CHAPTER
5

Static MOS Gate Circuits

CHAPTER OUTLINE

5.1 Introduction

This chapter addresses the design of combinational and sequential CMOS static logic gates. The material is built upon the foundations established in Chapter 4. In this chapter, we will continue to view the transistor as a logic switch with series resistance. For a unit-sized NMOS device, the series resistance is approximately 12.5 kΩ, whereas a unit-sized PMOS device is approximately 30 kΩ. These large-signal resistance values represent the average on-resistances when switching from high to low or low to high. Using the concepts of switch-based logic design, we can construct combinational logic gates such as NANDs, NORs, XORs, and multiplexers. We will examine the operation of these gates in detail and then study the voltage transfer characteristics (VTC), transistor sizing, first-order timing characteristics, and power. Extensions of switch-based networks can be used to implement

complex logic functions in a single CMOS stage through the application of the duality principle and DeMorgan's Laws.

Sequential logic elements have the ability to store information. The outputs of a sequential block are functions of both input and output values. As such, they are characterized by positive feedback loops between the outputs and inputs. In this chapter, the definitions and distinctions between two types of sequential elements, flip-flops and latches, are reviewed. Sequential circuits are constructed using conventional CMOS or pseudo-NMOS logic. In later chapters, we use clever techniques to reduce the number of transistors needed to implement these functions. The basic operation, limitations, and timing constraints for a variety of sequential elements are examined, including SR latches, JK flip-flops, D flip-flops, and D latches. We also describe certain timing constraints, the so-called set-up and hold times, for D-type flip-flops and latches.

Power and timing are the two main design specifications for digital integrated circuits. In this chapter, we explore the relationships between these two factors. We seek to minimize power and reduce delay as much as possible, but usually one is traded off against the other. Circuit activity is a key factor in the total power dissipation. Reducing the activity of a chip is the best way of reducing dynamic (or switching) power in CMOS circuits. This can be accomplished most effectively by choosing logic architectures that exhibit low activity. In addition, the arrival times of signals at gate inputs should be made equal to minimize glitches that unnecessarily contribute to power consumption. Further, the rise and fall times of all signals should be equalized, wherever possible, to limit power due to short-circuit current flow during switching. Power consumption of most circuits can also be decreased by reducing the operating voltage. This is the main reason why the supply voltage is reduced in each technology generation.[1] However, the threshold voltage has not been scaling as rapidly in order to control subthreshold leakage. When many millions of transistors are sitting idle but leaking current, it contributes significantly to the static (or standby) power consumption.

Since power and timing are usually traded off when designing circuits, a metric is needed to establish the goodness of a design relative to these two factors. To improve a design, we should try to reduce both power and delay. Therefore, a useful metric can be devised by multiplying the power by the delay to obtain a power-delay product, which is a measure of the energy needed to perform a given logic operation. This metric is most valuable when comparing single gate configurations that use the same supply voltage and drive the same load. However, it suffers from some limitations if used to compare the power of two different designs that perform the same function. A more effective metric when comparing two designs is the energy-delay product. This metric is obtained by multiplying the energy (i.e., power-delay product) by delay.

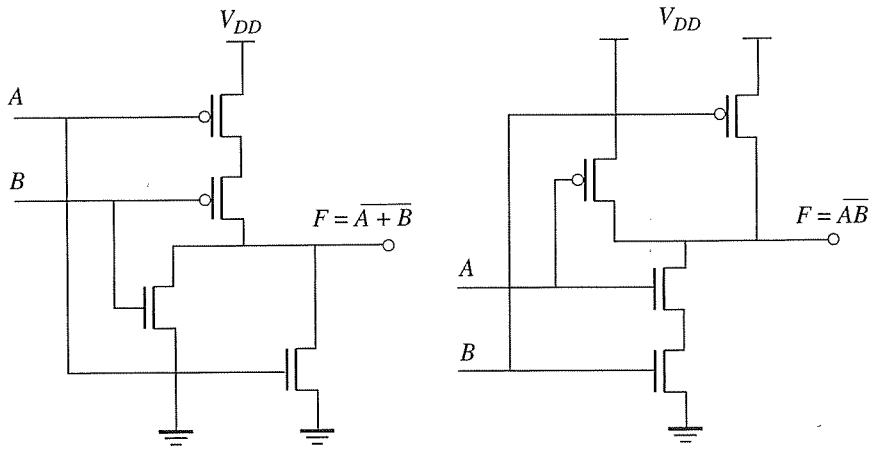
[1] Roughly speaking, the power supply voltage (in volts) is 10 times the technology generation (given in microns). For example, a 3.3 V supply is used in a 0.35 μm technology, a 1.8 V supply in a 0.18 μm technology, and a 1.2 V supply in a 0.13 μm technology.

In this chapter, basic static NAND and NOR gate circuits are described in Section 5.2 with emphasis on gate sizing, voltage transfer characteristics, and fanin/fanout considerations. Section 5.3 illustrates how complex functions can be implemented in a single CMOS stage using duality and DeMorgan's Law. The other useful logic functions such as XOR, XNOR, and multiplexers are described in Sections 5.4 and 5.5, respectively. Basic sequential circuit elements are explored in Sections 5.6 and 5.7. To conclude the chapter, we examine the important components of power dissipation in Section 5.8, and power-delay tradeoffs in Section 5.9.

5.2 CMOS Gate Circuits

To build NOR and NAND gates in CMOS is rather straightforward. We use the switch-level concepts that two NMOS transistors in series perform a logical AND function and two NMOS transistors in parallel perform a logical OR function. Similarly, two PMOS transistors in parallel perform an AND function and two PMOS transistors in series perform an OR function. The resulting circuits for the 2-input NOR and 2-input NAND gates are illustrated in Figure 5.1. For the 2-input NOR gate of Figure 5.1a, the output is pulled low if either A or B is high. This requires two pull-down devices in parallel. However, both inputs must be low to produce a high output. This requires two series p-channel devices connected from the output to V_{DD}. All NMOS devices have their bulk terminals connected to Gnd, while the PMOS devices have their bulk nodes connected to V_{DD}.

For the 2-input NAND gate of Figure 5.1b, the output is low only if both A and B are high. This implies two series-connected n-channel devices from the output to ground. On the other hand, if either input is low, then the output will go high. This can be implemented with two parallel p-channel devices connected from V_{DD} to the output. The extension to multiple input gates follows along the same lines. Specifically, for a multiple *fanin* NOR gate, we would have many parallel NMOS devices



(a) 2-input NOR gate (b) 2-input NAND gate

Figure 5.1
CMOS NOR and NAND gates.

and a *stack* of series-connected PMOS devices. For the NAND gate with multiple inputs, we would have a *stack* of NMOS devices connected to the output, and an equal number of parallel PMOS devices connected to the output.

Exercise 5.1 Draw 3-input and 4-input NOR and NAND gates. We often refer to these gates with names such as NAND3, NOR3, NAND4, and NOR4 for convenience. Explain how they each work. Do you see any problems arising as you increase the number of fanins (inputs) to 10?

5.2.1 Basic CMOS Gate Sizing

Now consider transistor sizing of CMOS logic gates, specifically NANDs and NORs. As mentioned before, transistor sizing is the process of specifying the widths of the transistors in the logic gates.[2] When we studied the inverter in the previous chapter, it was clear that device sizes control the rise and fall propagation delays in static CMOS. This is the primary reason why we specify the sizes of the transistors. To obtain approximately equal rise/fall delays, we found that the PMOS device must be roughly twice the size of the NMOS device. This is because its on-resistance for PMOS is roughly double that of the NMOS device.[3] For the time being, we will use the 2:1 ratio. Later, we will elaborate on the best ratio for the pull-up and pull-down transistors.

Given the 2:1 device sizes for the inverter of Figure 5.2a, we can now determine the transistor sizes in multifanin gates. Assume for the moment that all gates drive a single load capacitance, C_L and that we want to obtain the same delay as the inverter for the rise/fall cases. First, consider the NAND gate in Figure 5.2b. There are two parallel PMOS devices and two series NMOS devices. Sizing the four transistors is based on achieving the same delay as the inverter when driving the same load capacitance under worst-case input combinations. Since the two PMOS devices are in parallel, we assume that only one is *on* and the other is *off* during the pull-up phase. If we assume that both pull-ups are *on*, then this is the best-case situation. In design, we always consider the worst case wherever possible to ensure that our circuit works under all conditions. For this reason, each pull-up should be sized to be *equal* to the inverter pull-up, meaning that both pull-ups will be $2W$.

Next consider the two pull-down transistors that are in series for the NAND gate. Since both transistors must be *on* to pull the output low, we will see roughly twice the resistance if they are sized exactly the same as the inverter. Instead, we want to cut the resistance in half and so we double the widths.[4] This requires a size

[2] We assume that all devices have a minimum length, L, and so the width, W, is the only unknown.

[3] More precisely, the ratio of the PMOS resistance to NMOS resistance is 30 kΩ/12.5 kΩ = 2.4. To obtain equal delays we should size the PMOS device to be 2.4X larger than the NMOS device. However, we will find that there are a number of factors that make a 2:1 ratio a better choice. In fact, this 2:1 ratio was also the preferred ratio in the past due to mobility considerations.

[4] Recall that resistance is inversely proportional to the width.

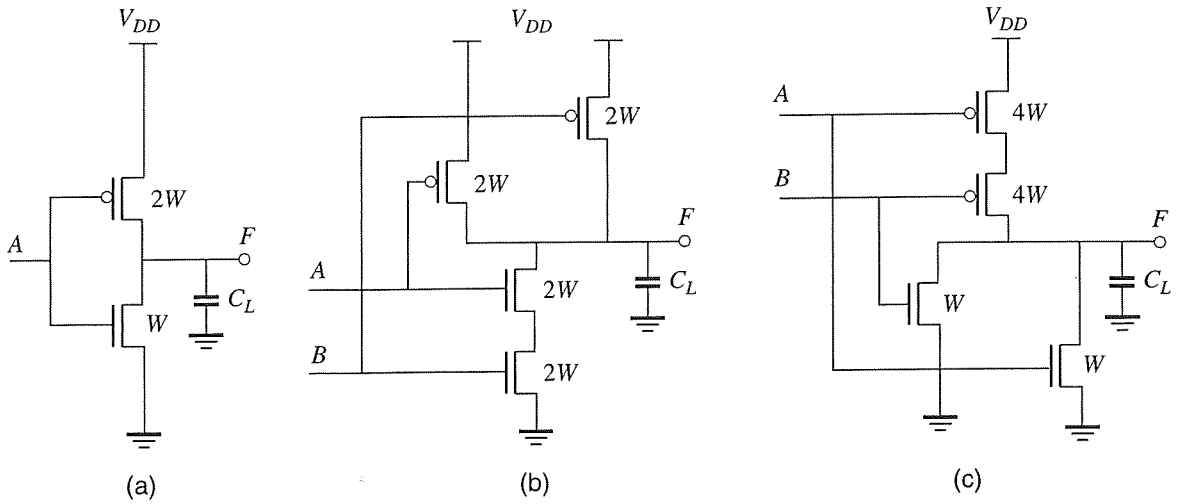


Figure 5.2
Device sizes for CMOS (a) inverter, (b) NAND, and (c) NOR gates.

of $2W$ for both pull-down devices.[5] As a result, all four devices in the NAND gate are $2W$ devices.

Similar considerations apply to the NOR gate of Figure 5.2c. Here the PMOS devices are in series and the NMOS devices are in parallel. Using the same arguments, both NMOS devices should be set to W to match the pull-down delay of the inverter in the worst case. The PMOS devices need to be doubled relative to the inverter case since they are in series. This leads to two $4W$ devices in series for the pull-up case to effectively cut the resistance in half for each device.

The same approach would be used to size multi-input pseudo-NMOS gates. The pseudo-NMOS inverter was covered in Chapter 4 and is shown again in Figure 5.3a. The size of the pull-up device is set to W_p and the pull-down device to W_n. The ratio of W_n/W_p depends primarily on the desired V_{OL}. For pseudo-NMOS gates, only the NMOS devices need to be sized since there is one PMOS pull-up and its size would remain the same as in the pseudo-NMOS inverter[6] of Figure 5.3a.

Consider sizing 3-input pseudo-NMOS NAND and NOR gates. For the NAND gate of Figure 5.3b, we would use devices with size $3W_n$; that is, use devices that are three times larger than its inverter counterpart. The reason is that there are three resistors in series and we must increase the size of the transistors by three to reduce the effective resistance of the pull-down path to that of the inverter. For the NOR gate of Figure 5.3c, the pull-down devices would all be the same size as the pseudo-NMOS inverter case since they are in parallel. Again, the worst-case condition has only one transistor pulling down, not all three. Therefore, the size must be set with this in mind.

The sizing of the NAND gate of Figure 5.3b can be understood from two other perspectives. First, it can be viewed from a layout perspective, as shown in Figure 5.4.

[5] This is true for long channel devices. More accurate device sizes would include velocity saturation effects but they are not considered until Chapter 6.

