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.

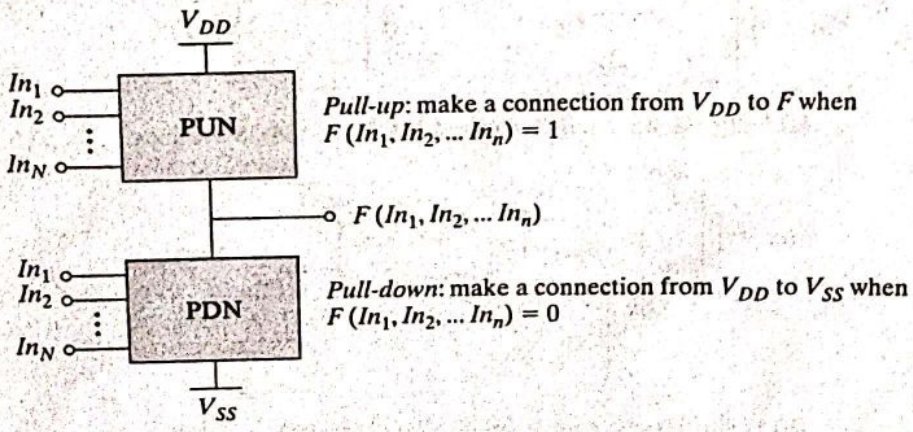
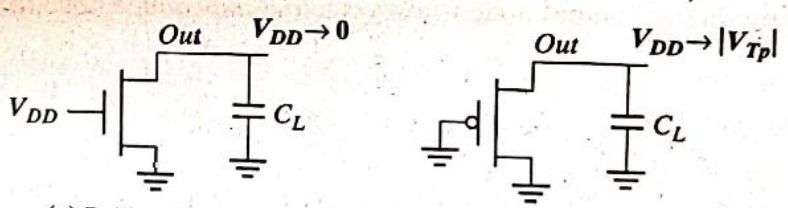


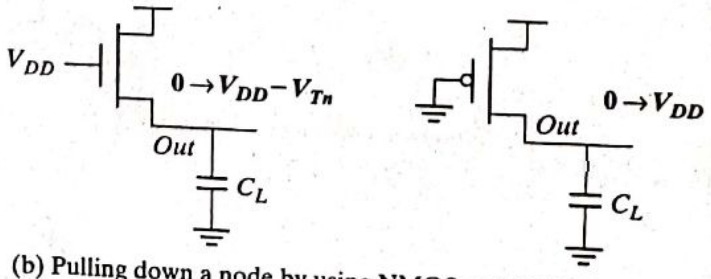
Figure 6-2 Complementary logic gate as a combination of a PUN (pull-up network) and a PDN (pull-down network).

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.

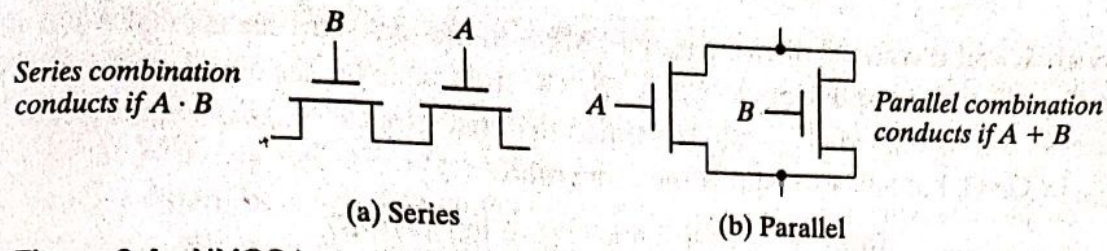


Figure 6-4 NMOS logic rules—series devices implement an AND, and parallel devices implement an OR.

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

- 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

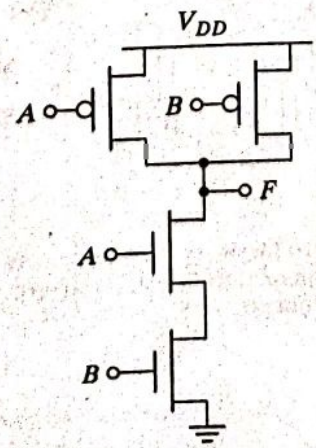


Figure 6-5 Two-input NAND gate in complementary static CMOS style.

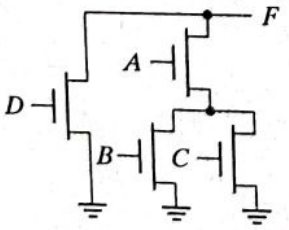
network, and it consists of two parallel PMOS transistors. This means that F is 1 if $A = 0$ or $B = 0$, which is equivalent to $F = \overline{A \cdot B}$. The truth table for the simple two input NAND gate is given in Table 6-1. It can be verified that the output F is always connected to either V_{DD}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

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

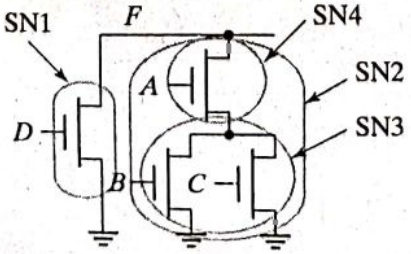
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Example 6.2 Synthesis of Complex CMOS Gate

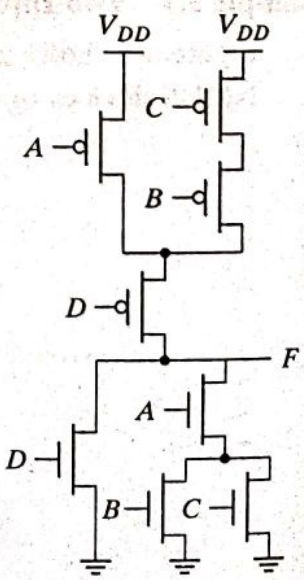
Using complementary CMOS logic, consider the synthesis of a complex CMOS gate whose function is $F = \overline{D + A \cdot (B + C)}$. The first step in the synthesis of the logic gate is to derive the pull-down network as shown in Figure 6-6a by using the fact that NMOS devices in series implements the 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-



(a) Pull-down network

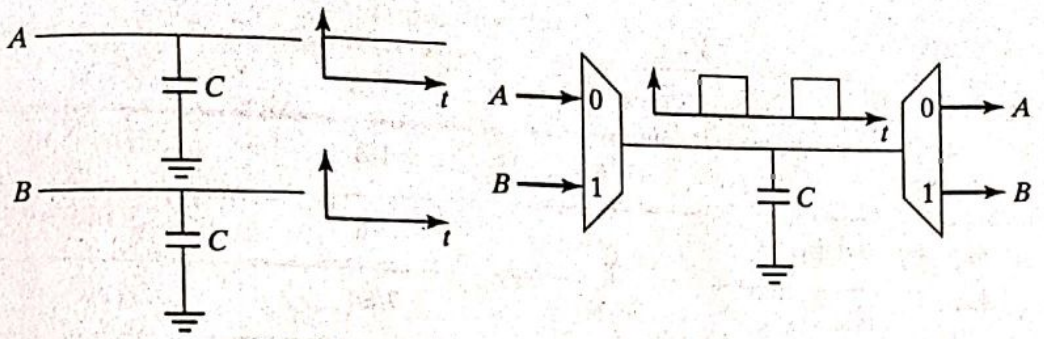


(b) Deriving the pull-up network hierarchically by identifying subnets



(c) Complete gate

Figure 6-6 Complex complementary CMOS gate.

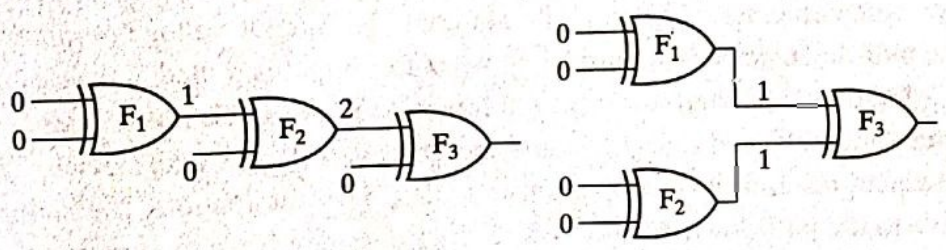


(a) Parallel data transmission (b) Serial data transmission

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.

■

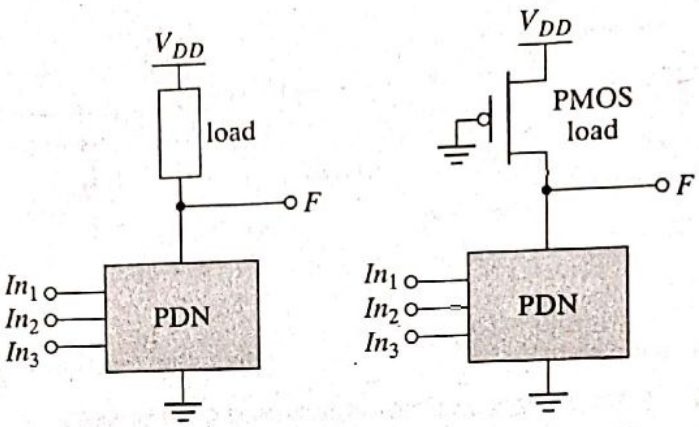
Summary

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



(a) Generic (b) Pseudo-NMOS

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}.

