there also exists a means to bring it from one state to the other one. In general, two different approaches may be used to accomplish the following:

- **Cutting the feedback loop.** Once the feedback loop is open, a new value can easily be written into *Out* (or *Q*). Such a latch is called *multiplexer based*, as it realizes that the logic expression for a synchronous latch is identical to the multiplexer equation:

$$Q = \overline{Clk} \cdot Q + Clk \cdot In \tag{7.3}$$

  This approach is the most popular in today's latches, and thus forms the bulk of this section.
- **Overpowering the feedback loop.** By applying a trigger signal at the input of the flip-flop, a new value is forced into the cell by overpowering the stored value. A careful sizing of the transistors in the feedback loop and the input circuitry is necessary to make this possible. A weak trigger network may not succeed in overruling a strong feedback loop. This approach used to be in vogue in the earlier days of digital design, but has gradually fallen out of favor. It is, however, the dominant approach to the implementation of static background memories (which we discuss more fully in Chapter 12). A short introduction will be given later in the chapter.

7.2.2 Multiplexer-Based Latches

The most robust and common technique to build a latch involves the use of transmission-gate multiplexers. Figure 7-6 shows an implementation of positive and negative static latches based on multiplexers. For a negative latch, input 0 of the multiplexer is selected when the clock is low, and the *D* input is passed to the output. When the clock signal is high, input 1 of the multiplexer, which connects to the output of the latch, is selected. The feedback ensures a stable output as long as the clock is high. Similarly in the positive latch, the *D* input is selected when the clock signal is high, and the output is held (using feedback) when the clock signal is low.

A transistor-level implementation of a positive latch based on multiplexers is shown in Figure 7-7. When *CLK* is high, the bottom transmission gate is on and the latch is transparent—that is, the *D* input is copied to the *Q* output. During this phase, the feedback loop is open, since the top transmission gate is off. Sizing of the transistors therefore is not critical for realizing correct functionality. The number of transistors that the clock drives is an important metric from a power perspective, because the clock has an *activity factor* of 1. This particular latch implemen-

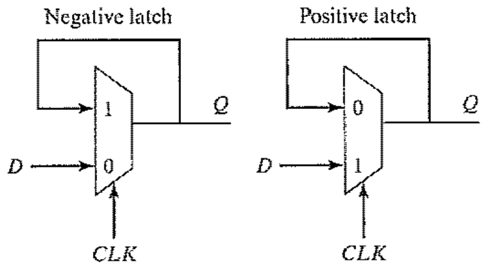


Figure 7-6 Negative and positive latches based on multiplexers.

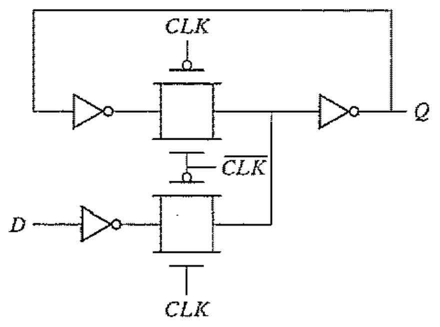


Figure 7-7 Transistor-level implementation of a positive latch built by using transmission gates.

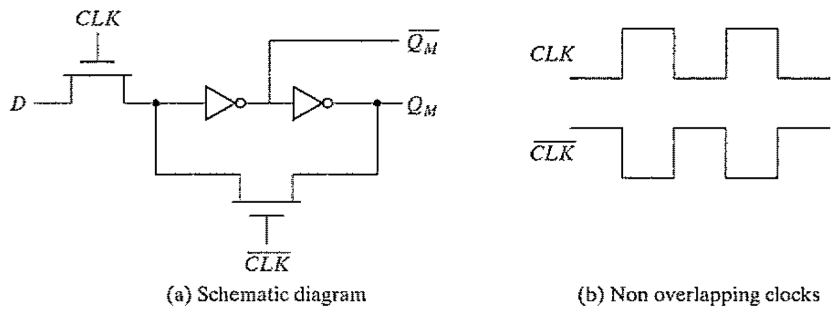


Figure 7-8 Multiplexer-based NMOS latch by using NMOS-only pass transistors for multiplexers.

tation is not very efficient from this perspective: It presents a load of four transistors to the *CLK* signal.