[6] There is an assumption here that the only capacitance is a single output loading C_L.

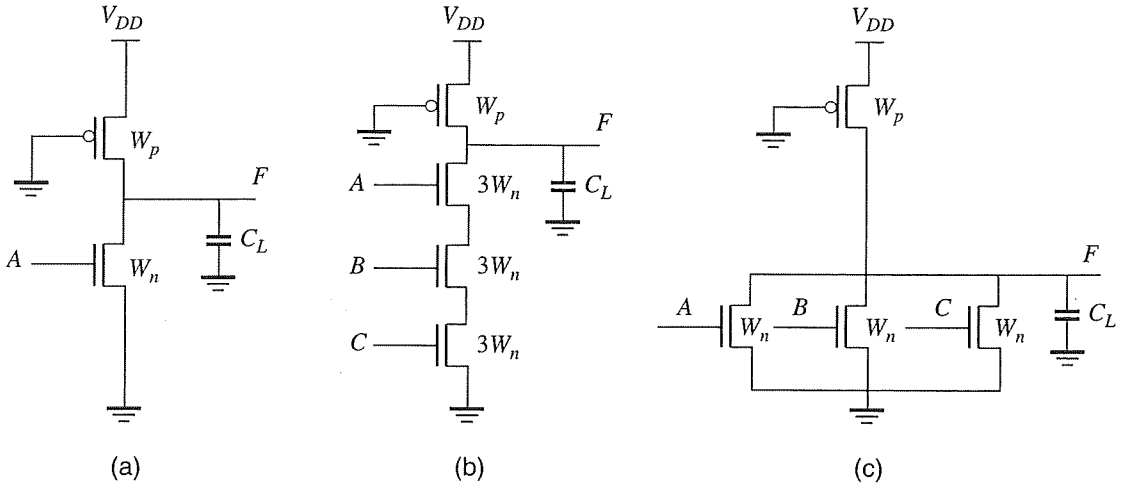


Figure 5.3
Device sizing for pseudo-NMOS (a) inverter, (b) NAND, and (c) NOR gates.

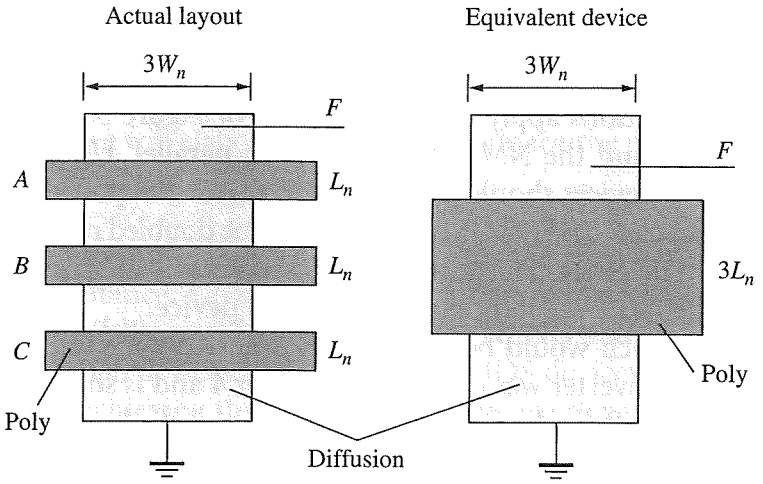


Figure 5.4
Layout and equivalent size of 3X device.

The layout of the NMOS series stack is given on the left, with three inputs, A, B, and C, on the poly layer. One diffusion region is defined with a width of $3W_n$, and three transistors with length L_n are created. The output F is identified at the top of the stack. When the inputs are connected together, the three devices in series can be merged to produce one "super" device with a size of $3W_n/3L_n$. This produces an effective size of W_n/L_n, which is equivalent to the size for the pseudo-NMOS pull-down of the inverter. Note that the sizing achieves the same delay characteristics as the inverter but also produces the same V_{OL}. In other words, we size pseudo-NMOS gates for both V_{OL} and timing when we choose the W/L ratios for the individual devices.

Another way to view the sizing of series stacks is to recall that the transistor on-resistance is inversely proportional to W. Larger values of W decrease the series

resistance of the transistor. What if we had three different W values in the series stack? When determining an equivalent W_{eq}, one can use the fact that the W values would combine as if we had parallel resistance.[7] For example, if we have three transistors in series with widths W_1, W_2, and W_3, we would combine them together to form an equivalent device with a width

$$W_{eq} = \frac{W_1 W_2 W_3}{W_1 W_2 + W_2 W_3 + W_3 W_1} \qquad \text{(series stack, all devices on)} \qquad (5.1)$$

This value can be used to determine on-resistance of the combined devices. For the case when all W's are equal to $3W_n$, this reduces to W_n, as expected.

In the case of a NOR gate, if all parallel pull-down devices are on simultaneously, the W values would combine as if we had series resistance to produce the equivalent value, W_{eq}:

$$W_{eq} = W_1 + W_2 + W_3 \qquad \text{(parallel devices, all devices on)} \qquad (5.2)$$

Of course, we design the NOR gate assuming that only one pull-down device is *on* at a time, since this is the worst case.

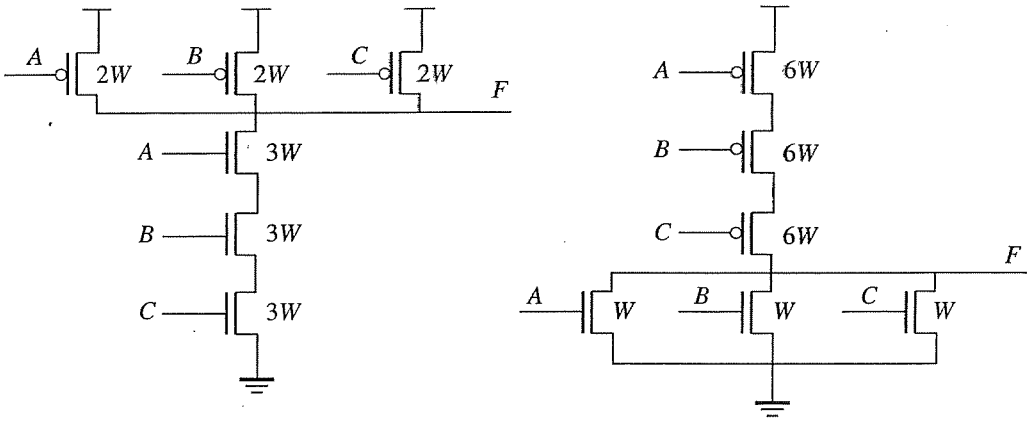
3-Input NAND and NOR Gate Sizing **Example 5.1**

Problem:

Determine the device sizes for 3-input NAND and NOR gates in conventional CMOS. Assume that the basic inverter is sized according to Figure 5.2a and that the goal of the design is to have NAND3 and NOR3 gates with the same delay characteristics as the inverter.

Solution:

Applying the considerations given in the above section, we derive the following sizes for the NAND3 and NOR3, respectively.



[7] Here we make use of the inverse relationship between R and W. The series stack produces series resistance, not parallel resistance!

Exercise 5.2 Determine the device sizes for NAND4 and NOR4 gates (i.e., 4-input versions of each type of gate). When you draw the two circuits, notice how high the series stacks are getting. We usually do not see 5-input gates because of the height of the series stacks and the corresponding resistances and layout areas.

5.2.2 Fanin and Fanout Considerations

When designing standard CMOS logic gates, it should be noted that there is a significant penalty for gates with a large number of inputs, so-called high *fanin* gates. This is due to the resistance of series stacks. Beyond 3 or 4 inputs, the resistance will be too high or the area will be too large. Consider the case of an 8-input AND gate. What is the best way to implement such a gate? It is certainly not as a single NAND with eight inputs followed by an inverter. The eight series NMOS devices would have to be very large to lower the resistance, as shown in Figure 5.5. However, the delay would still be unacceptable since the series stack would behave as an RC ladder circuit, due to the resistance and self-capacitance[8] of each transistor.

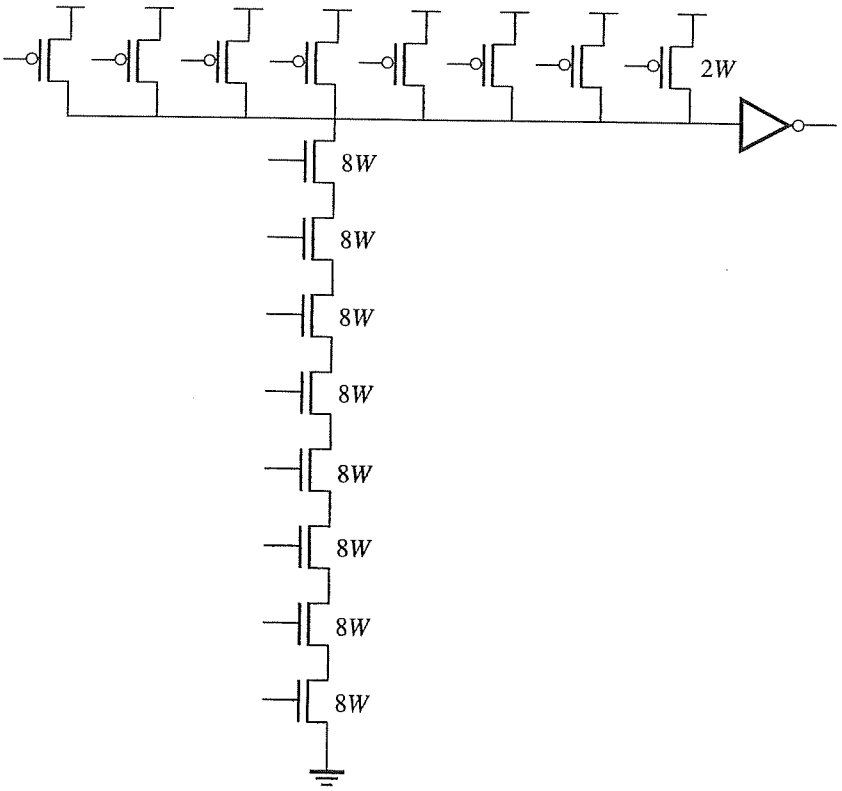


Figure 5.5
Eight-input AND gate.

[8] Self-capacitance effects would introduce a capacitor at each internal node of the series stack. These issues will be described in Chapter 6.

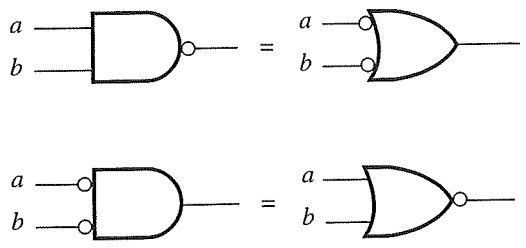


Figure 5.6
Schematic rendition of DeMorgan's Law.

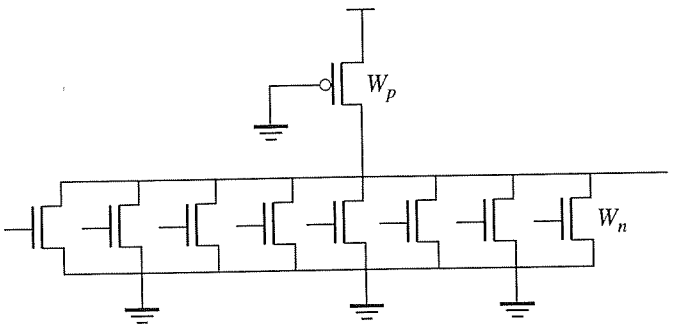
One alternative is to use DeMorgan's Laws and a pseudo-NMOS NOR gate to replace the AND gate. DeMorgan's Laws, in the most basic form, can be stated as

$$\overline{(a + b)} = \bar{a} \cdot \bar{b}$$
$$\overline{(a \cdot b)} = \bar{a} + \bar{b} \tag{5.3}$$

Graphically, we can represent the two equations as shown in Figure 5.6.

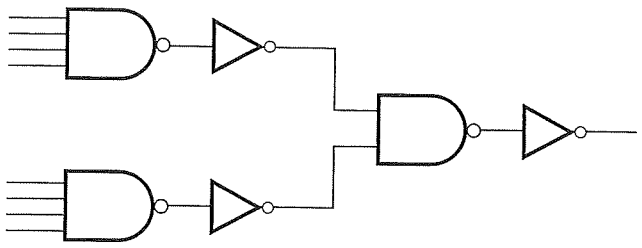
The application of DeMorgan's Law for the AND8 is shown in Figure 5.7a. Here the inputs to the AND gate are all inverted. We will need additional inverters at each input but they are not shown. The AND gate is converted to an 8-input NOR. The pseudo-NMOS implementation is shown in Figure 5.7b. This gate dissipates more power than the static implementation and exhibits a larger t_{PLH}. But the area is greatly reduced relative to the 8-input CMOS NAND because we only have one pull-up device, the pull-downs are much smaller, and the t_{PHL} is significantly lower.