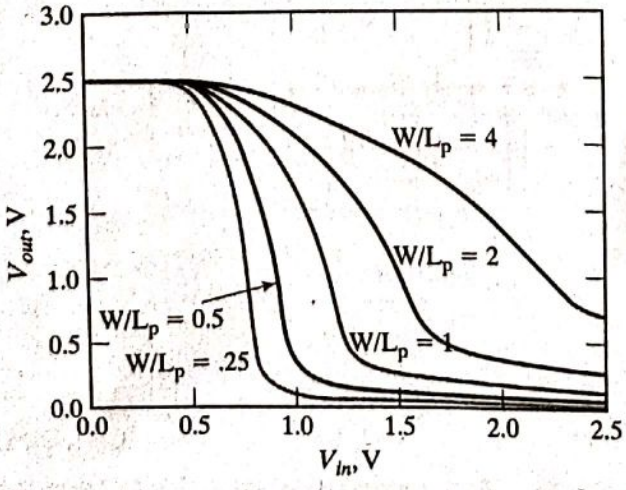


Figure 6-28 Voltage-transfer curves of the pseudo-NMOS inverter as a function of the PMOS size.

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.

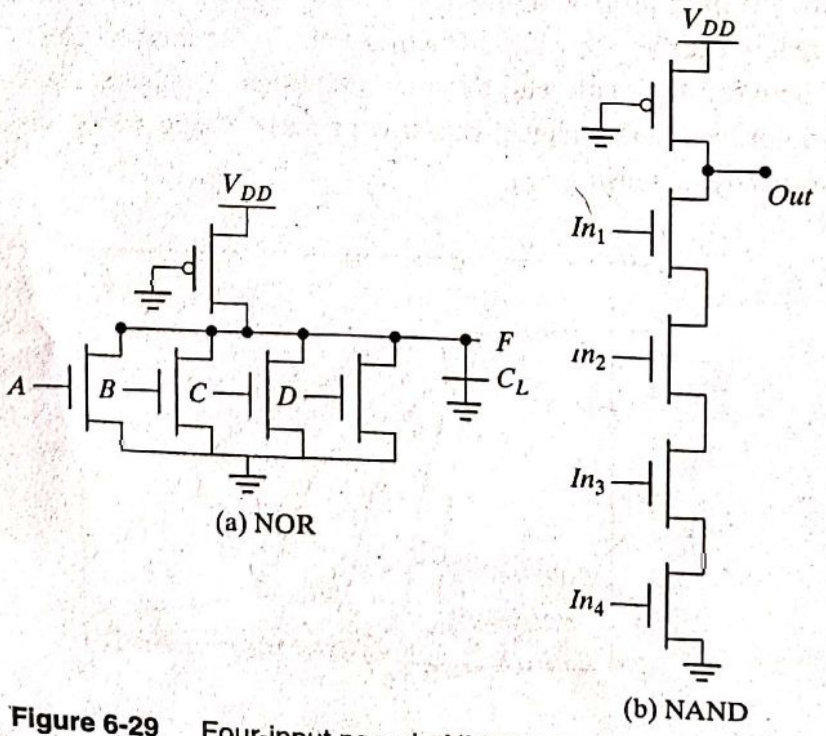


Figure 6-29 Four-input pseudo-NMOS NOR and NAND gates.

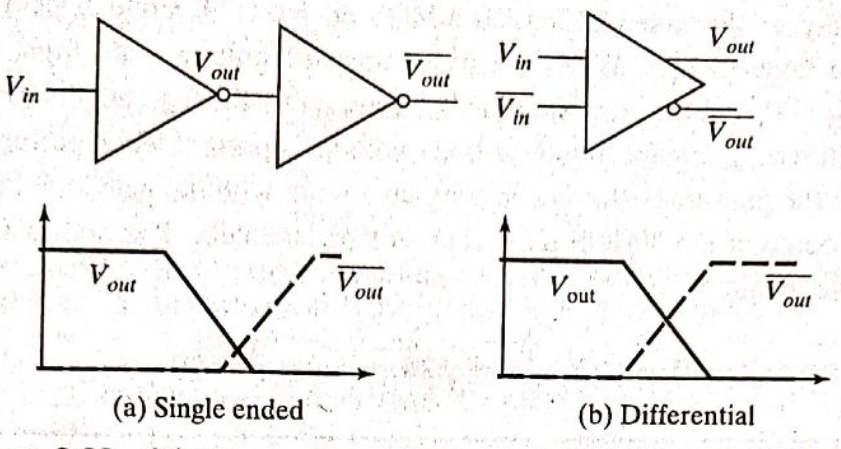


Figure 6-32 Advantage of over single-ended (a) differential (b) gate. ■

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.

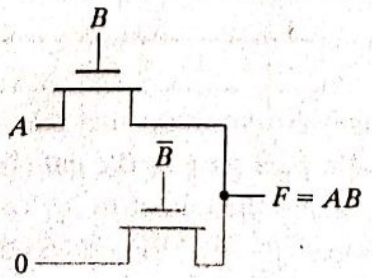


Figure 6-33 Pass-transistor implementation of an AND gate.

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

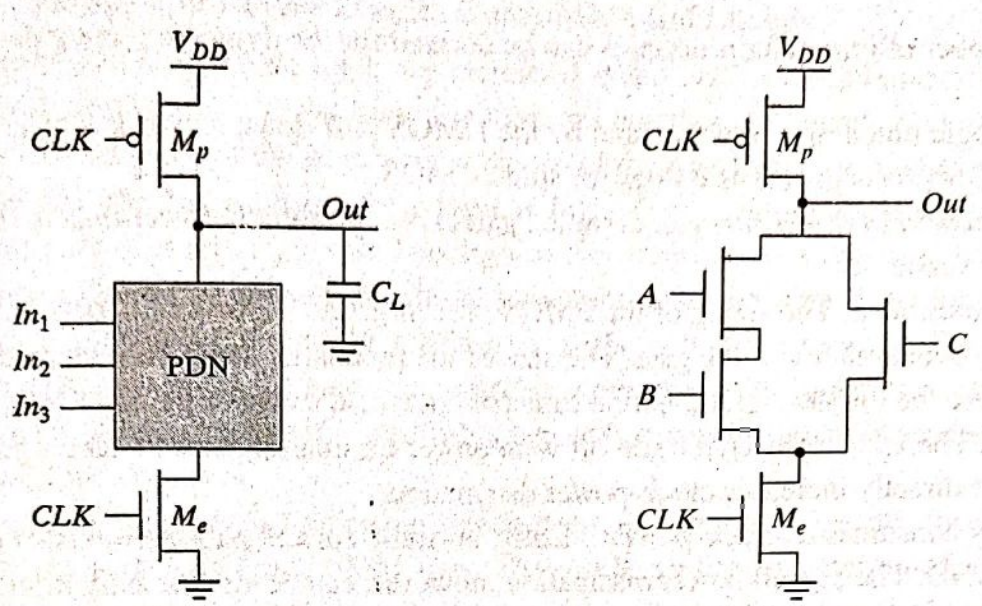
It was noted earlier that static CMOS logic with a fan-in of N requires $2N$ devices. A variety of approaches were presented to reduce the number of transistors required to implement a given logic function including pseudo-NMOS, pass-transistor logic, etc. The pseudo-NMOS logic style requires only $N + 1$ 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



(a) *n*-type network (b) Example

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:

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

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

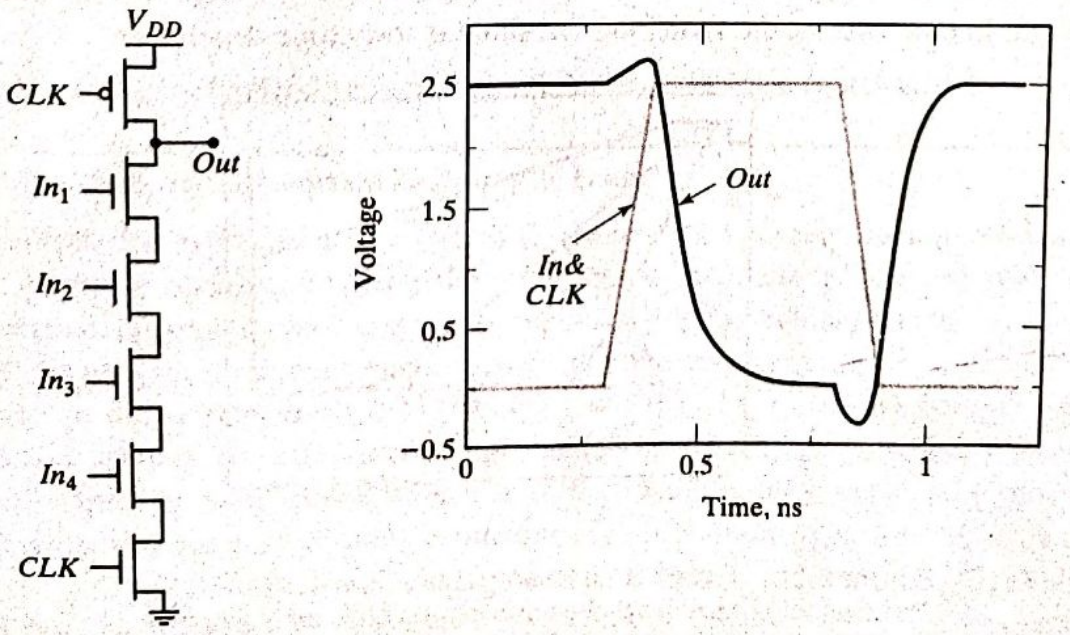


Figure 6-53 Schematic and transient response of a four-input dynamic NAND gate.