It is possible to reduce the clock load to two transistors by implementing multiplexers that use as NMOS-only pass transistors, as shown in Figure 7-8. When *CLK* is high, the latch samples the *D* input, while a low clock signal enables the feedback loop, and puts the latch in the hold mode. While attractive for its simplicity, the use of NMOS-only pass transistors results in the passing of a degraded high voltage of $V_{DD} - V_{Tn}$ to the input of the first inverter. This impacts both noise margin and the switching performance, especially in the case of low values of V_{DD} and high values of V_{Tn}. It also causes static power dissipation in the first inverter, because the maximum input voltage to the inverter equals $V_{DD} - V_{Tn}$, and the PMOS device of the inverter is never fully turned off.

7.2.3 Master–Slave Edge-Triggered Register

The most common approach for constructing an *edge-triggered* register is to use a *master–slave* configuration, as shown in Figure 7-9. The register consists of cascading a negative latch (master stage) with a positive one (slave stage). A multiplexer-based latch is used in this particular

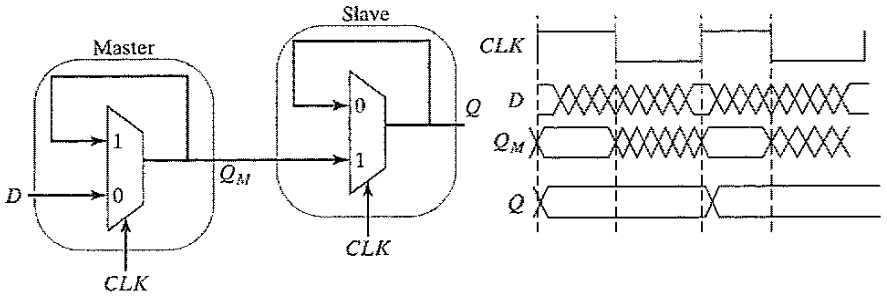


Figure 7-9 Positive edge-triggered register based on a master–slave configuration.

implementation, although any latch could be used. On the low phase of the clock, the master stage is transparent, and the D input is passed to the master stage output, Q_M. During this period, the slave stage is in the hold mode, keeping its previous value by using feedback. On the rising edge of the clock, the master stage stops sampling the input, and the slave stage starts sampling. During the high phase of the clock, the slave stage samples the output of the master stage (Q_M), while the master stage remains in a hold mode. Since Q_M is constant during the high phase of the clock, the output Q makes only one transition per cycle. The value of Q is the value of D right before the rising edge of the clock, achieving the *positive edge-triggered* effect. A negative edge-triggered register can be constructed by using the same principle by simply switching the order of the positive and negative latches (i.e., placing the positive latch first).

A complete transistor-level implementation of the master–slave positive edge-triggered register is shown in Figure 7-10. The multiplexer is implemented by using transmission gates as discussed in the previous section. When the clock is low ($\overline{CLK} = 1$), T_1 is on and T_2 is off, and the D input is sampled onto node Q_M. During this period, T_3 and T_4 are off and on, respectively. The cross-coupled inverters (I_5, I_6) hold the state of the slave latch. When the clock goes high, the master stage stops sampling the input and goes into a hold mode. T_1 is off and T_2 is on, and the cross-coupled inverters I_2 and I_3 hold the state of Q_M. Also, T_3 is *on* and T_4 is off, and Q_M is copied to the output Q.

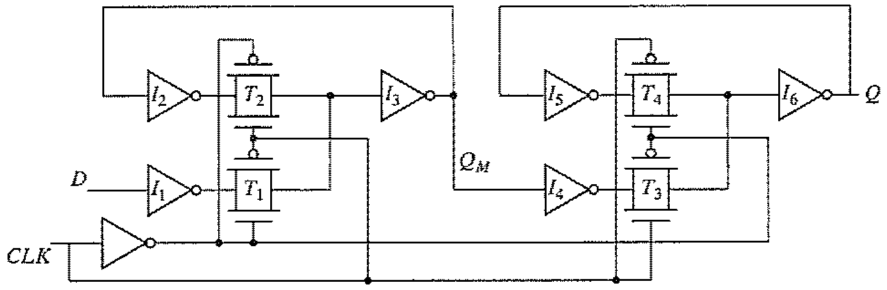


Figure 7-10 Master–slave positive edge-triggered register, using multiplexers.