(a)

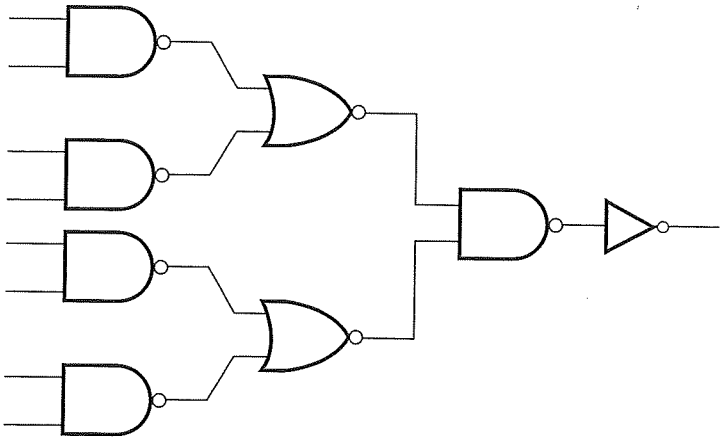


(b)

Figure 5.7
Using DeMorgan's Law to convert AND to NOR.



(a) NAND4-INV-NAND2-INV



(b) NAND2-NOR2-NAND2-INV

Figure 5.8
Multilevel logic implementation of an AND8 function.

We would only use this in a few places where a high fanin gate is required because of its drawbacks.

Another option is to construct a multilevel logic circuit to implement the AND function rather than using a single gate. For example, one possible circuit is the NAND4-INV-NAND2-INV structure as shown in Figure 5.8a. This is clearly more effective than a NAND8-INV implementation and can also be designed to have a shorter delay.

Many other alternatives exist along the same lines. In Figure 5.8b, a NAND2-NOR2-NAND2-INV cascade is illustrated. The best design for a given application will be dependent on timing, power, and area requirements.

Exercise 5.3 Suggest two other multilevel implementations of an AND8 circuit.

The *fanout* of a logic gate refers to the number of identical logic gates driven by it. This is illustrated in Figure 5.9. For example, an inverter driving four identical inverters is called a fanout-of-four (FO4) inverter. The fanout ratio is obtained by taking the total capacitance driven by the gate and dividing it by the input capacitance of the gate. When *cell libraries* are designed, the sizes of the devices must be

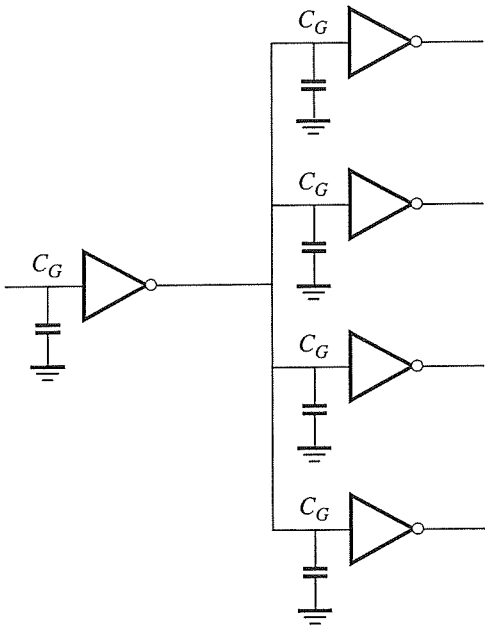


Figure 5.9
An inverter driving four identical fanouts.

specified without knowing the actual load being driven. Typically, gates are designed to drive a specific number of fanouts to deliver a specific target delay. In fact, different cells are designed to drive a different number of loads. Ultimately, the choice of the width, W, is based on this fanout ratio, as will be seen in Chapter 6.

5.2.3 Voltage Transfer Characteristics (VTC) of CMOS Gates

The VTC of CMOS gates are very similar to that of the CMOS inverter. Since there are multiple inputs, the actual characteristics depend on how the inputs are switched. For the 2-input NAND gate, shown in Figure 5.10, we first consider the case when one input is held high while the other is switched from low to high. In this case, one p-channel device can be removed from consideration (open circuit) while one n-channel device can be handled as a short-circuit since it behaves like a small resistor. If input A is held high while input B is switched from low to high, the gate is equivalent to an inverter with a $2W$ pull-up device and a $2W$ pull-down device. The resulting transfer characteristic would shift to the left of the standard inverter. If both inputs were switched together, we would have a $4W$ pull-up device and a W pull-down device, so the VTC would shift to the right of the standard inverter. These two curves are shown in Figure 5.10 relative to the standard inverter.

For the 2-input NOR gate, shown again in Figure 5.11, we would obtain similar results. Assume that input $B = 0$, while the A input is switched from low to high. In this case, one n-channel device can be removed from consideration (open circuit) while one p-channel device can be treated as a short-circuit. This implies that we have an equivalent inverter with a $4W$ pull-up device and a W pull-down device.

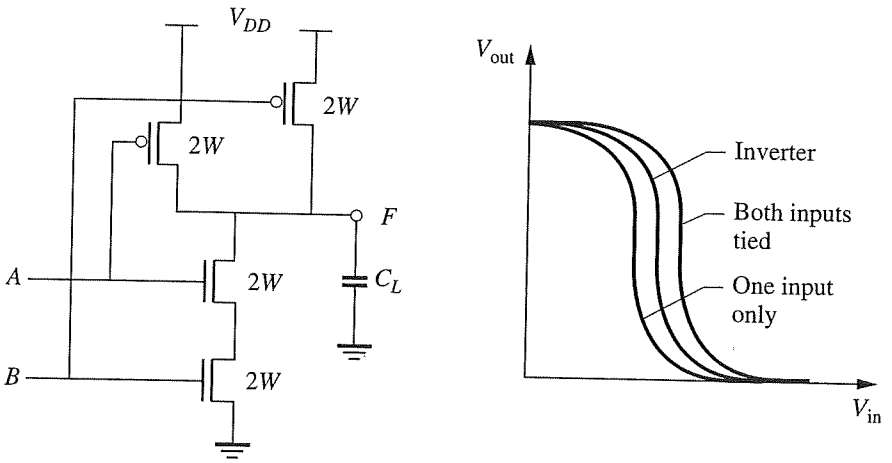


Figure 5.10
VTC for NAND2 gate.

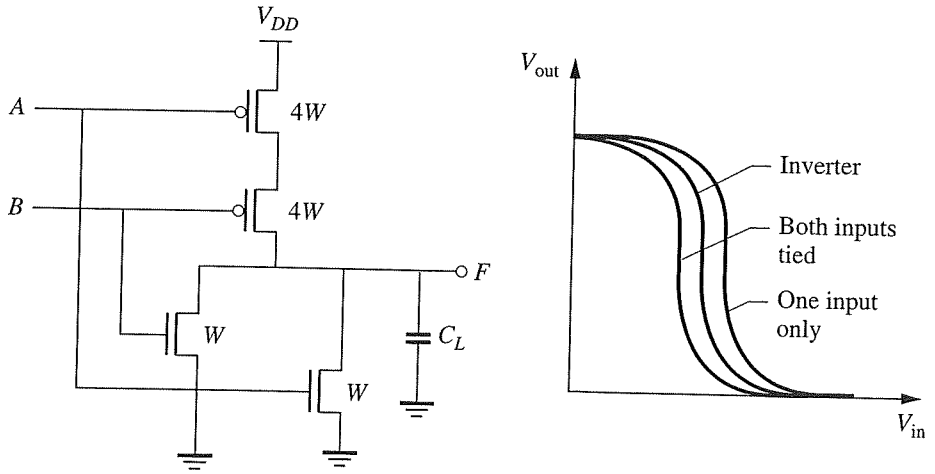


Figure 5.11
VTC for NOR gate.

Therefore, the V_S should shift to the right of a standard 1X inverter VTC. A second case exists when both inputs are connected together and switched from low to high. We have effectively a $2W$ pull-up and a $2W$ pull-down, which would produce a switching threshold, V_S, that is shifted to the left of the standard inverter. These results are illustrated in Figure 5.11.

Similar considerations would apply to multiple input NAND and NOR gates. Each individual input, if applied separately, produces a slightly different VTC (but not significantly different). In any case, the single-input switching case is usually taken as the VTC of the gate. From inspection of Figures 5.10 and 5.11, the NOR gate has a higher switching threshold than the NAND gate, and therefore NM_L is higher and NM_H is lower for the NOR gate as compared to the NAND gate.

SPICE Example of VTC for NAND Gate in 0.18 μm Technology **Example 5.2**

Problem:

(a) Compute the switching threshold, V_S, of a 2-input NAND gate shown in Figure 5.10 under two conditions: with one input tied to V_{DD} while the other input is swept from 0 to V_{DD} and a second case where both inputs are tied together. Assume 0.18 μm technology parameters with all transistor widths set to 400 nm.

(b) Using SPICE, compare the VTC curves of a 2-input NAND for the two cases above and compare the switching threshold results. Also include the two inputs, A and B, switching separately. Why are the results slightly different for each input?

Solution:

(a) When one input is high and the other is varied from 0 to V_{DD}, it is equivalent to having a pull-up with $W_P = 400$ nm and a pull-down of $W_N = 400$ nm (treat the other one as a very small resistor):

$$W_N = 400 \text{ nm}, \quad W_P = 400 \text{ nm}:$$

$$X = \sqrt{\frac{W_N/E_{CN}L_N}{W_P/E_{CP}L_P}} = \sqrt{\frac{W_N E_{CP}}{W_P E_{CN}}} = \sqrt{\frac{(400)(24)}{(400)(6)}} = 2.0$$

$$V_S = \frac{1.8 - 0.5 + (0.5)2.0}{1 + 2.0} = 0.77 \text{ V}$$

When both inputs are varied from 0 to V_{DD}, it is equivalent to having a pull-up with $W_P = 800$ nm and a pull-down of 200 nm:

$$W_P = 800 \text{ nm}, \quad W_P = 200 \text{ nm}:$$

$$X = \sqrt{\frac{W_N/E_{CN}L_N}{W_P/E_{CP}L_P}} = \sqrt{\frac{W_N E_{CP}}{W_P E_{CN}}} = \sqrt{\frac{(200)(24)}{(800)(6)}} = 1.0$$

$$V_S = \frac{1.8 - 0.5 + (0.5)1.0}{1 + 1.0} = 0.9 \text{ V}$$

(b) To compare the results against SPICE, use the following descriptions:

```
*NAND Gate with one input B tied to Vdd, input A switching
*Set supply and library
.param Supply=1.8                    *for 0.18 technology
.lib   'bsim3v3.cmosp18.lib'
.opt scale=0.1u                      *Set lambda
*Power supply
.global     Vdd   Gnd
Vdd         Vdd   Gnd      'Supply' *Supply is set by .lib call
```

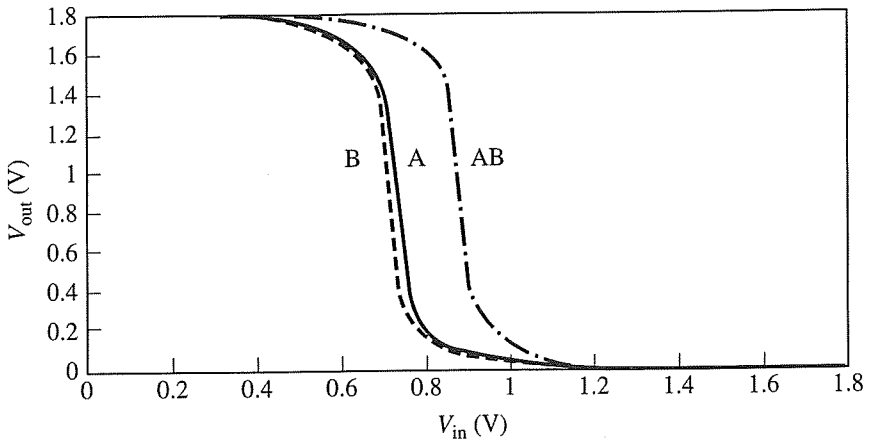
```
*Top level simulation netlist
mpa     out  A     Vdd      Vdd      PMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mpb     out  Vdd   Vdd      Vdd      PMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mna     out  A     x        Gnd      NMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mnb     x    Vdd   Gnd      Gnd      NMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
Vin     A    Gnd   'Supply'
* Simulation
.dc     Vin  0     'Supply'         'Supply/50'
.plot   dc   V(out)
.end

*NAND Gate with input A tied to Vdd, input B switching
*Set supply and library
.param Supply=1.8            *for 0.18 technology
.lib   'bsim3v3.cmosp18.lib'
.opt scale=0.1u              *Set lambda
*Power supply
.global       Vdd   Gnd
Vdd           Vdd   Gnd      'Supply' *Supply is set by .lib call
*Top level simulation netlist
mpa     out    Vdd   Vdd      Vdd    PMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mpb     out    B     Vdd      Vdd    PMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mna     out    Vdd   x        Gnd    NMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mnb     x      B     Gnd      Gnd    NMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
Vin     B      Gnd   'Supply'
*Simulation
.dc     Vin    0     'Supply'         'Supply/50'
.plot   dc     V(out)
.end


*NAND Gate with both inputs tied together
*Set supply and library
.param Supply=1.8            *for 0.18 technology
.lib    'bsim3v3.cmosp18.lib'
.opt scale=0.1u              *Set lambda
*Power supply
.global Vdd    Gnd
Vdd            Vdd    Gnd    'Supply'*Supply is set by .lib call
*Top level simulation netlist
mpa      out    in     Vdd    Vdd    PMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mpb      out    in     Vdd    Vdd    PMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mna      out    in     x      Gnd    NMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
mnb      x      in     Gnd    Gnd    NMOS1 l=2 w=4 ad=20 pd=4 as=20 ps=4
```

```
Vin     in      Gnd     'Supply'
*Simulation
.dc     Vin     0       'Supply'        'Supply/50'
.plot   dc      V(out)
.end
```

The following results are produced by SPICE:



As expected, the curve with both inputs tied is to the right of the curve for single inputs. From SPICE, $V_S = 0.745$ V for input A switching, $V_S = 0.721$ V for input B switching, and $V_S = 0.883$ V for the case where the two inputs are tied together. These are close to the hand calculated values. The hand calculations tend to produce slightly higher values due to the approximations made to simplify the expression for V_S. The reason why input A produces a slightly higher value of V_S is that the corresponding transistor experiences body effect during the switching process.