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.

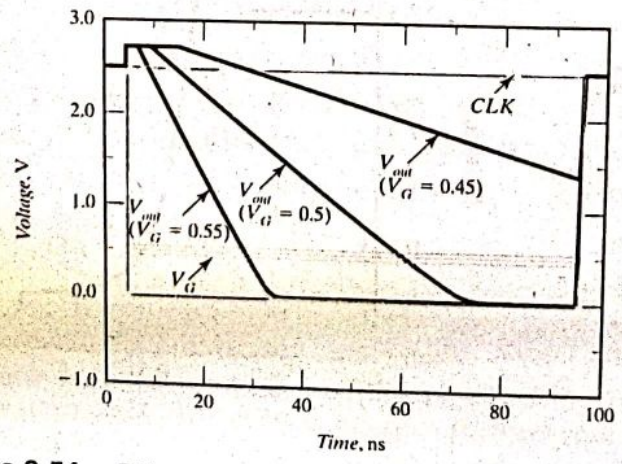


Figure 6-54 Effect of an input glitch on the output. The switching threshold depends on the time for evaluation. A larger glitch is acceptable if the evaluation phase is shorter.

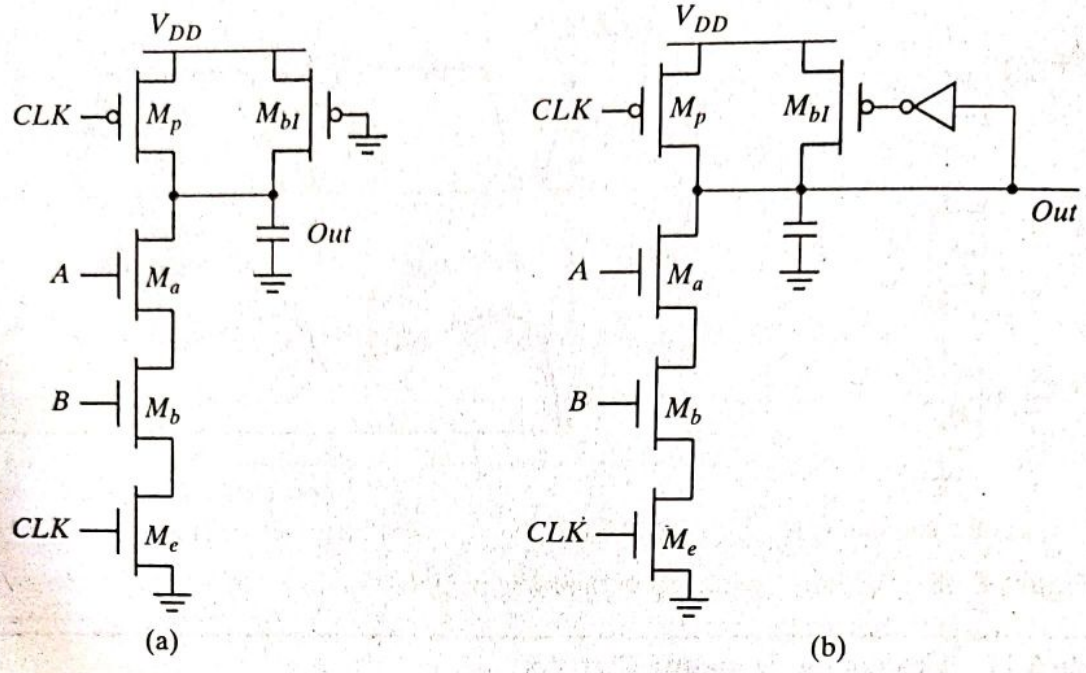


Figure 6-58 Static bleeders compensate for the charge leakage.

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

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

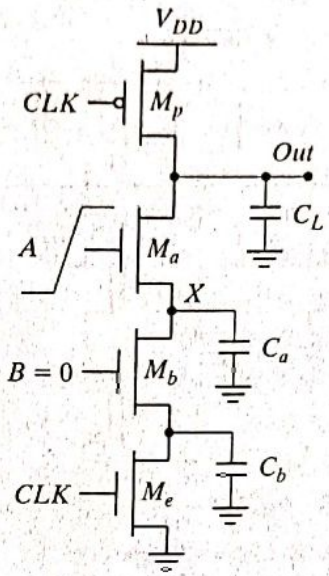


Figure 6-59 Charge sharing in dynamic networks.

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

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

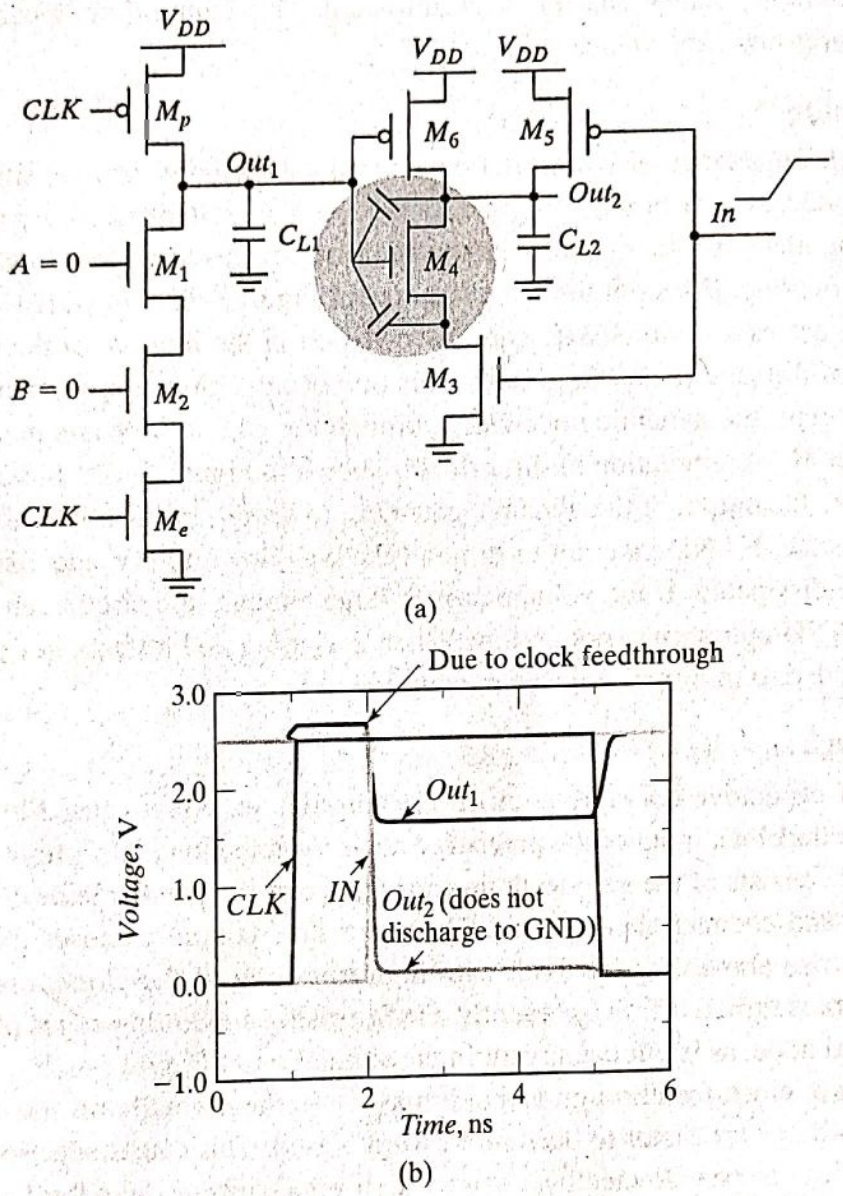


Figure 6-62 Example demonstrating the effect of backgate coupling: (a) circuit schematics; (b) simulation results.

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 $0 \rightarrow 1$ 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.