Problem 7.1 Optimization of the Master–Slave Register

It is possible to remove the inverters I_1 and I_4 from Figure 7-10 without loss of functionality. Is there any advantage to including these inverters in the implementation?

Timing Properties of Multiplexer-Based Master–Slave Registers

Registers are characterized by three important timing parameters: the setup time, the hold time and the propagation delay. It is important to understand the factors that affect these timing parameters and develop the intuition to manually estimate them. Assume that the propagation delay of each inverter is t_{pd_inv}, and the propagation delay of the transmission gate is t_{pd_tx}. Also assume that the contamination delay is 0, and that inverter, deriving $\overline{CLK}$ from CLK, has a delay of 0 as well.

The setup time is the time before the rising edge of the clock that the input data D must be valid. This is similar to asking the question, how long before the rising edge of the clock must the D input be stable such that Q_M samples the value reliably? For the transmission gate multiplexer-based register, the input D has to propagate through I_1, T_1, I_3, and I_2 before the rising edge of the clock. This ensures that the node voltages on both terminals of the transmission gate T_2 are at the same value. Otherwise, it is possible for the cross-coupled pair I_2 and I_3 to settle to an incorrect value. The setup time is therefore equal to $3 \times t_{pd\_inv} + t_{pd\_tx}$.

The propagation delay is the time it takes for the value of Q_M to propagate to the output Q. Note that, since we included the delay of I_2 in the setup time, the output of I_4 is valid before the rising edge of the clock. Therefore, the delay t_{c-q} is simply the delay through T_3 and I_6 ($t_{c-q} = t_{pd\_tx} + t_{pd\_inv}$).

The *hold time* represents the time that the input must be held stable after the rising edge of the clock. In this case, the transmission gate T_1 turns off when the clock goes high. Since both the D input and the CLK pass through inverters before reaching T_1, any changes in the input after the clock goes high do not affect the output. Therefore, the hold time is 0.

Example 7.1 Timing Analysis, Using SPICE

To obtain the setup time of the register while using SPICE, we progressively skew the input with respect to the clock edge until the circuit fails. Figure 7-11 shows the setup-time simulation assuming a skew of 210 ps and 200 ps. For the 210 ps case, the correct value of input D is sampled (in this case, the Q output remains at the value of V_{DD}). For a skew of 200 ps, an incorrect value propagates to the output, as the Q output transitions to 0. Node Q_M starts to go high, and the output of I_2 (the input to transmission gate T_2) starts to fall. However, the clock is enabled before the two nodes across the transmission gate T_2 settle to the same value. This results in an incorrect value being written into the master latch. The setup time for this register is 210 ps.

In a similar fashion, the hold time can be simulated. The D-input edge is once again skewed relative to the clock signal until the circuit stops functioning. For this design, the

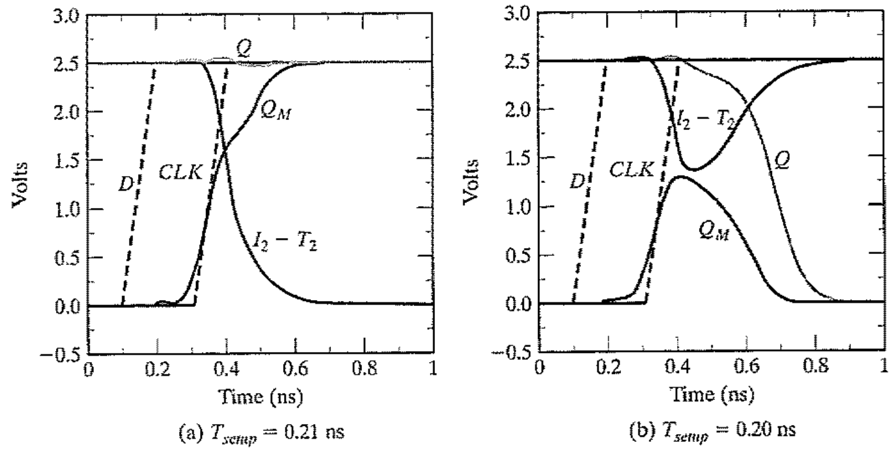


Figure 7-11 Setup time simulation.