5.3 Complex CMOS Gates

There are many situations where we would like to implement more complicated functions than just NANDs and NORs. For example, if we wanted to realize the logic for

$$F = \overline{(A + B) \cdot C} \tag{5.4}$$

we could implement it as a series of multilevel logic gates consisting of ANDs, ORs, and inverters. However, CMOS lends itself to efficient implementation of such functions through the use of *duality* and *DeMorgan's Laws*. Implementations of this variety are referred to as AND-OR-INVERT (AOI) and OR-AND-INVERT (OAI)

gates. Consider the fact that in a NAND gate, we have parallel PMOS devices and series NMOS devices. These are duals of each other. Similarly, NOR gates have parallel NMOS devices and series PMOS devices. Again, these are duals of each other.

The dual of any logic function can be obtained by exchanging the AND and OR operations. For example, the following functions are duals:

$$ab \Leftrightarrow a + b$$

$$(a + b)c \Leftrightarrow ab + c$$

We encountered a special case of DeMorgan's Laws earlier in the chapter. In general, DeMorgan states that the complement of a function can be obtained by replacing each variable or element with its complement, and then exchanging the AND and OR functions. This is one of the most valuable rules in Boolean algebra and it is extremely useful in constructing the PMOS and NMOS configurations for complex CMOS gates.

For the function given in Equation (5.4), we could apply DeMorgan as follows:

$$F = \overline{(A + B) \cdot C} = \overline{(A + B)} + \overline{C} = \overline{A} \cdot \overline{B} + \overline{C} \tag{5.5}$$

The logic expressions for F in Equations (5.4) and (5.5) tell us how to implement the CMOS circuit in one logic stage. If we first examine Equation (5.4), it specifies how the NMOS devices should be configured. Since the NMOS transistors act to pull the output low, we should complement F to obtain its implementation. The NMOS function is $\overline{F} = (A + B) \cdot C$. The NMOS implementation requires two parallel transistors with inputs A and B in series with a third transistor with input C.

Equation (5.5) reveals how we should implement the PMOS function. Since the PMOS function pulls the output high and the devices effectively complement each term, we should implement the function after a term-by-term complement. That is, we should implement $F = AB + C$ using the PMOS devices. This requires two series PMOS devices to implement AB and one parallel transistor with the input C. The resulting function is an OAI function, as shown in Figure 5.12.

To generalize the example shown above, we need to combine duality and DeMorgan to produce a desired function in a single stage. In fact, we need to construct two switch networks to implement a function in CMOS: a pull-down network (n-complex) and a pull-up network (p-complex), as shown Figure 5.13. The n-complex is defined as the complement of the function that sets the output to 0 due to an active path to Gnd. The p-complex is defined as the function of complemented variables that sets the output to 1 creating a path to V_{DD}. The two functions should be duals of one another (after each literal is complemented).

Algorithmically, we should take the given function F and produce the complement to obtain the function that we implement in the n-complex. The dual network should be used for the p-complex. This is obtained by exchanging AND and OR operations. A visual inspection of the two networks for the duality property can be carried out to check for correctness at the end of the process. Sizing the transistors follows the same steps as described for the basic NAND and NOR gates.

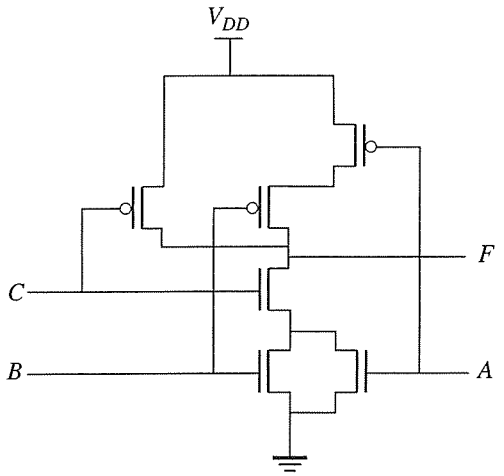


Figure 5.12
Complex CMOS gate implementation of $\overline{(A + B) \cdot C}$.

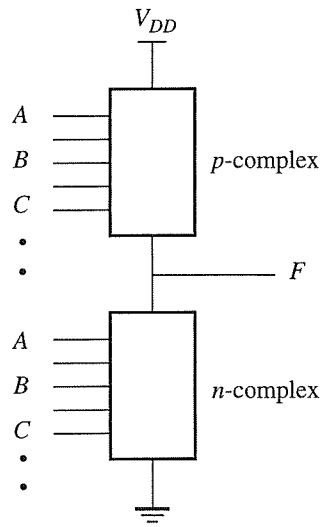


Figure 5.13
Generalized complex gate representation.

Complex CMOS Gate Circuits **Example 5.3**

Problem:

Implement $F = \overline{AB + CD}$ as a single-stage complex CMOS gate and a pseudo-NMOS gate.

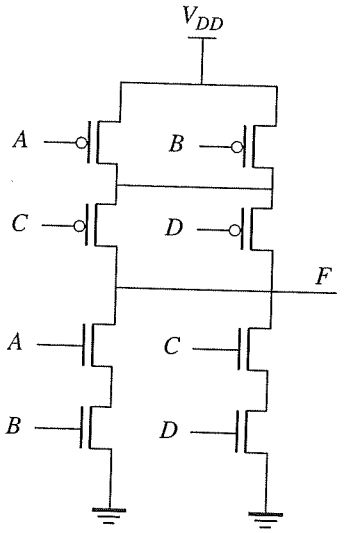
Solution:

For the CMOS gate: first complement F to obtain the n-complex function: $\overline{F} = AB + CD$. This implies that we need two series NMOS paths in parallel with one another. One series path implements AB while the other implements CD.

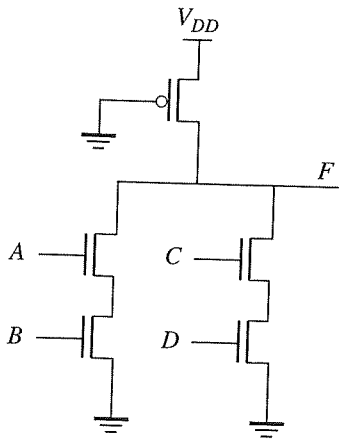
Next, take the dual of the complement of F:

$$\bar{F}|_{dual} = (A + B)(C + D)$$

This is the function implemented in the p-complex. The same expression would be obtained if we applied DeMorgan's Law to F and then complemented each literal. This implies that we need a series connection of two parallel devices, one with A or B and the other with C or D.



For the pseudo-NMOS gate, only the first part is needed to construct the n-complex. The pull-up network is, of course, only a single PMOS device. Clearly, this is more efficient than the CMOS case in terms of area.



5.4 XOR and XNOR Gates

Two logic functions that are important in digital design are the XOR and the XNOR functions:

$$f_{XOR} = \bar{a}\,b + a\bar{b} = (XOR)$$

$$f_{XNOR} = \bar{a}\,\bar{b} + ab = (XNOR)$$

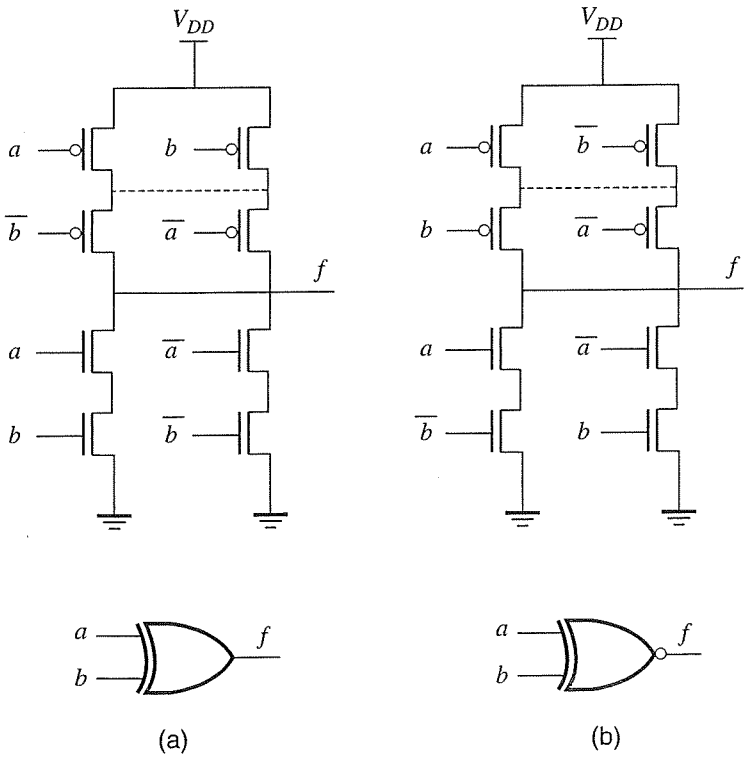


Figure 5.14
Static implementation of (a) XOR and (b) XNOR gates.

There are a variety of ways to implement these functions. A number of static versions of these gates will be presented here, while versions based on the CMOS transmission gate will be presented in later chapters.

The straightforward implementation assumes that we have access to a, $\overline{a}$, b, and $\overline{b}$. In that case, the implementations are simply based on the material in the above sections. The corresponding circuits are shown in Figure 5.14. For the XOR gate, we first complement the f_{XOR} function and use it to create the pull-down function. Therefore, two branches are needed, one for ab and the other for $\overline{a}\,\overline{b}$. The transistor configuration can be interpreted as follows: if both a and b are high, or a and b are low, the output is low. Otherwise, the output is high.

The pull-up branch implements the f_{XOR} function directly but replaces each term with its complement. The result is shown in Figure 5.14a. Note that the additional connection between the p-channel devices is unnecessary in this case (shown as a dotted line).

A similar process is used to obtain the XNOR gate of Figure 5.14b. In this case, the pull-down tree consists of the complement of f_{XNOR} with each branch implementing $\overline{a}\,b$ and $a\,\overline{b}$, respectively. This implies that if a and b are different, the output will be low. Otherwise it will be high. The pull-up network should implement the dual, with parallel connections in place of the series connections of the pull-down tree.

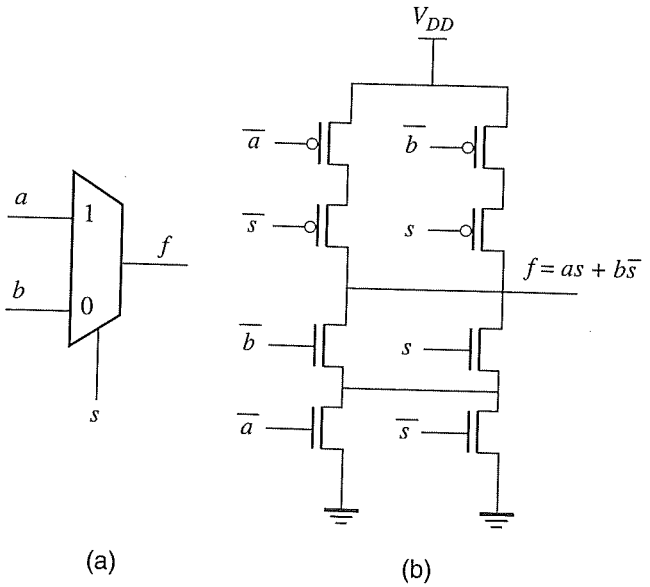


Figure 5.15
Multiplexer (a) logic symbol and (b) CMOS implementation.

5.5 Multiplexer Circuits

The multiplexer circuit produces an output by selecting one of N inputs. There are many ways to implement this type of circuit. We will examine transmission gate versions in Chapter 7. Here we examine the simplest possible multiplexer to select one out of two possible inputs. The function to be implemented is

$$f = as + b\bar{s} \quad (MUX)$$

where s is the selection signal and a and b are the two inputs. This function is reminiscent of the XOR and XNOR gates and can be implemented in a similar fashion as shown in Figure 5.15. The multiplexer is often referred to in its short form as the MUX.