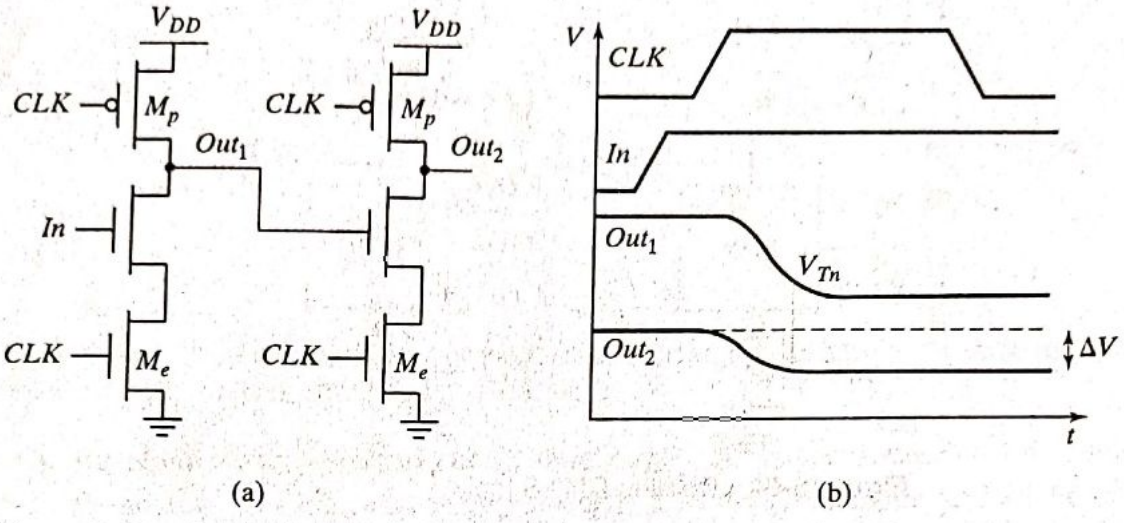


Figure 6-63 Cascade of dynamic *n*-type blocks.

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.

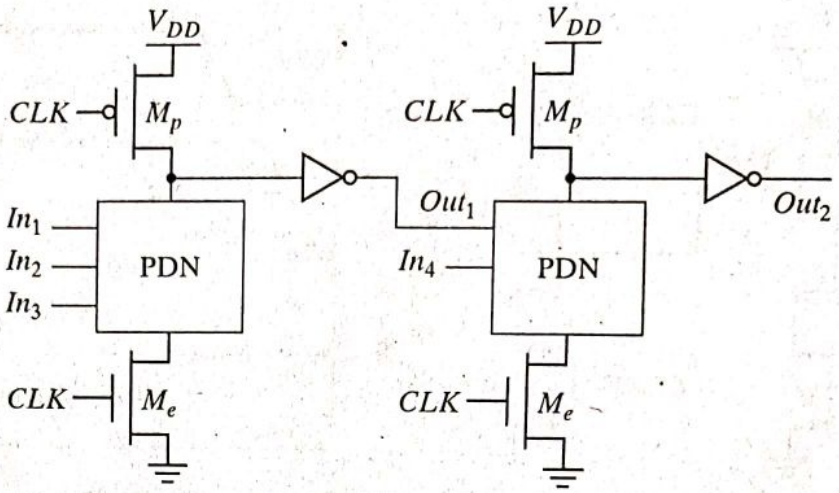


Figure 6-64 Domino CMOS logic.

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.

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

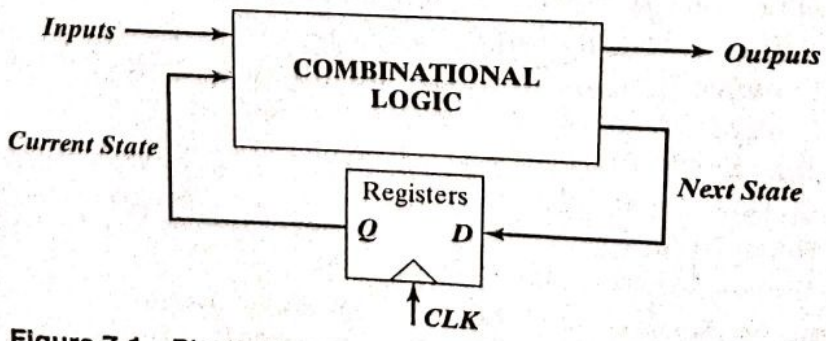


Figure 7-1 Block diagram of a finite-state machine, using *positive edge-triggered* registers.

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

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

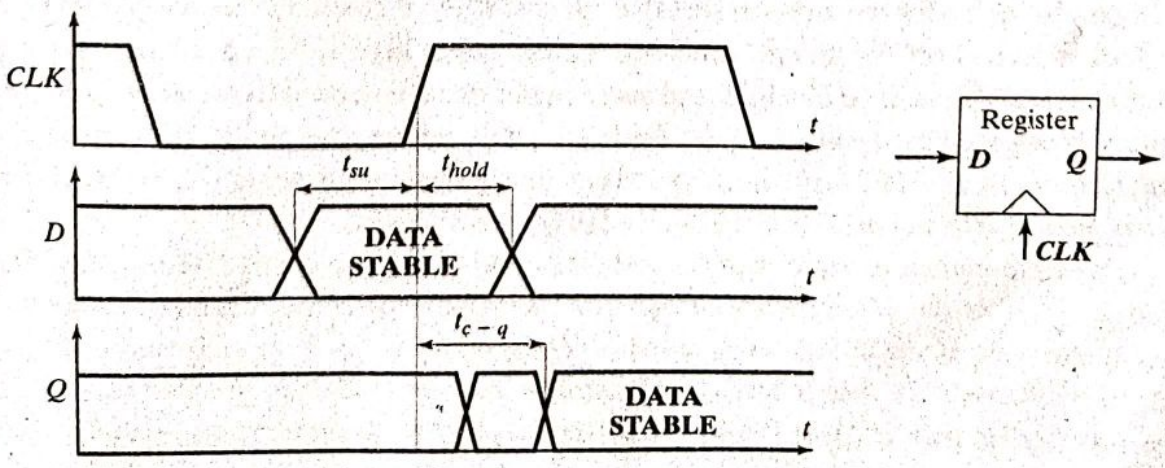


Figure 7-2 Definition of *setup time*, *hold time*, and *propagation delay* of a synchronous register.

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:

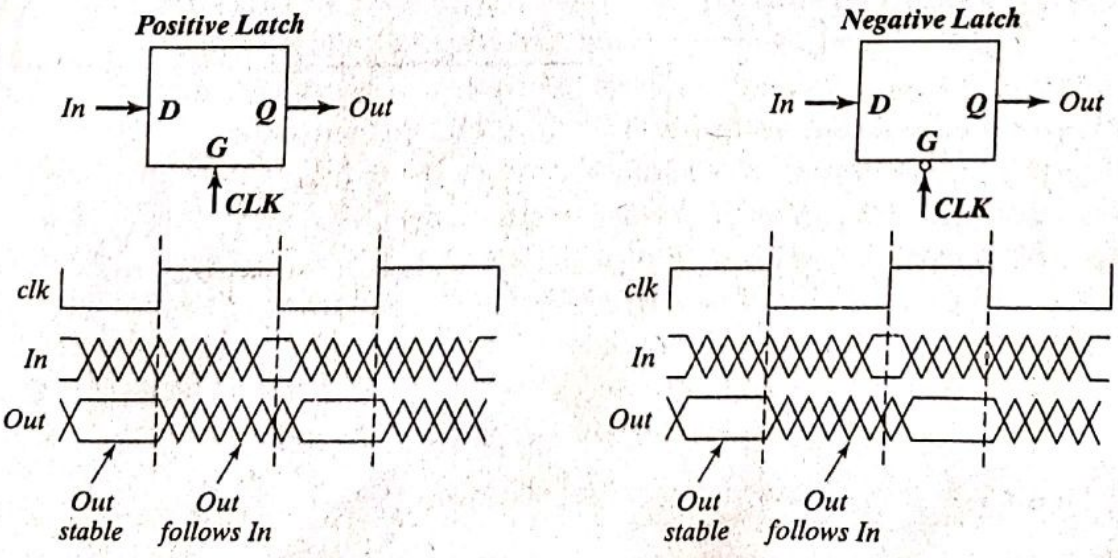


Figure 7-3 Timing of positive and negative latches.

- 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.

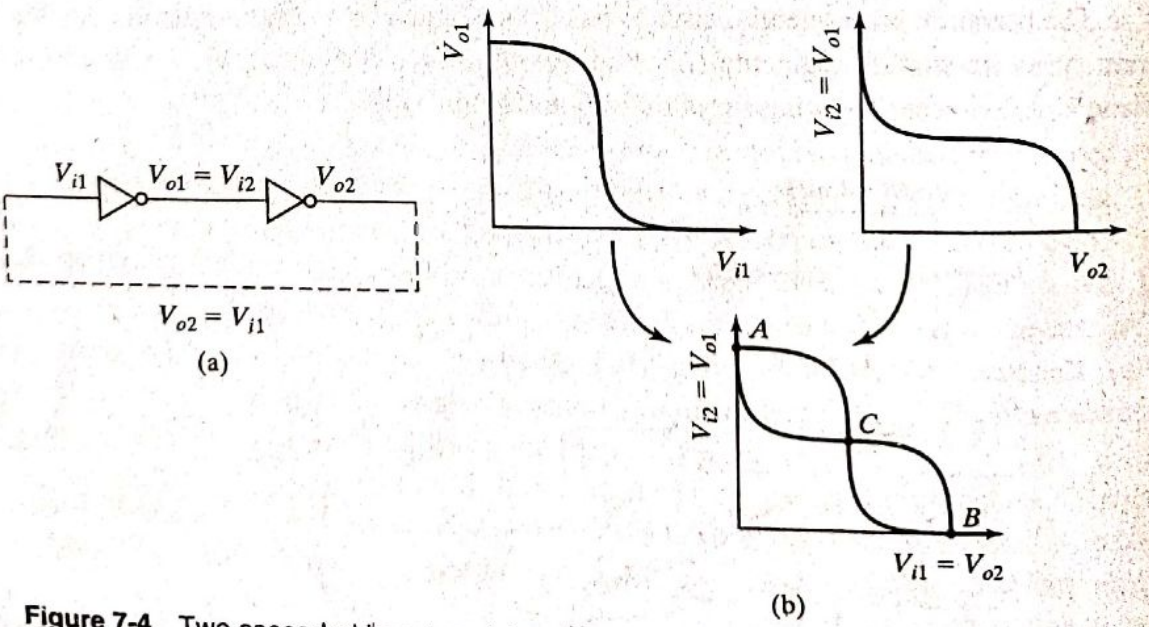


Figure 7-4 Two cascaded inverters (a) and their VTCs (b).