hold time is 0 (i.e., the inputs can be changed on the clock edge). Finally, for the propagation delay, the inputs transition at least one setup time before the rising edge of the clock, and the delay is measured from the 50% point of the *CLK* edge to the 50% point of the *Q* output. From this simulation (Figure 7-12), $t_{c-q(lh)}$ was 160 ps, and $t_{c-q(hl)}$ was 180 ps.

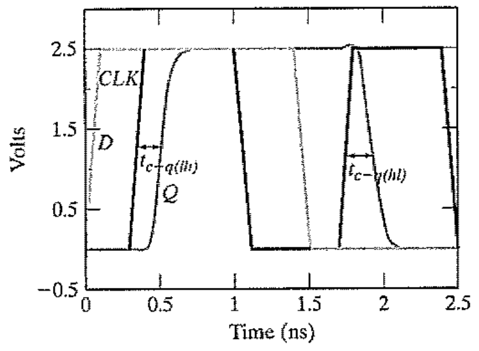


Figure 7-12 Simulation of propagation delay of transmission gate register.

The drawback of the transmission-gate register is the high capacitive load presented to the clock signal. The clock load per register is important, since it directly impacts the power dissipation of the clock network. Ignoring the overhead required to invert the clock signal—since the inverter overhead can be amortized over multiple register bits—each register has a clock load of eight transistors. One approach to reduce the clock load at the cost of robustness is to make the circuit

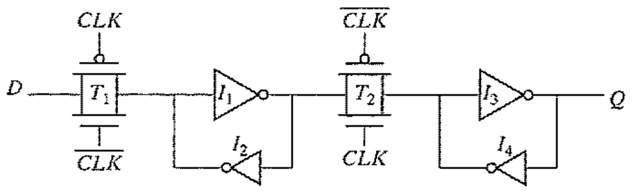


Figure 7-13 Reduced load clock load static master–slave register.

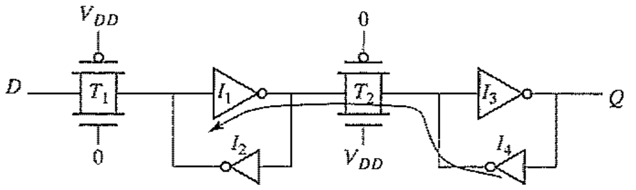


Figure 7-14 Reverse conduction possible in the transmission gate.

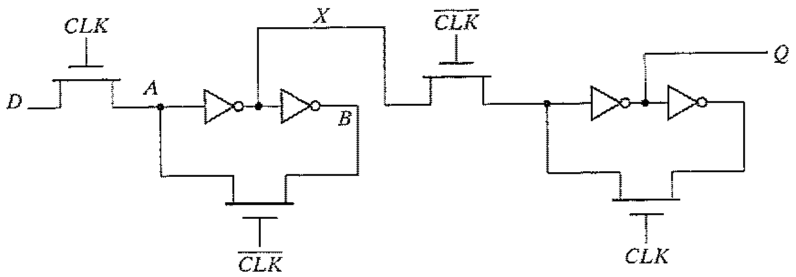
ratioed. Figure 7-13 shows that the feedback transmission gate can be eliminated by directly cross-coupling the inverters.

The penalty paid for the reducted in clock load is an increased design complexity. The transmission gate (T_1) and its source driver must overpower the feedback inverter (I_2) to switch the state of the cross-coupled inverter. The sizing requirements for the transmission gates can be derived by using an analysis similar to the one used for the sizing of the level-restoring device in Chapter 6. The input to the inverter I_1 must be brought below its switching threshold in order to make a transition. If minimum-sized devices are to be used in the transmission gates, it is essential that the transistors of inverter I_2 should be made even weaker. This can be accomplished by making their channel lengths larger than minimum. Using minimum or close-to-minimum size devices in the transmission gates is desirable to reduce the power dissipation in the latches and the clock distribution network.