5.6 Flip-Flops and Latches

In all logic circuits described so far, the output is directly related to the input by some logic combination. Typically, there are no feedback loops in these circuits, so the outputs are always a logical combination of the inputs. As a class, these circuits are known as *combinational logic circuits.*[9]

There is another class of circuits, known as *sequential logic circuits*, in which the new outputs also are dependent on the inputs and the preceding values of outputs. Examples are counters and data registers. A characteristic of static sequential circuits is that one or more output nodes are intentionally connected back to inputs to give *positive feedback,* or *regeneration.* This is the same type of regenerative property

[9] Some have been known to inadvertently use the term combinatorial circuits rather than the correct term: combinational circuits.

that was described for inverter chains, except that the signal propagates around feedback loops in sequential circuits.

Basic to sequential circuits is the *bistable circuit.* This type of circuit has two stable operating points. Prominent examples of the bistable circuit found in digital ICs are

1. Latches
2. Flip-flops

These two terms have been used interchangeably in the literature, but there is a fundamental difference between them. We will explore this difference in more detail throughout this section, but a brief explanation of the two types of elements is in order here. Latches propagate values from input to output continuously when enabled (typically by a clock signal) whereas flip-flops propagate values at discrete points in time (typically on the rising or falling edge of a clock). As such, latches are said to be *level-sensitive* or *transparent,* whereas flip-flops are said to be *edge-triggered.* This section will elaborate these concepts and include analysis and design of these circuits using CMOS technologies. The circuits can be implemented directly using the gates described above, although more efficient architectures are available as seen in later chapters.

5.6.1 Basic Bistable Circuit

We begin with a simple bistable circuit and progress to the elements that are used in mainstream chip design. Shown in Figure 5.16 are two cross-coupled logic inverters and a voltage transfer characteristic that is typical of such a circuit. The plotted VTC for Inverter 1, V_{out} versus V_{in}, and Inverter 2, V_{in} versus V_{out}, are shown on the same graph. Note that there are three possible operating points for the circuit: points *A*, *B*, and *C*. *A* and *B* are *stable operating points*, while *C* is an *unstable point* (sometimes referred to as a *metastable* point).

At point *A*, the low voltage level of V_{in} results in a high voltage level at V_{out}, which causes a low level at V_{in}. Similarly, the high voltage level of V_{in} at point *B*

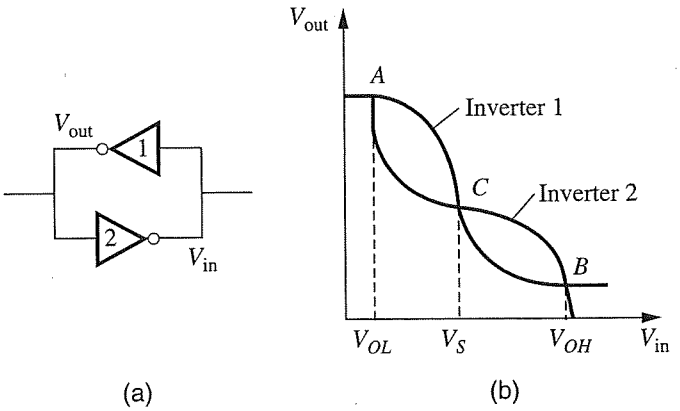


Figure 5.16
Cross-coupled inverters and corresponding VTC.

causes a low level at V_{out} which in turn reinforces the high level of V_{in}. Since these two operating conditions are self-consistent, A and B are considered to be stable points of operation. Alternatively, the gain of the two inverters at the two stable points is less than unity which keeps their outputs stable.

At point C, the slope is positive and greater than unity (in magnitude). Both Inverter 1 and 2 are conducting. As a result, point C is unstable since any small voltage change introduced into the circuit at, say, V_{in} will be amplified and *regenerated* around the circuit loop, causing the operating point to move to one of the two stable points. With V_{in} made positive with respect to V_{out} due to noise, stability is reached at point B. With the polarity of the noise source reversed, stability is realized at point A. Note that the cross-over point is the switching threshold, V_S, a value that was used to determine the single source noise margin. In essence, the cross-coupling of two inverter circuits results in a *bistable* circuit with two stable states. For stability it is necessary that the voltage gain around the loop be less than 1; otherwise, the positive gain will amplify the change and eventually force the outputs to switch to the opposite state due to the regenerative effect.

For a bistable circuit to change state it is necessary that the voltage gain of the circuit be made greater than 1. This can be accomplished by introducing a *trigger voltage pulse* at V_{in}. With the input of Inverter 1 in the low state, a positive trigger pulse is required at V_{in}. The voltage amplitude of the pulse should be sufficient to raise the voltage past V_S where the gain of the circuit is greater than unity; then, due to the cross-coupled circuit and resulting positive feedback, the trigger pulse will be regenerated and a change to the opposite state will be initiated.

The width of the trigger pulse need be only a little greater than the total propagation delay time around the circuit loop. The average propagation delay for a single gate is given by

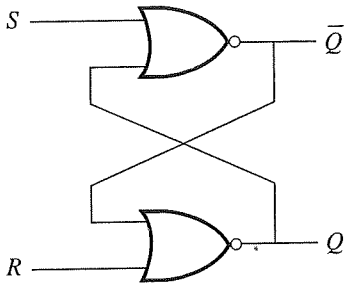
$$t_P = \frac{t_{PHL} + t_{PLH}}{2} \tag{5.6}$$

where t_{PHL} is the high-to-low propagation delay, and t_{PLH} is the low-to-high propagation delay. The delay around the inverter loop is $t_{PHL} + t_{PLH}$; that is, twice the average propagation delay time of the logic inverters. So long as the trigger pulse is greater than this value, the circuit will change state.

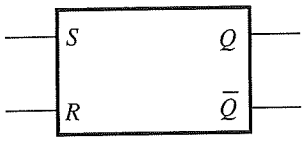
In summary, a bistable circuit may be triggered to move from one stable state to the other by moving an input past V_S for a duration of $2t_P$. In the absence of a trigger, the circuit remains in that stable state indefinitely, provided that power remains applied to the circuit. Another common name for the bistable circuit is a *flip-flop*. However, the name flip-flop is usually reserved for a more complex circuit than a simple cross-coupling of two inverters.

5.6.2 SR Latch

The simplest form of the bistable circuit is the *set-reset (SR) latch*. This circuit latches, or remembers, the incoming trigger pulse. Two implementations of this circuit are described below. It is a useful circuit that forms the basis of the memory circuits, described in Chapter 8.



(a)



(b)

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

(c)

Figure 5.17
SR latch composed of cross-coupled NOR gates.

SR Latch with NOR Gates. In Figure 5.17a, the SR latch is implemented with two NOR2 gates. One of the inputs of each NOR gate is used to cross-couple to the output of the other NOR gate, while the second input provides a means of triggering the latch from one stable state to the other. This circuit is considered to be a latch since the output is a continuous function of the input values (i.e., it is transparent). The logic symbol for the latch is shown in Figure 5.17b. The two outputs Q and $\overline{Q}$ are complementary and, by definition, the latch is in the *set state* with a logic 1 at the Q output and logic 0 at the $\overline{Q}$ output. Conversely, with a logic 0 at the Q output and logic 1 at the $\overline{Q}$ output, the latch is in the *reset state.*

From Figure 5.17a, it can be seen that a logic 1 at the set (S) input will cause a logic 0 at the $\overline{Q}$ output and, with the reset (R) input at logic 0, it will result in a logic 1 at the Q output. That is, the latch is now set, and the S input can safely be returned to the 0 state. The trigger pulse at the S input of sufficient duration causes the latch to be set. Alternatively, a logic 1 at the R input will cause a logic 0 at the Q output and, with the S input now at a logic 0, the result is a logic 1 at the $\overline{Q}$ output. Since the latch responds to high voltage levels at the inputs, S and R are referred to as *active high inputs.* When the inputs are both returned to the 0 state, the flip-flop behaves as a cross-coupled pair of inverters, as shown in Figure 5.16a, that hold the value until further changes are experienced at the inputs.

The input/output results are provided in the *truth table* for the SR latch in Figure 5.17c. The truth table lists the output states for all the possible combinations of the input states. The first entry in the table is the hold state since, with low voltage levels at both S and R inputs, there is no change in the Q and $\overline{Q}$ outputs. The second and third entries are the reset and set states, respectively. In the fourth entry with high levels at both S and R inputs, the result is that both Q and $\overline{Q}$ are low. Since the outputs are not complementary, a logic value of 1 at both S and R inputs is considered to be a *forbidden,* or *not-allowed,* condition. It is only considered forbidden in the sense that the outputs are not complements of each other. Actually what happens is that when the input trigger pulses are returned to their quiescent levels, the final state of the latch is due to whichever input is last to go low.

The delay from S to Q can be estimated as two NOR gate delays, while the delay from S to $\overline{Q}$ is only one NOR delay. Similarly, R to Q is one gate delay while R to $\overline{Q}$ is two gate delays.

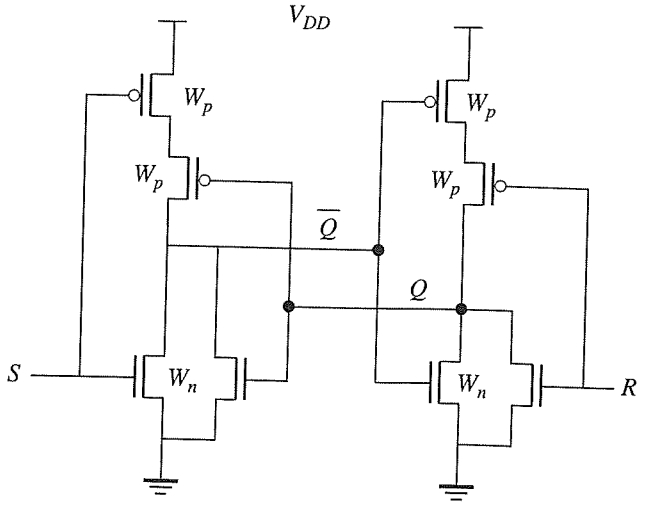
Example 5.4 **SR Latch Design Using NOR Gates**

Problem:

Design an SR latch in 0.13 μm CMOS using NOR gates to deliver a delay of 400 ps from S to Q and from R to $\overline{Q}$. Assume that the total load to be driven by Q and $\overline{Q}$ is 100 fF, and $L = 100$ nm.

Solution:

The following cross-coupled NOR configuration is used to implement the SR latch.



To obtain the desired propagation delays, we use the relationship $t_{PHL} = t_{PLH} = 0.7R_{eff}C_L$. There are two propagation delays from S to Q and R to $\overline{Q}$, so we divide the

two delays equally between the two NOR gates. For the NMOS devices, we know that $R_{eqn} = 12.5\ \text{k}\Omega/\square$. Therefore,

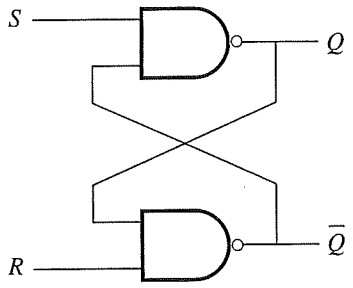
$$t_{PHL} = 0.7(12.5\ \text{k}\Omega)\frac{L}{W}(100\ \text{fF}) = 200\ \text{ps}$$

$$\therefore \frac{W}{L} = 4.4$$

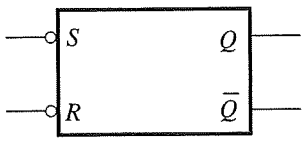
Set $W_n = 440$ nm for both NMOS devices.

For PMOS devices, we use our rule-of-thumb and simply quadruple the size of the NMOS devices to obtain $W_p = 1.76\,\mu$m.

SR Latch with NAND Gates. The SR latch can also be designed with NAND gates, as in Figure 5.18a. Similar to the NOR circuit, the reset condition of the latch is with $Q = 0$ and $\overline{Q} = 1$. However, with the NAND circuit, the S and R inputs are normally at logic 1. The latch is set with $S = 0$, and reset with $R = 0$. Hence this latch responds to *active low inputs*, as is indicated by the small circles at the S and R inputs of the



(a)



(b)

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

(c)

Figure 5.18
SR latch with NAND gates.

logic symbol shown in Figure 5.18b. It is also reflected in the characteristic table of Figure 5.18c. For the NAND configuration of the SR latch, the forbidden condition requires $S = R = 0$ since this leads to $Q = \overline{Q} = 1$.

The delay from S to Q can be estimated as one NAND gate delay, while the delay from S to $\overline{Q}$ is two NAND delays. Similarly, R to Q is two gate delays while R to $\overline{Q}$ is only one gate delay. The design of this circuit follows the design of the NAND gates described earlier. However, the problem of the forbidden state prevents this type of memory element from frequent use, as in the case of the NOR latch.