[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.

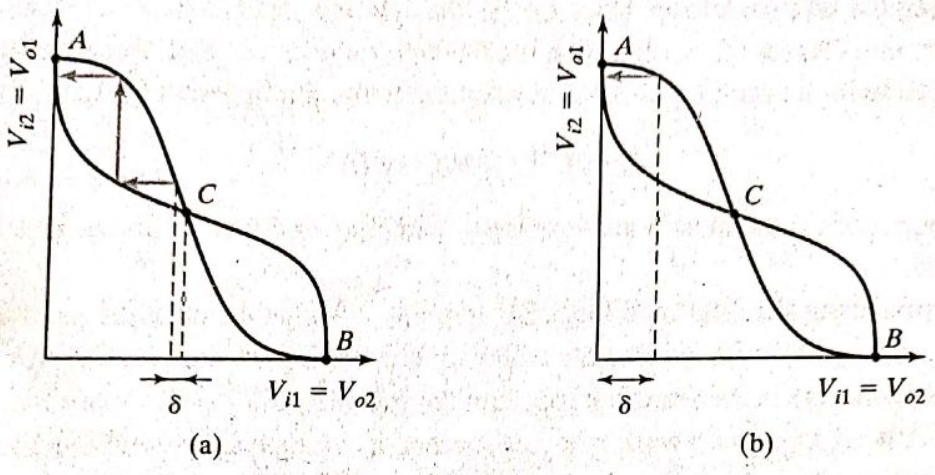


Figure 7-5 Metastable versus stable operation points.

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-

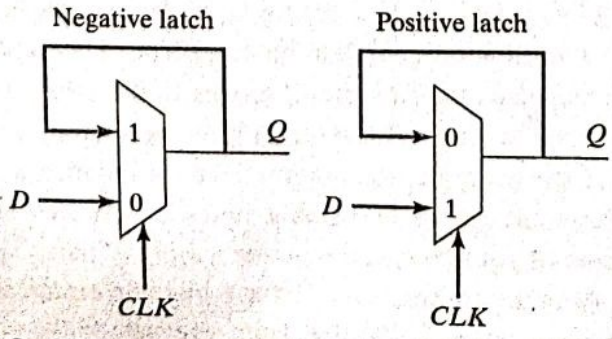


Figure 7-6 Negative and positive latches based on multiplexers.

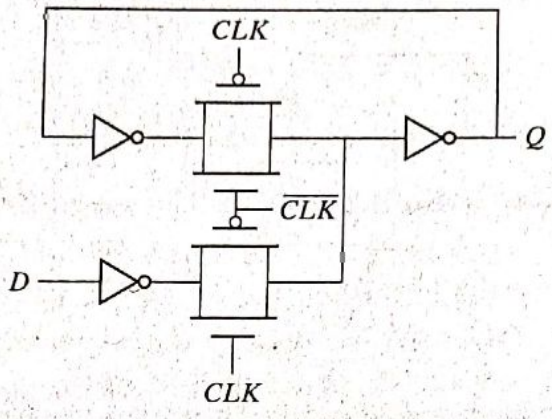
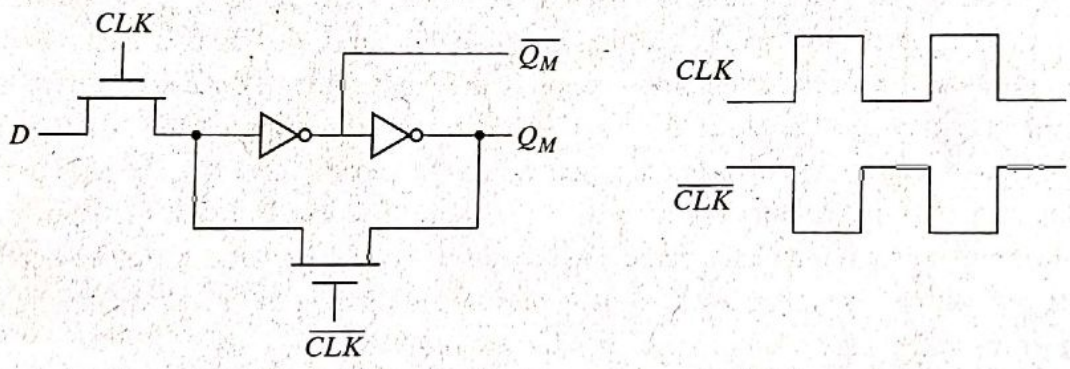


Figure 7-7 Transistor-level implementation of a positive latch built by using transmission gates.



(a) Schematic diagram (b) Non overlapping clocks

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

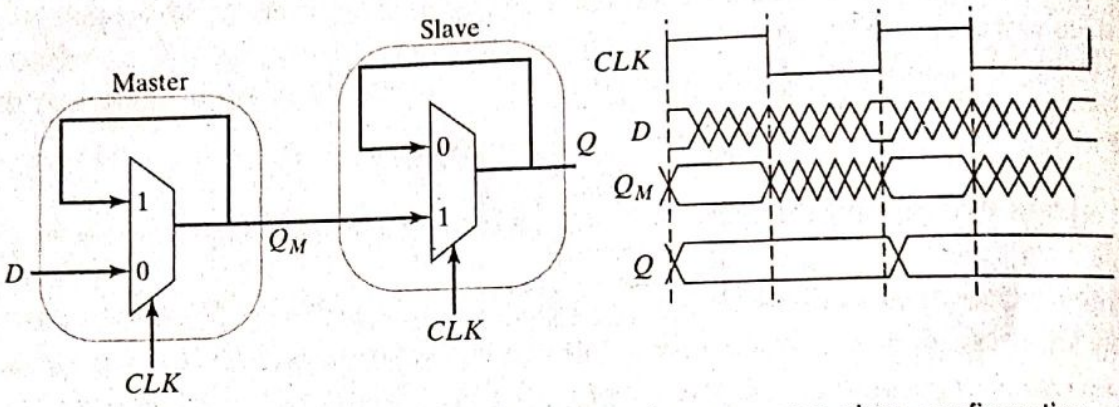


Figure 7-9 Positive edge-triggered register based on a master–slave configuration.

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.

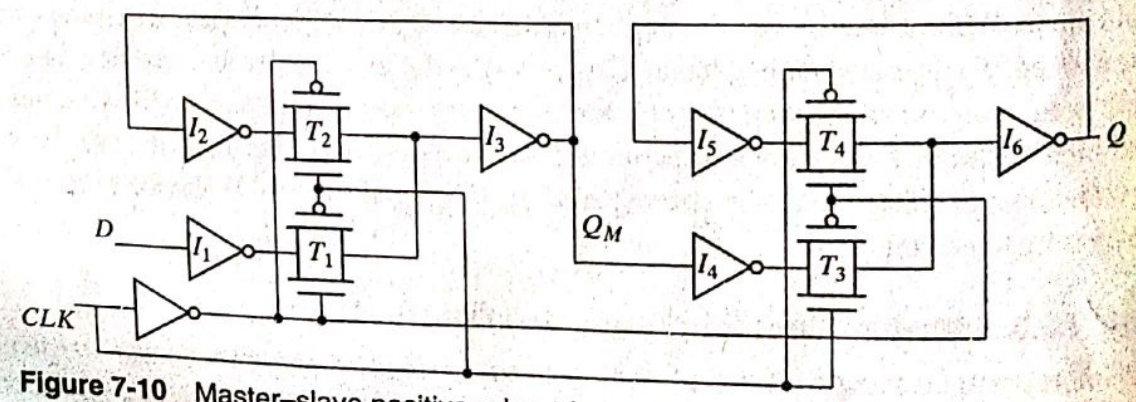


Figure 7-10 Master–slave positive edge-triggered register, using multiplexers.

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

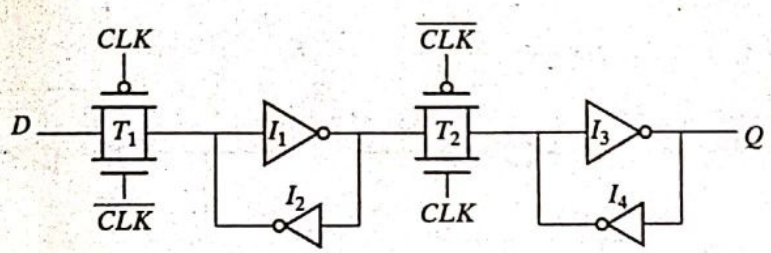


Figure 7-13 Reduced load clock load static master–slave register.