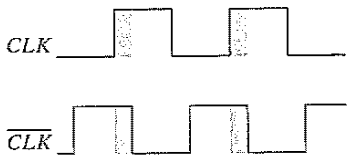
Another problem with this scheme is *reverse conduction*—the second stage can affect the state of the first latch. When the slave stage is on (Figure 7-14), it is possible for the combination of T_2 and I_4 to influence the data stored in the I_1–I_2 latch. As long as I_4 is a weak device, this fortunately not a major problem.

Non-Ideal Clock Signals

So far, we have assumed that $\overline{CLK}$ is a perfect inversion of *CLK*, or in other words, that the delay of the generating inverter is zero. Even if this were possible, this still would not be a good assumption. Variations can exist in the wires used to route the two clock signals, or the load capacitances can vary based on data stored in the connecting latches. This effect, known as *clock skew*, is a major problem, causing the two clock signals to overlap, as shown in Figure 7-15b. *Clock overlap* can cause two types of failures, which we illustrate for the NMOS-only negative master–slave register of Figure 7-15a.



(a) Schematic diagram

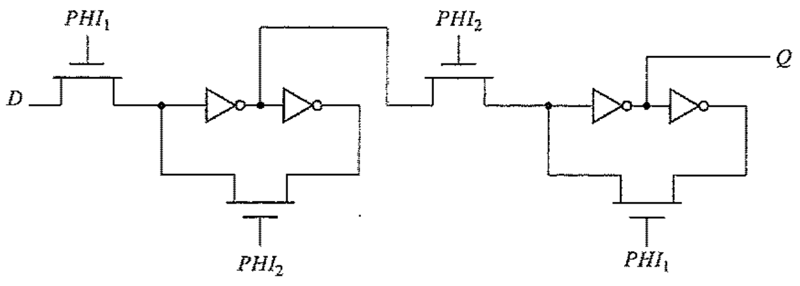


(b) Overlapping clock pairs

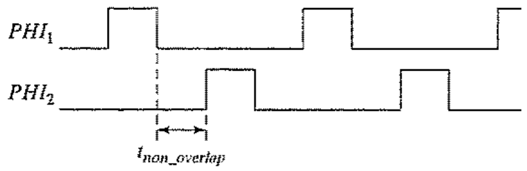
Figure 7-15 Master–slave register based on NMOS-only pass transistors.

1. When the clock goes high, the slave stage should stop sampling the master stage output and go into a hold mode. However, since *CLK* and $\overline{CLK}$ are both high for a short period of time (the *overlap period*), both sampling pass transistors conduct, and there is a direct path from the *D* input to the *Q* output. As a result, data at the output can change on the rising edge of the clock, which is undesired for a negative edge-triggered register. This is known as a *race* condition in which the value of the output *Q* is a function of whether the input *D* arrives at node *X* before or after the falling edge of $\overline{CLK}$. If node *X* is sampled in the metastable state, the output will switch to a value determined by noise in the system.
2. The primary advantage of the multiplexer-based register is that the feedback loop is open during the sampling period, and therefore the sizing of the devices is not critical to functionality. However, if there is clock overlap between *CLK* and $\overline{CLK}$, node *A* can be driven by both *D* and *B*, resulting in an undefined state.

These problems can be avoided by using two *nonoverlapping clocks* instead, PHI_1 and PHI_2 (Figure 7-16), and by keeping the nonoverlap time $t_{non\_overlap}$ between the clocks large enough so that no overlap occurs even in the presence of clock-routing delays. During the nonoverlap time, the *FF* is in the high-impedance state—the feedback loop is open, the loop gain is zero, and the input is disconnected. Leakage will destroy the state if this condition holds for too long—hence the name *pseudostatic*: The register employs a combination of static and dynamic storage approaches, depending upon the state of the clock.



(a) Schematic diagram



(b) Two-phase nonoverlapping clocks

Figure 7-16 Pseudostatic two-phase *D* register.