Exercise 5.4 Draw the transistor schematics for NAND-based SR latches using CMOS gates and pseudo-NMOS gates. Size the devices to deliver 400 ps propagation delays when driving 100 fF loads in 0.13 μm technology. Use the same solution process as shown for the NOR gates in Example 5.4.

5.6.3 JK Flip-Flop

By the addition of two feedback lines the ambiguity at the output of the SR latch, when both S and R inputs are activated at the same time, can be overcome. The device is known as a *JK flip-flop*. The S and R inputs are renamed J and K, respectively. An all-NAND version of this circuit is shown in Figure 5.19a. An important addition is a *clock* (CK) input, provided so that the change in the output logic states of the flip-flop can be synchronized with a system clock. Hence, the J and K inputs are termed the *synchronous inputs; J* is the *clocked-set* input and K the *clocked-reset* input. Note from the logic symbol for this circuit given in Figure 5.19b that all three inputs are activated by high voltage levels.

The JK flip-flop is set and reset in a similar manner to the SR latch, except for the synchronizing with CK, by way of the two NAND gates at the input of the flip-flop. With the flip-flop reset, $Q = 0$ and $\overline{Q} = 1$, and entry is only by way of the J input. A value of $J = 1$ would cause the flip-flop to set, making the Q output high and $\overline{Q}$ low. This now enables the K input; if $K = 1$ and if CK is still high, the flip-flop will again change state. This is the flip-flop action from which it derives its name.[10] Note that if the clock continues to stay high with $J = K = 1$, the flip-flop will oscillate. Hence for this version of the JK flip-flop there is a very definite restriction on the pulse width of CK. It must be less than the propagation delay time through the flip-flop.

The characteristic table of the JK flip-flop is given in Figure 5.19c. Since the two outputs are always complementary, only the state of the output following the $n + 1$st clock is given, indicated as Q_{n+1}. With $J = K = 0$, there is no change in the state of the flip-flop after clocking, indicated as Q_n. But with $J = K = 1$, the flip-flop changes state when clocked, indicated by $\overline{Q}_n$. A characteristic of all JK flip-flops is that with both J and K at a high level the unit will *toggle* when clocked; that is, it will

[10] Technically, the outputs are directly controlled by the inputs while the clock is high and therefore it should be referred to as a latch.

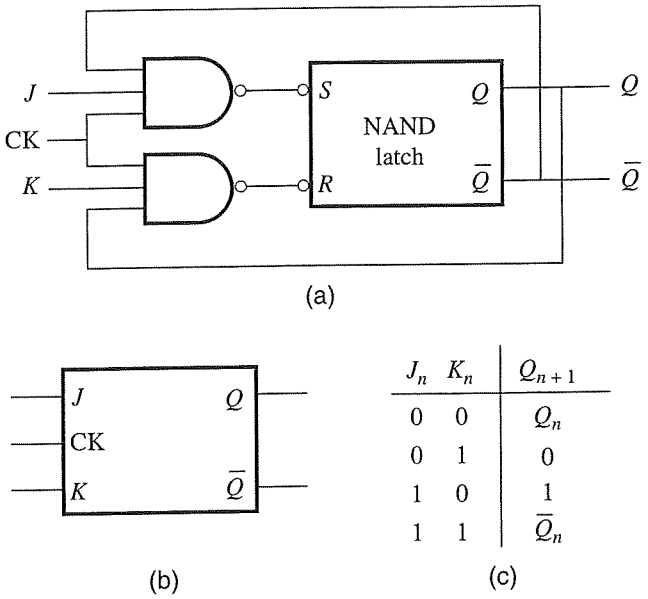


Figure 5.19
JK flip-flop.

change state irrespective of its original state. Again, if the clock is high for a long enough period, the flip-flop output will oscillate in this type of configuration. This undesirable limitation can be eliminated with the master-slave principle, described in the next section.

5.6.4 JK Master-Slave Flip-Flop

As shown in the logic diagram of Figure 5.20, the JK *master-slave* flip-flop is simply a cascade of two JK flip-flops, with the master driving the slave. An important point to notice is that the master is activated by CK, but the slave is activated by an inverted form of CK, namely, $\overline{CK}$. The operation of the master-slave principle is as

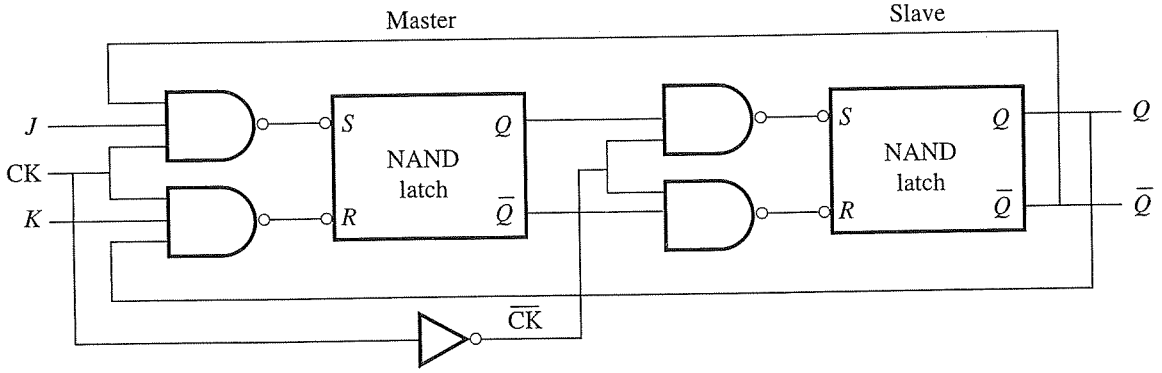


Figure 5.20
Master-slave flip-flop using JK flip-flop.

follows. As CK rises, $\overline{CK}$ goes down; at some point, $\overline{CK}$ has fallen sufficiently to disable the input NAND gates of the slave. This isolates the slave from the master and freezes the state of the slave latch. In the meantime, CK rises enough to enable the input NAND gates of the master. Hence depending on the state of the J and K feedback lines, the state of the J or K input can be entered into the master. Now on the falling edge of the clock pulse, CK goes down; the input NAND gates to the master are disabled, freezing the state of the master latch. Then, the NAND gates to the slave are enabled and the state of the master is transferred to the slave. The outputs Q and $\overline{Q}$ are obtained from the slave latch. Due to this master-slave principle, there is no limit to how wide CK can be since, when the output changes state, entry into the master has been disabled. Only on the next clock pulse will the master input gates be enabled, and then not until a time corresponding to the slave being disabled. Note that there is a minimum limit to the width of CK. It must be wider than the propagation delay time through the master flip-flop for proper functionality.

A problem with the master-slave flip-flop is that it is subject to what is known as *ones catching*. That is, with CK high, suppose the slave is in the reset state, and the J input gate is enabled for a short while and then returns to 0. This will inadvertently set the first latch. Since this is a level sensitive circuit, the high value on J will propagate into the master latch and lock in place. Thus, any "spike" or "glitch" on the J input line will cause the master latch to be set. It is impossible to reset this latch, since entry into the K input gates is disabled by the K feedback line. Thus, we say the J input has *caught a 1* that will subsequently be transferred to the slave when CK goes down. One way to prevent ones catching is to keep the duration of the 1 state for CK as short as possible. Another way is to make use of JK *edge-triggered* flip-flops, which is the preferred approach.

Exercise 5.5

Draw a timing diagram that illustrates the 1's catching drawback of the JK master-slave flop of Figure 5.20. Start by placing the JK in the reset state and apply a periodic clock signal. When the clock is high, introduce a short pulse on the J input. Draw the corresponding logic waveforms on the outputs of the NAND latches. When does the undesired 1 appear at the Q output?

5.6.5 JK Edge-Triggered Flip-Flop

The logic diagram of a JK edge-triggered flip-flop is shown in Figure 5.21a. With CK at a high level, entry into the input NAND gates is controlled by the JK feedback lines in a similar manner as for the master-slave flip-flop. But entry into the NAND latch is prevented until CK goes low.

When CK does go low, the input NAND gates are disabled; the output of the flip-flop changes due to the state of the J and/or K inputs just prior to CK going low. For proper functionality, the values on the J and K inputs must be held stable for a finite length of time before the clock edge. This is the *set-up time* of the flip-flop. The circuit is forgiving of any spikes or glitches on the J or K lines prior to this time as long as the set-up time is satisfied. Some flip-flop designs require the state of the

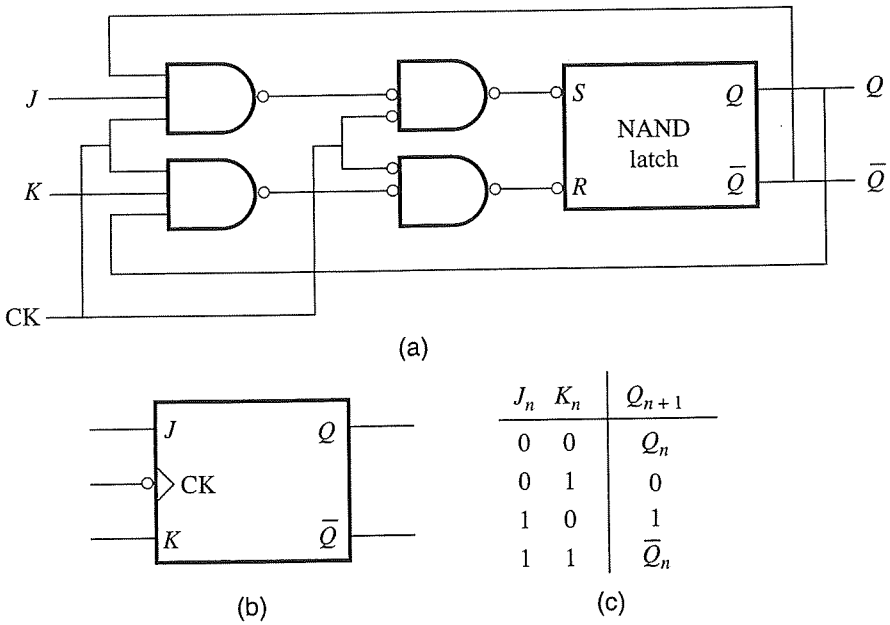


Figure 5.21
JK negative edge-triggered flip-flop.

J and K input lines to remain stable for a time even after the clock edge arrives. This is known as the *hold time*. Usually the minimum hold time is 0 ns.

As shown in Figure 5.21b, the logic symbol for an edge-triggered flip-flop is differentiated from the master-slave by a small > sign at the CK input. The presence or absence of the inverting symbol is used to indicate whether the outputs Q and $\overline{Q}$ change on the falling or rising edge of the clock pulse. The logic symbol indicates a negative edge-triggered flip-flop. The corresponding truth table is shown in Figure 5.21c.

In summary of the JK flip-flops, we have noted that the device may be of the master-slave or edge-triggered type. The outputs may change following or on the rising or falling edge of the clock pulse. The device usually has additional inputs for direct set and reset so that the outputs can be initialized to the desired settings.

5.7 D Flip-Flops and Latches

Another very useful flip-flop, most widely used in CMOS digital circuits and systems for the storage of data, is the *D flip-flop*. The truth table of a D-type flip-flop is shown in Figure 5.22a. With clocking, the Q output simply follows the D input. The $\overline{Q}$ output is always complementary to the Q output.

The D-flop may be of two types. It may be *transparent*, as shown in the logic diagram (Figure 5.22b). The CK input is really an enable input that, when high, allows the Q output to follow the D input. The state of the element is frozen only when CK goes low. Or, it may be *edge-triggered*, as in Figure 5.22c. Similar to other edge-triggered flops, with CK either high or low the D input has no effect on the

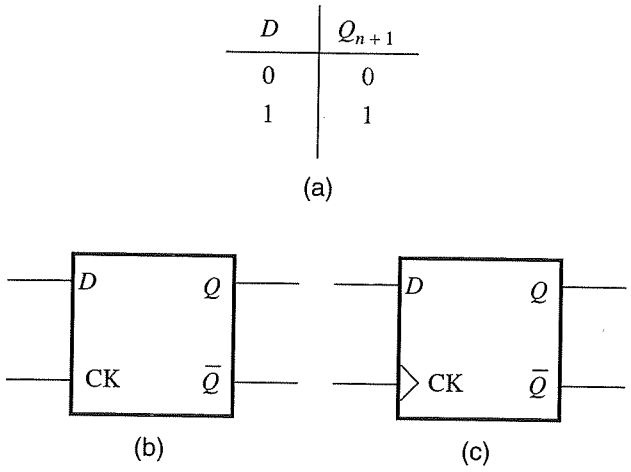


Figure 5.22
(a) Truth table, (b) transparent D-latch, and (c) edge-triggered D-flop.

output. Data are transferred from the *D* input to the output only on the rising edge of the clock pulse.[11] At this point, it should be emphasized that the working definition of a latch is an element that is level-sensitive and transparent, whereas a flip-flop is edge-triggered and otherwise opaque.[12]

To further describe the difference between the two types of sequential elements, it is useful to examine the outputs, given the same input and clock signal. Consider Figure 5.23 where the D-latch is driven by IN and CLK. The output is a continuous function of the input while the clock is high. That is, the D-latch allows the input to propagate through the latch to the output when CLK is high and prevents its propagation when low. Initially, the output is unknown. When the clock goes high, the low input propagates to the output. When the input data goes high, it appears at the output after a short delay. When the clock goes low, the last value of the input is held. Note that the falling edge of CLK is the important one since this is when the data is latched at the output. Further changes in the input are not propagated to the output. Therefore, the small negative pulse in the input of Figure 5.23 is not seen at the output.