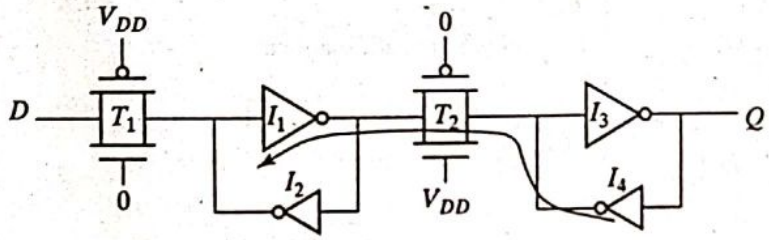


Figure 7-14 Reverse conduction possible in the transmission gate.

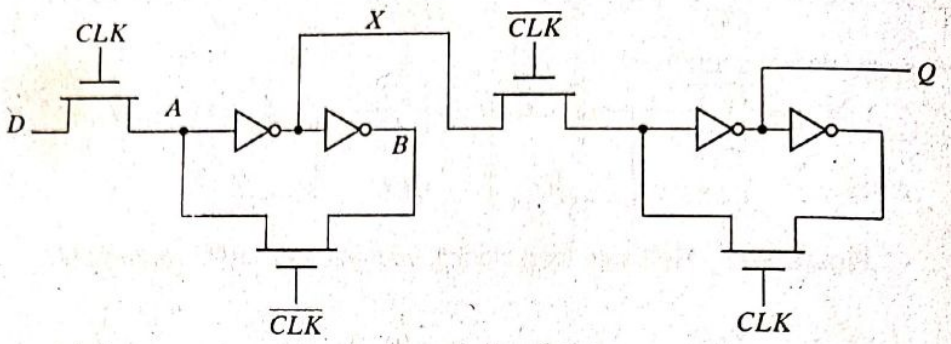
ratioed. Figure 7-13 shows that the feedback transmission gate can be eliminated by directly cross-coupling the inverters.

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.

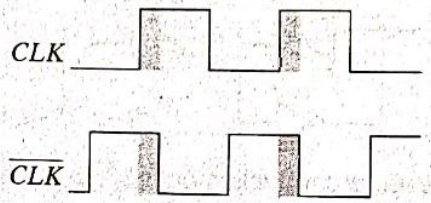
Another problem with this scheme is *reverse conduction*—the second stage can affect the state of the first latch. When the slave stage is on (Figure 7-14), it is possible for the combination of T_2 and I_4 to influence the data stored in the I_1–I_2 latch. As long as I_4 is a weak device, this fortunately 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

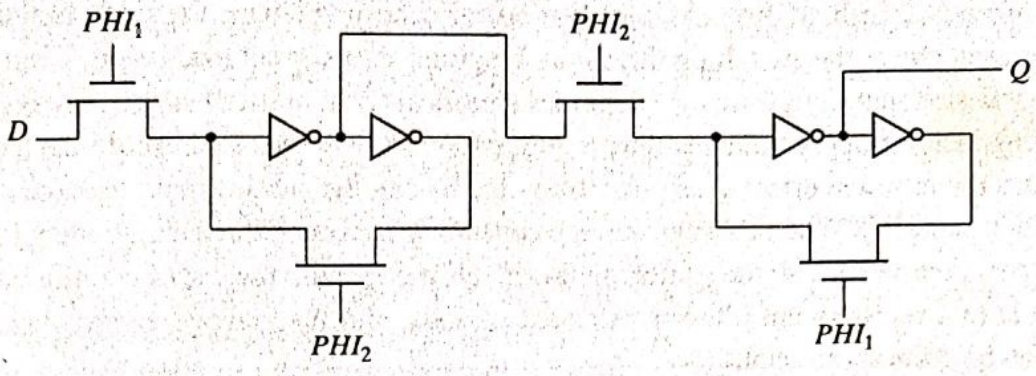


(b) Overlapping clock pairs

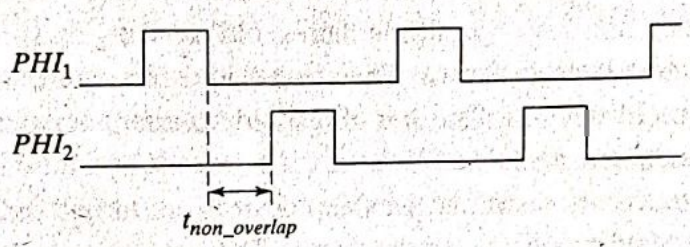
Figure 7-15 Master–slave register based on NMOS-only pass transistors.

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?

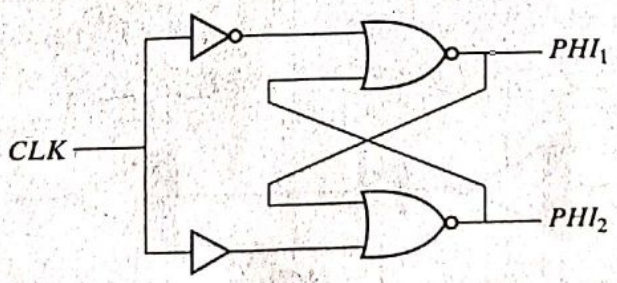


Figure 7-17 Circuitry for generating a two-phase nonoverlapping clock.

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

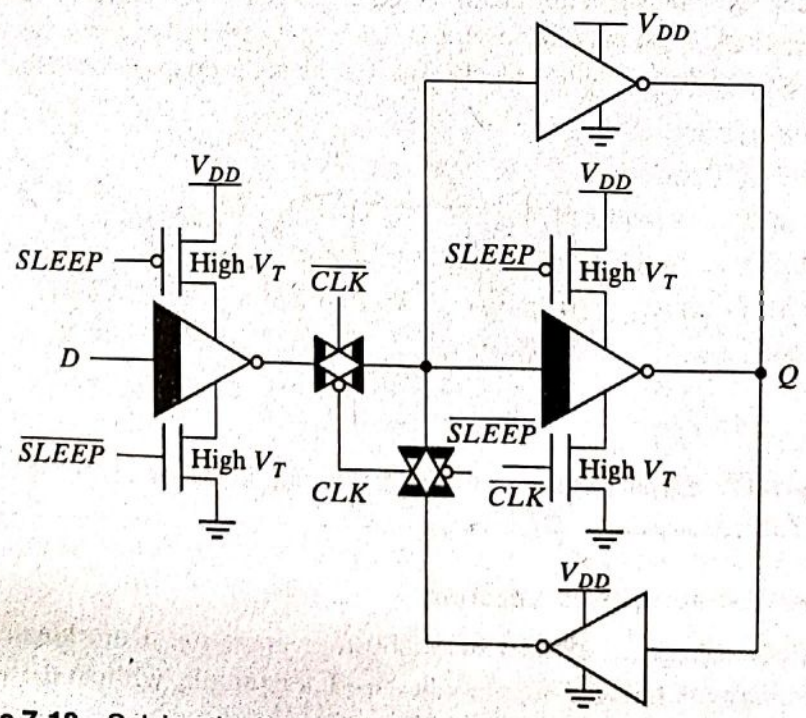


Figure 7-18 Solving the leakage problem, using multiple-threshold CMOS.

such as channel length modulation and DIBL. Figure 7-22b plots the transient response for different device sizes and confirms that an individual W/L ratio of greater than 3 is required to overpower the feedback and switch the state of the latch.

7.3 Dynamic Latches and Registers

Storage in a static sequential circuit relies on the concept that a cross-coupled inverter pair produces a bistable element and can thus be used to memorize binary values. This approach has the useful property that a stored value remains valid as long as the supply voltage is applied to the circuit—hence the name *static*. The major disadvantage of the static gate, however, is its complexity. When registers are used in computational structures that are constantly clocked (such as a pipelined datapath), the requirement that the memory should hold state for extended periods of time can be significantly relaxed.

This results in a class of circuits based on temporary storage of charge on parasitic capacitors. The principle is exactly identical to the one used in dynamic logic—charge stored on a capacitor can be used to represent a logic signal. The absence of charge denotes a 0, while its presence stands for a stored 1. No capacitor is ideal, unfortunately, and some charge leakage is always present. A stored value can thus only be kept for a limited amount of time, typically in the range of milliseconds. If one wants to preserve signal integrity, a periodic *refresh* of the value is necessary; hence, the name *dynamic* storage. Reading the value of the stored signal from a capacitor without disrupting the charge requires the availability of a device with a high-input impedance.

7.3.1 Dynamic Transmission-Gate Edge-Triggered Registers

A fully dynamic positive edge-triggered register based on the master–slave concept is shown in Figure 7-23. When $CLK = 0$, the input data is sampled on storage node 1, which has an equivalent capacitance of C_1, consisting of the gate capacitance of I_1, the junction capacitance of T_1, and the overlap gate capacitance of T_1. During this period, the slave stage is in a hold mode, with node 2 in a high-impedance (floating) state. On the rising edge of clock, the transmission gate T_2 turns on, and the value sampled on node 1 right before the rising edge propagates to the output Q (note that node 1 is stable during the high phase of the clock, since the first transmission gate is

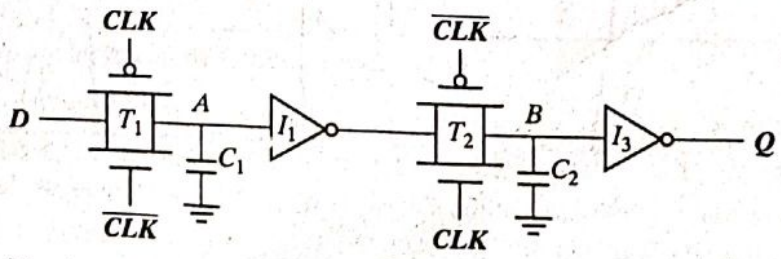


Figure 7-23 Dynamic edge-triggered register.