Problem 7.2 Generating Nonoverlapping Clocks

Figure 7-17 shows one possible implementation of the clock generation circuitry for generating a two-phase nonoverlapping clock. Assuming that each gate has a unit gate delay, derive the timing relationship between the input clock and the two output clocks. What is the nonoverlap period? How can this period be increased if needed?

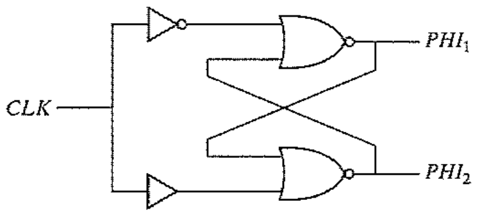


Figure 7-17 Circuitry for generating a two-phase nonoverlapping clock.

7.2.4 Low-Voltage Static Latches

The scaling of supply voltages is critical for low-power operation. Unfortunately, certain latch structures do not function at reduced supply voltages. For example, without the scaling of device thresholds, NMOS-only pass transistors (e.g., Figure 7-16) don't scale well with supply voltage due to its inherent threshold drop. At very low power supply voltages, the input to the inverter cannot be raised above the switching threshold, resulting in incorrect evaluation. Even with the use of transmission gates, performance degrades significantly at reduced supply voltages.

Scaling to low supply voltages thus requires the use of reduced threshold devices. However, this has the negative effect of exponentially increasing the subthreshold leakage power (as discussed in Chapter 6). When the registers are constantly accessed, the leakage energy typically is insignificant compared with the switching power. However, with the use of conditional clocks, it is possible that registers are idle for extended periods, and the leakage energy expended by registers can be quite significant.

Many solutions are being explored to address the problem of high leakage during idle periods. One approach involves the use of Multiple Threshold devices, as shown in Figure 7-18 [Mutoh95]. Only the negative latch is shown. The shaded inverters and transmission gates are implemented in low-threshold devices. The low-threshold inverters are gated by using high-threshold devices to eliminate leakage.

During the normal mode of operation, the sleep devices are turned on. When the clock is low, the *D* input is sampled and propagates to the output. The latch is in the hold mode when the clock is high. The feedback transmission gate conducts and the cross-coupled feedback is enabled. An extra inverter, in parallel with the low-threshold one, is added to store the state when the latch is in *idle* (or *sleep*) mode. Then, the high-threshold devices in series with the low-threshold inverter are turned off (the *SLEEP* signal is high), eliminating leakage. It is assumed that clock

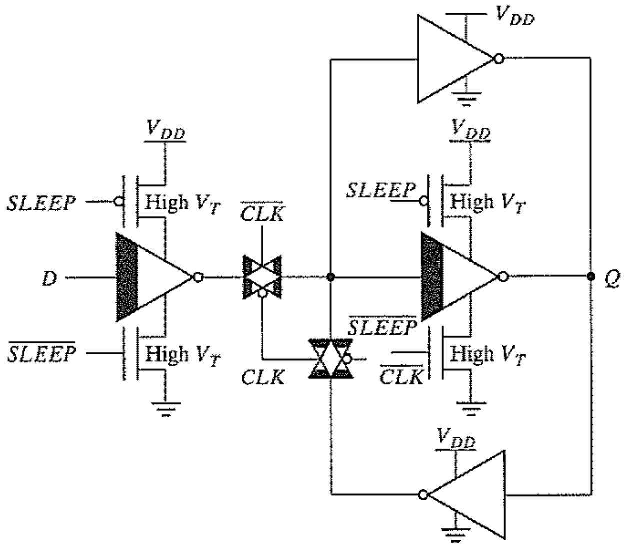


Figure 7-18 Solving the leakage problem, using multiple-threshold CMOS.

is held high when the latch is in the sleep state. The feedback low-threshold transmission gate is turned on and the cross-coupled high-threshold devices maintain the state of the latch.