Now consider the D-flop in Figure 5.24. In this case, it is constructed using two D-latches in a master-slave configuration driven by CLK. For the positive edge-triggered D-flop, the important edge is the leading edge of the clock waveform. The leading edge captures and propagates the last value at the input just prior to the clock transition; the final state of the flip-flop is due to the state of the *D* input just before clocking occurs. In the waveform shown, there are two points sampled: in one case, the input is low and in the other case the input is high. As a result, we see a transition at the output corresponding to the second rising clock edge. The actions

[11] Since there is no bubble on the CK input, this is a positive edge-triggered flip-flop; a bubble would indicate a negative edge-triggered flop.
[12] Designers tend to use the terms flip-flop and latch interchangeably which is some cause for confusion. We will strictly use latch and flip-flop definitions from this point onward in this text to avoid such confusion.

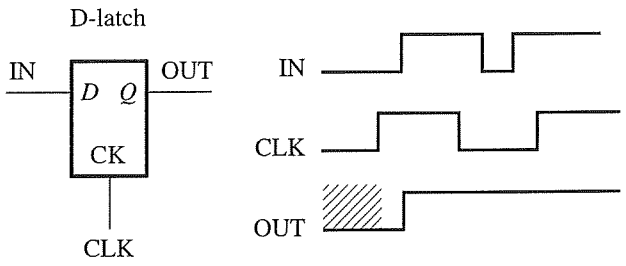


Figure 5.23
D-latch operation.

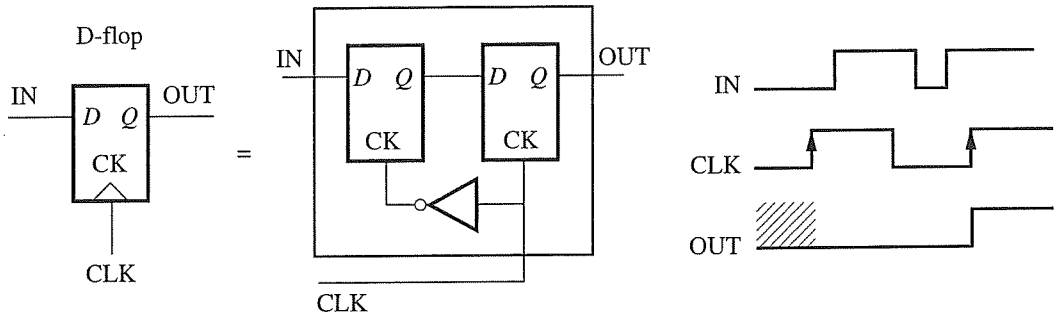


Figure 5.24
D-flop operation.

are equivalent to a sample-and-hold circuit, where the sampling is done when the clock edge hits the flop.

There are a number of important timing characteristics associated with flops and latches. The timing parameters for the flip-flop are illustrated in Figure 5.25. The clock edge is the reference point for these timing parameters. The setup time, T_{setup}, is the time that the incoming data must be stable before the clock arrives. The hold time, T_{hold}, is the length of time that the data remains stable after the clock arrives for proper operation. As shown in the figure, if the data is stable before the setup time and continues to be stable after the hold time, the flop will properly capture the data. However, if it settles after the hold time it is guaranteed to fail. A failure implies storage of the wrong data.

If the data arrives within the period designated by the setup and hold times, the flop may or may not capture the correct value. This is the uncertainty interval. However, the user of such a flop should not permit signals to stabilize in this range since proper performance is not guaranteed. Another parameter of importance is the clock-to-Q delay, $T_{clk\text{-}q}$. This is the delay from the time that the clock arrives to the point at which the Q output stabilizes. In reality, the incoming data must arrive at T_{setup} before the clock hits, and the output is valid at $T_{clk\text{-}q}$ after the clock edge. Viewed in this way, the "overhead" of the flop is $T_{setup} + T_{clk\text{-}q}$. When designing flops, we should try to minimize T_{setup}, T_{hold}, and $T_{clk\text{-}q}$.

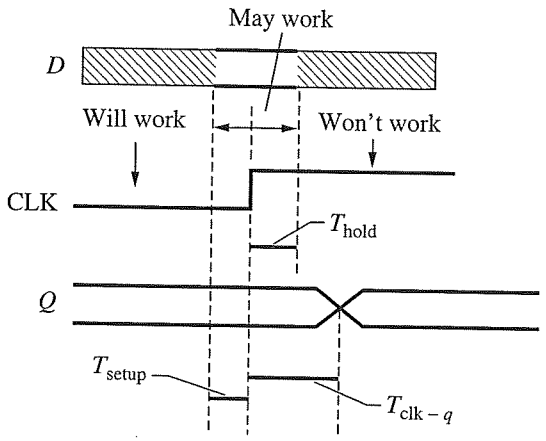


Figure 5.25
Flip-flop timing parameters.

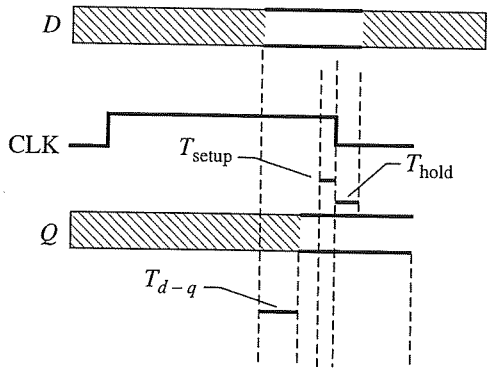


Figure 5.26
Latch timing parameters.

A similar set of parameters applies to the D-latch as shown in Figure 5.26. The interesting edge of the clock is the falling edge, so the corresponding setup and hold times are associated with this edge. However, data begins to flow after the positive going edge of the clock so the setup and hold times are not as critical, and are typically much smaller for the D-latch. The more important parameter is the "overhead" associated with the latch, which is referred to as the D-to-Q delay, $T_{d\text{-}q}$. This is the amount by which the data is delayed by the presence of the latch. Since the latch is transparent when the clock is high, we are more interested in reducing the delay between input and output as opposed to the setup and hold times. However, if there is a late arriving signal near the falling edge, it must satisfy the setup and hold constraints or the circuit will fail.

We now illustrate how a D-latch can be constructed using conventional CMOS logic. A logic diagram for the D-latch is given in Figure 5.27. This is suitable for implementation using AOI gates as shown in Figure 5.28. An edge-triggered D-type flip-flop may be constructed from two such transparent latches in cascade driven by complementary clocks. The number of transistors needed to realize a D-flop in this

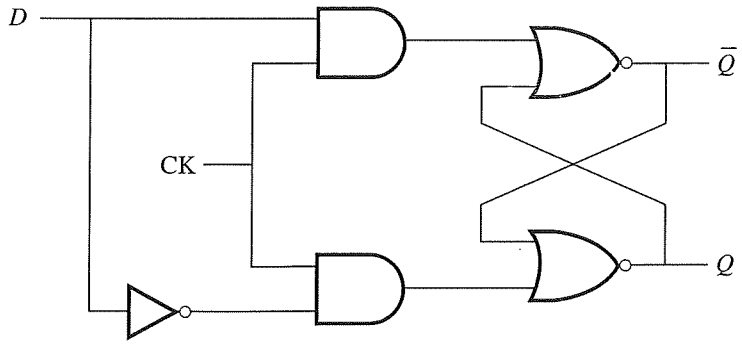


Figure 5.27
Gate level realization of D-latch.

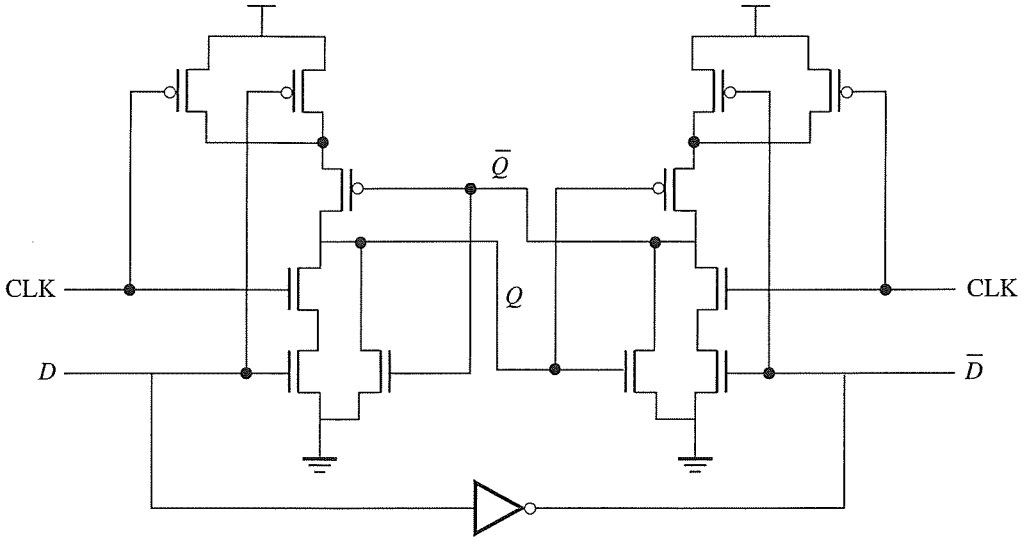


Figure 5.28
AOI implementation of D-latch.

manner is large. More efficient circuits for D-latches and D-flip-flops will be presented in Chapter 7.

5.8 Power Dissipation in CMOS Gates

Power consumption in CMOS digital designs has been increasing at an alarming rate over the past few technology generations. It has become perhaps the most important design specification in recent years since it affects power grid design, chip temperature, packaging decisions, and long-term reliability. If the trend for chip power continues, it will begin to dictate the logic/memory composition of future designs and force most of the chip to be dominated by memory, which consumes less power than logic circuits. In this section, we examine the sources of power dissipation in CMOS gates and describe circuit techniques to minimize power in CMOS designs. As we describe these techniques, take note of the role of timing in

controlling power. To minimize power, we often adjust the timing characteristics of a design. In effect, the tradeoff between power and timing is a key issue in CMOS digital design.

Power is due to current flowing from the supply to ground. When computing power, we must add up all the sources of current flow from V_{DD} to Gnd and multiply it by the potential difference between the two supply rails. The general power equation is given by

$$P = I_D V_{DD} \tag{5.7}$$

where I_D is the current flowing from V_{DD} to Gnd.

In CMOS, the sources of power can be broadly categorized into two groups:

1. Dynamic Power
2. Static Power

Dynamic power arises from three sources: power due to capacitance switching, short-circuit power due to "crowbar" current flowing from V_{DD} to Gnd during switching, and power due to glitches in the output waveforms. Static power is due to leakage currents (subthreshold current and source/drain junction reverse-bias current) and dc standby current (e.g., pseudo-NMOS circuits with low output). The two most important sources of power dissipation today are capacitance switching and subthreshold leakage. Static dc power dissipation is of primary concern only when using pseudo-NMOS gates. We now describe these components in detail.

5.8.1 Dynamic (Switching) Power

Most of the chip power today is due to the charging and discharging of capacitances in the circuit as a result of logic switching events. When switching events occur, the supply current acts to charge the output load capacitance on one part of the cycle, and discharge the capacitance on the other half of the cycle. Effectively, we have current flowing from V_{DD} to Gnd (albeit on different parts of the cycle) and this leads to power dissipation. The frequency of switching, f, determines the actual power that is consumed.

To compute the average power dissipation due to switching, we need to determine I_D and multiply it with V_{DD}. The considerations are detailed for the CMOS inverter of Figure 5.29a. When the input switches from high to low, the PMOS device turns on, as shown in Figure 5.29b. It charges up the output to V_{DD}, thereby storing energy on the capacitor. When the input switches from low to high, the NMOS device turns on and discharges the capacitor as illustrated in Figure 5.29c. The average charging current is

$$I_{D,\text{avg}} = C\frac{dV}{dt} = \frac{C_L \Delta V_{\text{swing}}}{\Delta t} = C_L V_{DD} f_{\text{avg}} \tag{5.8}$$

Note that in the above equation, we assume that the average switching frequency of the inverter is f_{avg} and the voltage swing is V_{DD}.