WARNING: The dynamic circuits shown in this section are very appealing from the perspective of complexity, performance, and power. Unfortunately, robustness considerations limit their use. In a fully dynamic circuit like that shown in Figure 7-23, a signal net that is capacitively coupled to the internal storage node can inject significant noise and destroy the state. This is especially important in ASIC flows, where there is little control over coupling between signal nets and internal dynamic nodes. Leakage currents cause another problem: Most modern processors require that the clock can be slowed down or completely halted, to conserve power in low-activity periods. Finally, the internal dynamic nodes do not track variations in power supply voltage. For example, when *CLK* is high for the circuit in Figure 7-23, node *A* holds its state, but it does not track variations in the power supply seen by I_1. This results in reduced noise margins.

Most of these problems can be adequately addressed by adding a weak feedback inverter and making the circuit *pseudostatic* (Figure 7-25). While this comes at a slight cost in delay, it improves the noise immunity significantly. Unless registers are used in a highly-controlled environment (for instance, a custom-designed high-performance datapath), they should be made pseudostatic or static. This holds for all latches and registers discussed in this section.

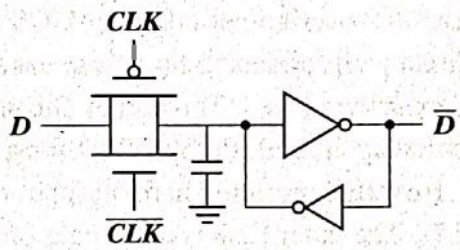


Figure 7-25 Making a dynamic latch pseudostatic.

7.3.2 C²MOS—A Clock-Skew Insensitive Approach

The C²MOS Register

Figure 7-26 shows an ingenious positive edge-triggered register that is based on a master–slave concept insensitive to clock overlap. This circuit is called the *C²MOS* (Clocked CMOS) *register* [Suzuki73], and operates in two phases:

1. $CLK = 0$ ($\overline{CLK} = 1$): The first tristate driver is turned on, and the master stage acts as an inverter sampling the inverted version of *D* on the internal node *X*. The master stage is in the evaluation mode. Meanwhile, the slave section is in a high-impedance mode, or in a hold mode. Both transistors M_7 and M_8 are off, decoupling the output from the input. The output *Q* retains its previous value stored on the output capacitor C_{L2}.
2. The roles are reversed when $CLK = 1$: The master stage section is in hold mode (M_3-M_4 off), while the second section evaluates (M_7–M_8 on). The value stored on C_{L1} propagates to the output node through the slave stage, which acts as an inverter.

The overall circuit operates as a positive edge-triggered master–slave register very similar to the transmission-gate-based register presented earlier. However, there is an important difference:

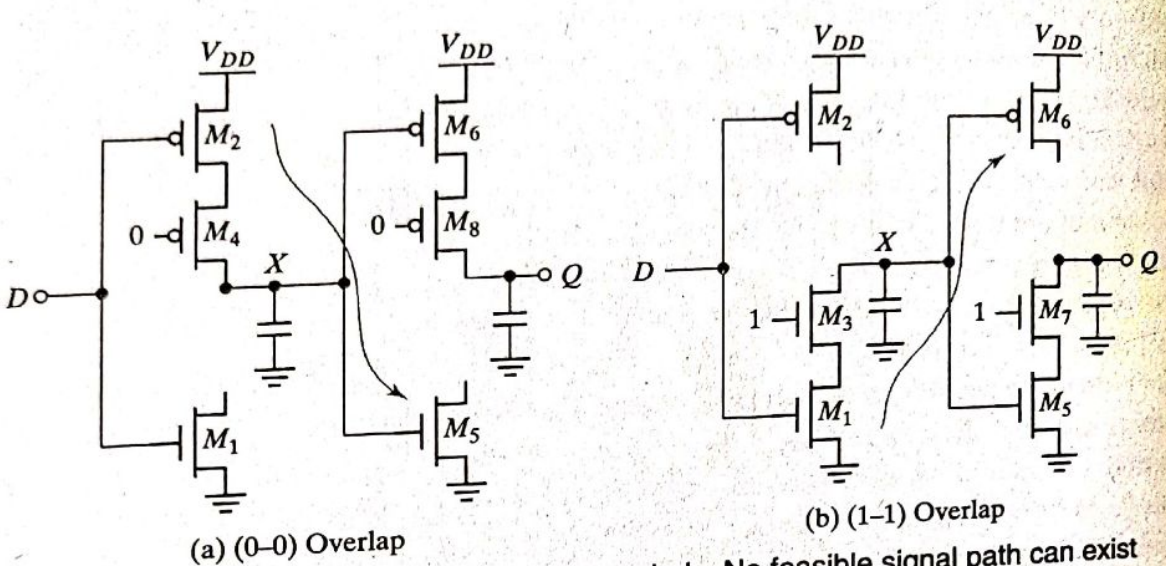


Figure 7-27 C^2MOS *D* FF during overlap periods. No feasible signal path can exist between *In* and *D*, as illustrated by the arrows.

exists a time slot where both the NMOS and PMOS transistors are conducting. This creates a path between input and output that can destroy the state of the circuit. Simulations have shown that the circuit operates correctly as long as the clock rise time (or fall time) is smaller than approximately five times the propagation delay of the register. This criterion is not too stringent, and it is easily met in practical designs. The impact of the rise and fall times is illustrated in Figure 7-28, which plots the simulated transient response of a C^2MOS *D* FF for clock slopes of, respectively, 0.1 and 3 ns. For slow clocks, the potential for a *race condition* exists.

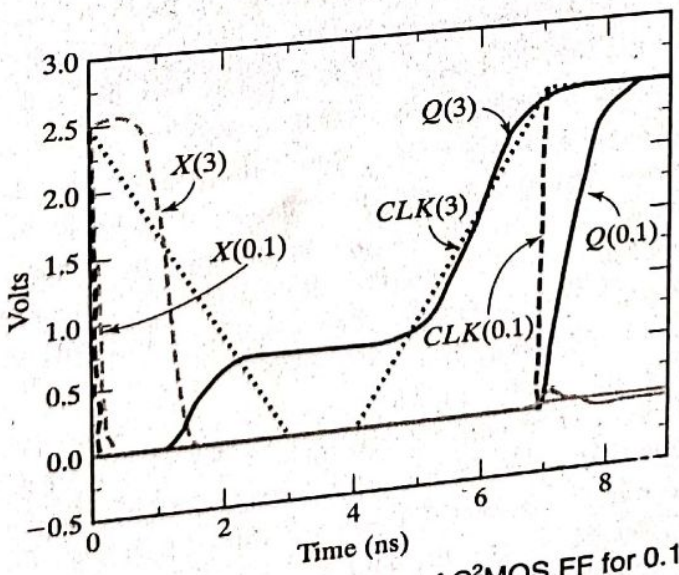


Figure 7-28 Transient response of C^2MOS FF for 0.1-ns and 3-ns clock rise/fall times, assuming *In* = 1.

Dual-Edge Registers

So far, we have focused on edge-triggered registers that sample the input data on only one of the clock edges (rising or falling). It also is possible to design sequential circuits that sample the input on both edges. The advantage of this scheme is that a lower frequency clock—half the original rate—is distributed for the same functional throughput, resulting in power savings in the clock distribution network. Figure 7-29 shows a modification of the C²MOS register enabling sampling on both edges. It consists of two parallel master–slave edge-triggered registers, whose outputs are multiplexed by using tristate drivers.

When clock is high, the positive latch composed of transistors M_1–M_4 is sampling the inverted D input on node X. Node Y is held stable, since devices M_9 and M_{10} are turned off. On the falling edge of the clock, the top slave latch M_5–M_8 turns on, and drives the inverted value of X to the Q output. During the low phase, the bottom master latch (M_1, M_4, M_9, M_{10}) is turned on, sampling the inverted D input on node Y. Note that the devices M_1 and M_4 are reused, reducing the load on the D input. On the rising edge, the bottom slave latch conducts and drives the inverted version of Y on node Q. Data thus changes on both edges. Note that the slave latches operate in a complementary fashion—that is, only one of them is turned on during each phase of the clock.