Problem 7.3 Transistor Minimization in the MTCMOS Register

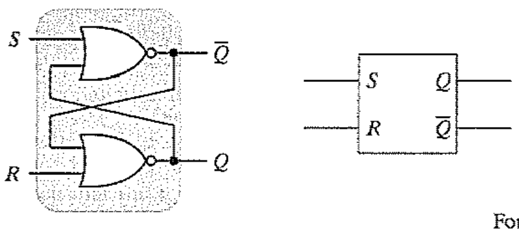
Unlike combinational logic, both NMOS and PMOS high-threshold devices are required to eliminate the leakage in low-threshold latches. Explain why this is the case.

Hint: Eliminate the high-V_T NMOS or high-V_T PMOS of the low-threshold inverter on the right of Figure 7-18, and investigate potential leakage paths.

7.2.5 Static SR Flip-Flops—Writing Data by Pure Force

The traditional way of causing a bistable element to change state is to overpower the feedback loop. The simplest incarnation accomplishing this is the well-known *SR*, or *set-reset, flip-flop*, an implementation of which is shown in Figure 7-19a. This circuit is similar to the cross-coupled inverter pair with NOR gates replacing the inverters. The second input of the NOR gates is connected to the trigger inputs (S and R) that make it possible to force the outputs Q and $\overline{Q}$ to a given state. These outputs are complimentary (except for the SR = 11 state). When both S and R are 0, the flip-flop is in a quiescent state and both outputs retain their values. (A NOR gate with one of its inputs being 0 looks like an inverter, and the structure looks like a cross-coupled inverter.) If a positive (or 1) pulse is applied to the S input, the Q output is forced into the 1 state (with $\overline{Q}$ going to 0) and vice versa: A 1-pulse on R resets the flip-flop, and the Q output goes to 0.

These results are summarized in the *characteristic table* of the flip-flop, shown in Figure 7-19c. The characteristic table is the truth table of the gate and lists the output states as functions of all possible input conditions. When both S and R are high, both Q and $\overline{Q}$ are forced to zero. Since this does not correspond with our constraint that Q and $\overline{Q}$ must be complementary, this input mode is considered forbidden. An additional problem with this condition is that when the input triggers return to their zero levels, the resulting state of the latch is unpredictable, and depends on whatever input is last to go low. Finally, Figure 7-19b shows the schematic symbol of the SR flip-flop.



S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
1	1	0	0

Forbidden State

(a) Schematic diagram (b) Logic symbol (c) Characteristic table

Figure 7-19 NOR-based *SR* flip-flop.

Problem 7.4 SR Flip-Flop, Using NAND Gates

An SR flip-flop can also be implemented by using a cross-coupled NAND structure, as shown in Figure 7-20. Derive the truth table for a such an implementation.

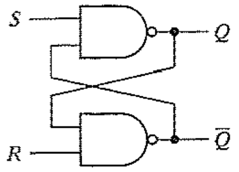


Figure 7-20 NAND-based SR flip-flop.

The SR flip-flop shown so far is purely asynchronous, which does not match well with the synchronous design methodology, the preferred strategy for more than 99% of today's integrated circuits. A clocked version of the latch is shown in Figure 7-21. It consists of a cross-coupled inverter pair, plus four extra transistors to drive the flip-flop from one state to another, and to provide synchronization. In steady state, one inverter resides in the high state, while the other one is low. No static paths between V_{DD} and GND exist. Transistor sizing is, however, essential to ensure that the flip-flop can transition from one state to the other when requested.

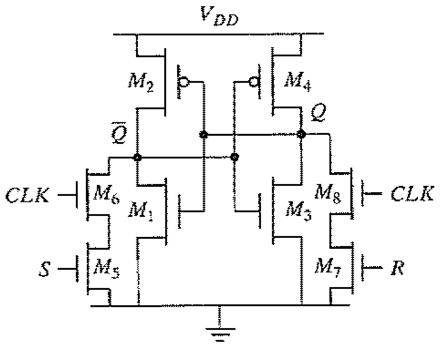


Figure 7-21 Ratioed CMOS *SR* latch.

Example 7.2 Transistor Sizing of Clocked SR Latch

Consider the case in which Q is high and an R pulse is applied. In order to make the latch switch, we must succeed in bringing Q below the switching threshold of the inverter M_1–M_2. Once this is achieved, the positive feedback causes the flip-flop to invert states. This requirement forces us to increase the sizes of transistors M_5, M_6, M_7, and M_8. The combination of transistors M_4, M_7, and M_8 forms a ratioed inverter. Assume that the cross-coupled inverter pair is designed such that the inverter threshold V_M is located at $V_{DD}/2$. For a 0.25-μm CMOS technology, the following transistor sizes were selected: $(W/L)_{M_1} = (W/L)_{M_3} = (0.5\ \mu\text{m}/0.25\ \mu\text{m})$, and $(W/L)_{M_2} = (W/L)_{M_4} = (1.5\ \mu\text{m}/0.25\ \mu\text{m})$. Assuming $Q = 0$, we determine the minimum sizes of M_5, M_6, M_7, and M_8 to make the device switchable.

To switch the latch from the $Q = 0$ to the $Q = 1$ state, it is essential that the low level of the ratioed, pseudo-NMOS inverter $(M_5–M_6)–M_2$ be below the switching threshold of the inverter $M_3–M_4$ that equals $V_{DD}/2$. It is reasonable to assume that as long as $V_{\overline{Q}} > V_M$, V_Q equals 0 and the gate of transistor M_2 is grounded. The boundary conditions on the transistor sizes can be derived by equating the currents in the inverter for $V_{\overline{Q}} = V_{DD}/2$, as given in Eq. (7.4) (this ignores channel-length modulation). The currents are determined by the saturation current, since $V_S = V_{DD} = 2.5$ V and $V_M = 1.25$ V. We assume that M_5 and M_6 have identical sizes and that W/L_{5-6} is the effective ratio of the series-connected devices. Under this condition, the pull-down network can be modeled by a single transistor M_{5-6}, whose length is twice the length of the individual devices:

$$k'_n\left(\frac{W}{L}\right)_{5-6}\left((V_{DD}-V_{Tn})V_{DSATn}-\frac{V_{DSATn}^2}{2}\right)$$
$$= -k'_p\left(\frac{W}{L}\right)_2\left((-V_{DD}-V_{Tp})V_{DSATp}-\frac{V_{DSATp}^2}{2}\right) \tag{7.4}$$

Using the parameters for the 0.25-μm process, Eq. (7.4) results in the constraint that the effective $(W/L)_{M_{5-6}} \geq 2.26$. This implies that the individual device ratio for M_5 or M_6 must be larger than approximately 4.5. Figure 7-22a shows the DC plot of $V_{\overline{Q}}$ as a function of the individual device sizes of M_5 and M_6. We notice that the individual device ratio of greater than 3 is sufficient to bring the $\overline{Q}$ voltage to the inverter switching threshold. The difference between the manual analysis and simulation arises from second-order effects

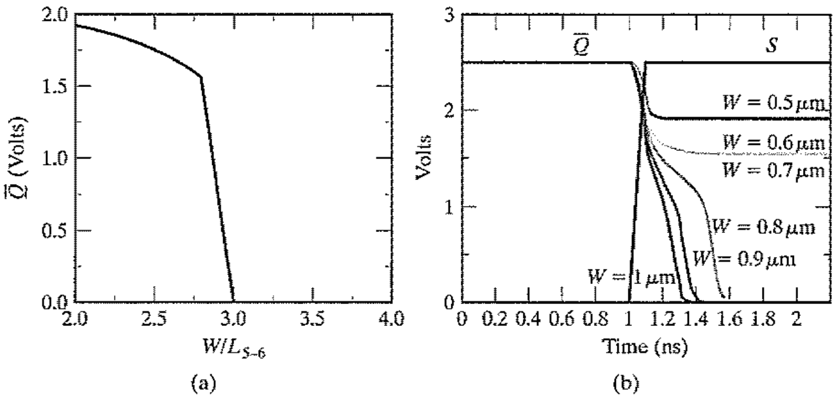


Figure 7-22 Sizing issues for *SR* flip-flop. (a) DC output voltage versus pull-down device size M_{5-6} (with $W/L_2 = 1.5\ \mu\text{m}/0.25\ \mu\text{m}$). (b) Transient response showing that M_5 and M_6 must each have a W/L larger than 3 to switch the *SR* flip-flop.