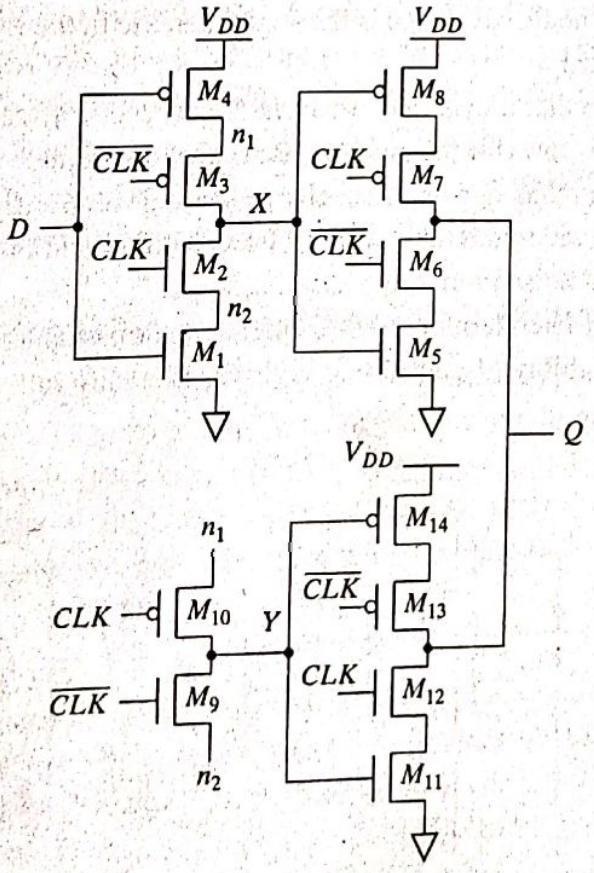
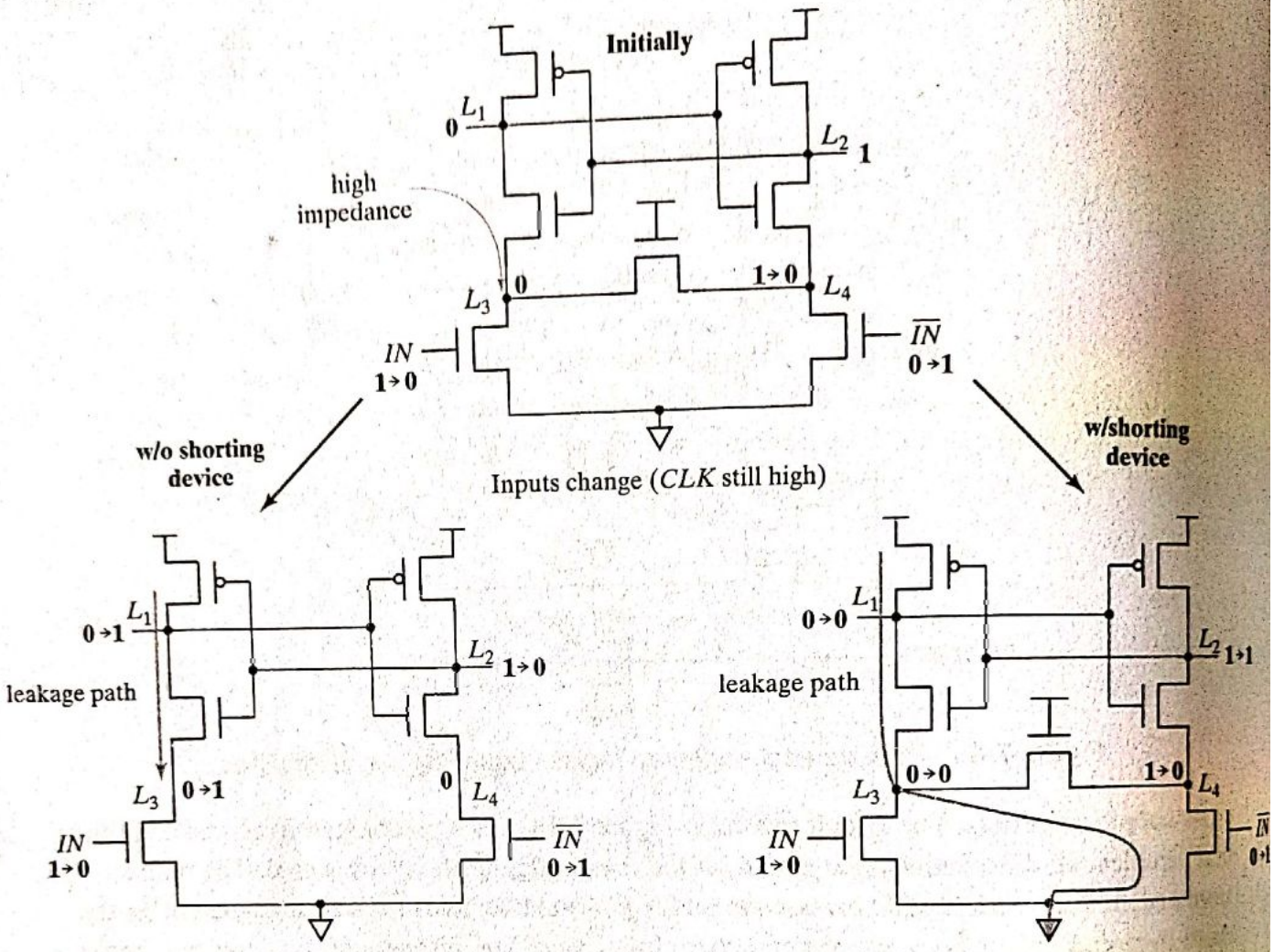


Figure 7-29 C²MOS-based dual-edge triggered register.



L_3 is isolated so charge accumulates until L_1/L_3 change state, causing L_2 to change state as well. As a result, the flip-flop outputs change.

The leakage current attempts to charge L_1/L_3, but the DC path through the shorting transistor allows it to leak away to ground.

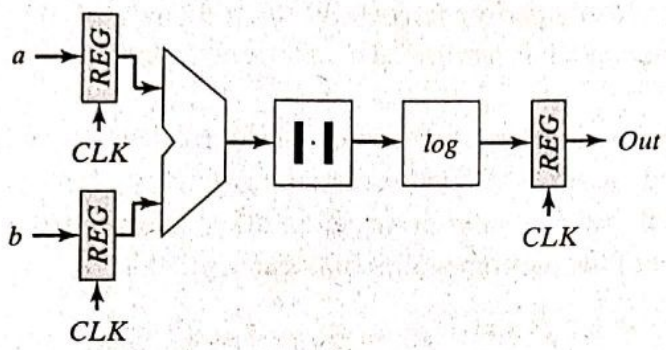
Figure 7-39 The need for the shorting transistor M_4.

7.5 Pipelining: An Approach to Optimize Sequential Circuits

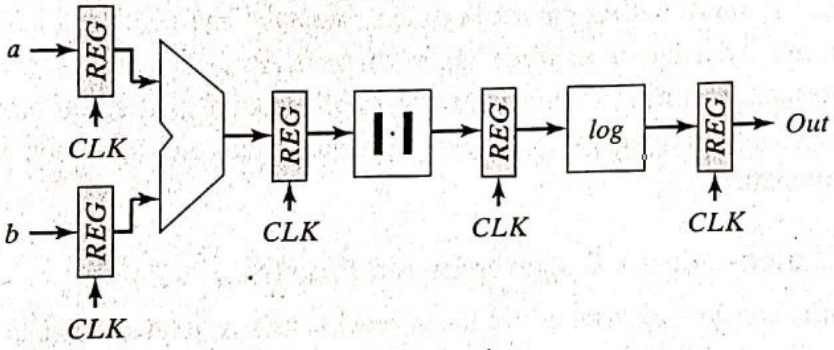
Pipelining is a popular design technique often used to accelerate the operation of datapaths in digital processors. The concept is explained with the example of Figure 7-40a. The goal of the presented circuit is to compute $\log(|a + b|)$, where both a and b represent streams of numbers (i.e., the computation must be performed on a large set of input values). The minimal clock period T_{min} necessary to ensure correct evaluation is given as

$$T_{min} = t_{c-q} + t_{pd,logic} + t_{su} \tag{7.7}$$

where t_{c-q} and t_{su} are the propagation delay and the setup time of the register, respectively. We assume that the registers are edge-triggered D registers. The term $t_{pd,logic}$ stands for the worst case delay path through the combinational network, which consists of the adder, absolute value, and logarithm functions. In conventional systems (that don't push the edge of technology), the



(a) Reference circuit



(b) Pipelined version

Figure 7-40 Datapath for the computation of $\log(|a + b|)$.

latter delay is generally much larger than the delays associated with the registers and dominates the circuit performance. Assume that each logic module has an equal propagation delay. We note that each logic module is then active for only one-third of the clock period (if the delay of the register is ignored). For example, the adder unit is active during the first third of the period and remains idle (no useful computation) during the other two-thirds of the period. Pipelining is a technique to improve the resource utilization, and increase the functional through-put. Assume that we introduce registers between the logic blocks, as shown in Figure 7-40b. This causes the computation for one set of input data to spread over a number of clock-periods, as shown in Table 7-1. The result for the data set (a_1, b_1) only appears at the output after three clock periods.

Table 7-1 Example of pipelined computations.

Clock Period	Adder	Absolute Value	Logarithm
1	$a_1 + b_1$		
2	$a_2 + b_2$	$\|a_1 + b_1\|$	
3	$a_3 + b_3$	$\|a_2 + b_2\|$	$\log(\|a_1 + b_1\|)$
4	$a_4 + b_4$	$\|a_3 + b_3\|$	$\log(\|a_2 + b_2\|)$
5	$a_5 + b_5$	$\|a_4 + b_4\|$	$\log(\|a_3 + b_3\|)$