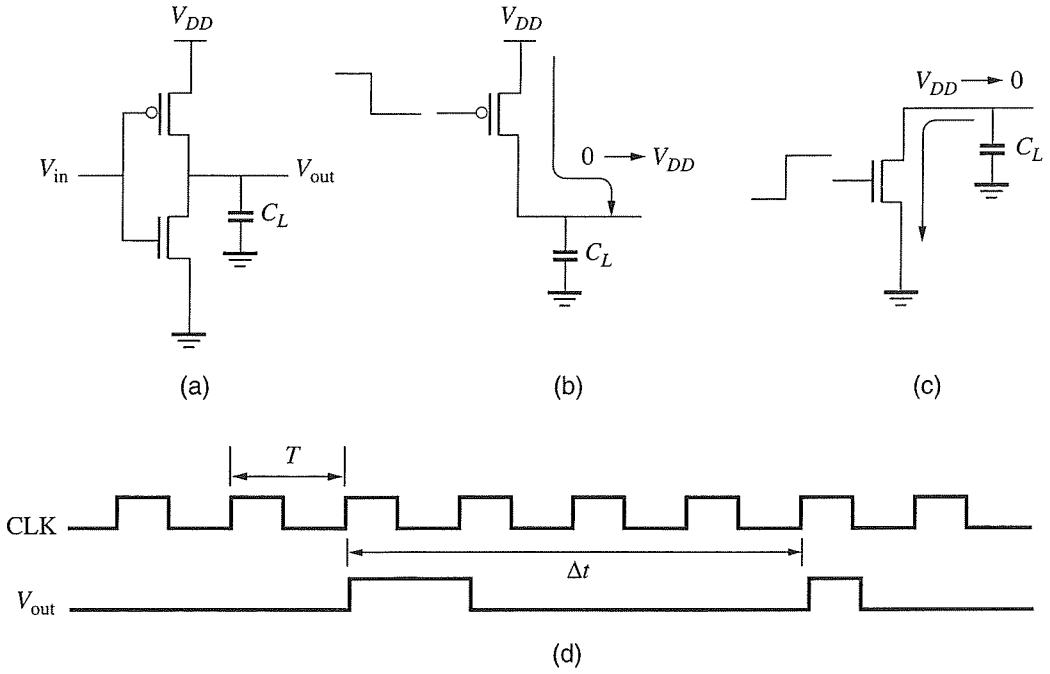


Figure 5.29
Dynamic power dissipation considerations.

The combination of these two processes results in power dissipation. Since the current is associated with a capacitor, we can write

$$P_{\text{switching}} = I_{D,\text{avg}} V_{DD} = C_L \Delta V_{\text{swing}} f_{\text{avg}} V_{DD} = C_L V_{DD}^2 f_{\text{avg}} \tag{5.9}$$

This equation tells us exactly what our options are to reduce power: keep C_L small, reduce the swing, reduce V_{DD}, or reduce the switching frequency, f_{avg}. The approach used to reduce power will be dependent on the design style used, timing constraints, area constraints, and other tradeoffs. Usually power reduction requires a combination of many methods.

One factor that we should examine more closely is the average switching frequency of the inverter, f_{avg}. The clock frequency is normally taken to be f_{clk} and clearly the clock switches on every cycle.[13] If we refer to a transition from high to low or low to high as a *toggle*, then it follows that we need two toggles to have power dissipation. On the other hand, most logic gates do not switch on every cycle as shown in Figure 5.29d. The average frequency of operation can be specified using an activity factor, $\alpha_{0\rightarrow 1}$, that is multiplied by the clock frequency, f_{clk}. The power equation can be modified to include this activity factor as follows:

$$P_{\text{switching}} = \alpha_{0\rightarrow 1} C_L V_{DD}^2 f_{\text{clk}} \tag{5.10}$$

[13] In fact, the clock circuit may consume up to 40% of the chip power due to its large capacitance and high frequency.

With this power equation and knowledge of the activity factor for each gate, we can accurately estimate the chip power. While $\alpha_{0\to1}$ should be computed for each gate separately, we can also compute an average activity factor and capacitance for the whole chip and use these values to estimate the average chip power due to switching.

Example 5.5 **Activity Factor Calculation**

Problem:

In Figure 5.29d, we have a total of four toggles of the output over the duration of eight clock cycles. What is the activity factor for this node?

Solution:

Number of clock cycles = 8
Number of toggles at output = 4. Two toggles are required for power dissipation.

$$\alpha_{0\to1} = \frac{\#\text{ toggles}/2}{\#\text{ clock cycles}} = \text{activity factor (switching factor)} = \frac{4/2}{8} = 25\%$$

Example 5.6 **Comparison of Hand Calculation of Power with SPICE**

Compute the power due to switching for an inverter with $W_p = 800$ nm and $W_n = 400$ nm driving a total load of 50 fF with an average switching frequency of 250 MHz. Compare the results with a transient simulation in SPICE. Which value is larger, the one obtained by hand calculation or by SPICE? Use 0.18 μm technology parameters for the analysis with $V_{DD} = 1.8$ V. Ignore parasitic capacitances and set input rise/fall times to 300 ps in SPICE.

Solution:

The dynamic power can be computed by hand as

$$P = CV^2f = (50\text{ fF})(1.8)^2(250\text{ MHz}) = 40.5\ \mu\text{W}$$

A SPICE simulation can be performed to confirm this result. A transient analysis is required, followed by an average power measurement over a 4 ns period. Note that the inverter must toggle high and low in order to create a "power event." The input file for SPICE is as follows:

```
*Power dissipated by inverter
*Set supply and library
.param Supply=1.8              *for 0.18 technology
.lib    'bsim3v3.cmosp18.lib'
.opt scale=0.1u                *Set lambda
```

```
* Power supply
.global        Vdd    Gnd
Vdd            Vdd    Gnd      'Supply' *Supply is set by .lib call
*Subcircuit
.subckt inv    in     out
mp      out    in     Vdd      Vdd    PMOS1 l=2 w=8 ad=0 pd=0 as=0 ps=0
mn      out    in     Gnd      Gnd    NMOS1 l=2 w=4 ad=0 pd=0 as=0 ps=0
Cload   out    Gnd    50fF
.ends
* Top level simulation netlist
Xinv    in     1      inv
Vin     in     Gnd    pulse    0      'Supply' 1ns 200ps 200ps 1.7ns 4ns
* Simulation
.tran          50ps   5ns
.measure       tran   avgpwr   avg    P(Xinv) from=1n to=5n
.end
```

From this simulation, the average power dissipated by the inverter during the interval is found to be 46.3 μW. This is approximately 14% higher than the hand calculation.

As seen in the above example, if we run SPICE on a single inverter and measure power, we will find that the measured value exceeds the hand-calculated value. That is, $P_{\text{SPICE}} > CV_{DD}^2 f$. Why would this be the case? The primary reason is due to short-circuit current or crowbar current (although a very small percentage of it is due to leakage current). The origin of the crowbar current, I_{SC}, is illustrated in Figure 5.30.

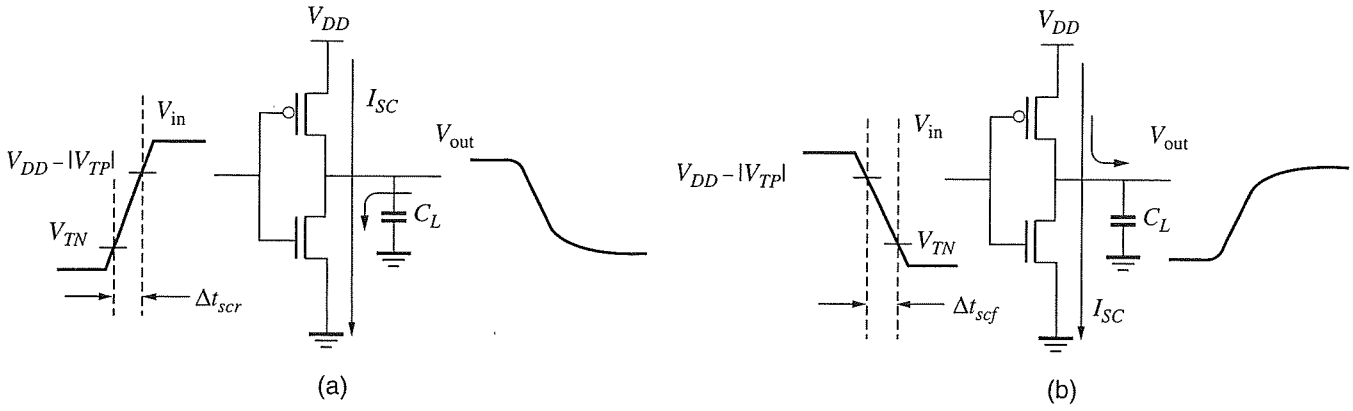


Figure 5.30
Short-circuit current flow during switching process.

Crowbar current is the current that flows directly from V_{DD} to Gnd during switching events. The reason why short-circuit flows is that for a certain period of time both transistors are *on* simultaneously; that is, $|V_{GS}| > |V_T|$ for both devices. If we apply a step input, only one device would be *on* at any given point in time, and we would not observe any short-circuit current. However, since all inputs have a finite slope, both devices are *on* when $V_{TN} < V_{\text{in}} < V_{DD} - |V_{TP}|$. Crowbar current flows in both charging and discharging conditions. The time period that short-circuit current flows depends on the rise/fall time of the input:

$$\Delta t_{sc} = \Delta t_{scr} + \Delta t_{scf}$$

where Δt_{scr} and Δt_{scf} are illustrated in Figure 5.30. When discharging, the sum of the supply current and the discharging current flows through the pull-down device. This is illustrated in Figure 5.30a. During the charging process, part of the supply current flows to the capacitor while the remainder flows to Gnd as shown in Figure 5.30b. The average power associated with this current is

$$P_{SC} = I_{SC} V_{DD}$$

Since crowbar current flows during all switching events, it is possible to rewrite the equation as follows. Taking $I_{SC,\text{avg}}$ as the average crowbar current during the two switching intervals, then

$$I_{SC} = \frac{\Delta t_{sc}}{T} I_{SC,\text{avg}}$$

This can be substituted into the previous equation to obtain

$$P_{SC} = \frac{\Delta t_{sc} I_{SC,\text{avg}}}{T} V_{DD} = \Delta t_{sc} I_{SC,\text{avg}} V_{DD} f_{\text{clk}} \tag{5.11}$$

It is desirable to replace the first two terms by a term that is similar to the dynamic power expression, as follows:

$$I = C\frac{dV}{dt} \Rightarrow I\Delta t = C\Delta V$$

$$\therefore \Delta t_{sc} I_{SC,\text{avg}} = C_{sc} V_{DD}$$

Substituting into Equation (5.11), we obtain

$$\therefore P_{SC} = C_{sc} V_{DD}^2 f_{\text{clk}}$$

Finally, we introduce another switching activity factor and set $C_{sc} = \alpha_{sc} C_L$. With this adjustment, we have

$$P_{SC} = \alpha_{sc} C_L V_{DD}^2 f_{\text{clk}} \tag{5.12}$$

In effect, we are incorporating all the effects of the short-circuit current into a factor α_{sc} that depends on the device threshold voltages and input rise and fall times. The total dynamic power can be written as

$$P_{\text{dynamic}} = \alpha_{0\to1} C_L V_{DD}^2 f_{\text{clk}} + \alpha_{sc} C_L V_{DD}^2 f_{\text{clk}} = \alpha C_L V_{DD}^2 f_{\text{clk}} \qquad (5.13)$$

The first term accounts for the power due to capacitance switching while the second term accounts for the short-circuit power. The new α incorporates both the activity factor and short-circuit current effects.

To reduce short-circuit power, we should set the rise and fall times, or *edge rates*, to be as short as possible. This will minimize Δt_{sc}, the time that both devices are on. However, this is a local optimization that may not be globally optimal. If we use large drivers at the input to reduce the rise/fall times, this increases the capacitance, thereby increasing the overall dynamic power. In effect, there is a tradeoff between dynamic power of the previous gate and short-circuit power of the next gate. We usually seek to minimize the total power due to the sum of these two components. One solution that has been found to be effective is to make the input and output edge rates sharp and about equal. This tends to minimize the overall power due to the two components of dynamic power. This is one example where, in order to minimize power, we must consider the timing characteristics of the signals. We will find many instances where the timing is an important factor in reducing the overall chip power.

There is one other source of dynamic power due to differences in the arrival times of signals at gate inputs. If a given input signal arrives first and forces the output to switch, and later another input signal arrives and causes the output to switch back to the original value, we have a *glitch* in the output. This type of undesired output pulse causes unnecessary power dissipation, which can be modeled in terms of switching power. It can be a major component of the total power if the designer is not careful. Glitches tend to propagate through the fanout gates and cause unintended transitions in subsequent stages, increasing the power dissipation even further. Depending on the number of glitches in the chip, it can quickly add up to a nonnegligible amount of power.

To reduce glitches, signals should be made to arrive at roughly the same time at all gate inputs. This requires a balancing of the path delays and gate delays in a combinational logic block. Whenever possible, the designer should select logic circuits that minimize the potential for glitches. This may also involve retiming a given circuit or selecting architectures that are glitch free, or exhibit fewer glitches than others. The key, as will be clear in many design guidelines throughout this book, is to keep things as balanced as possible.

Glitch Example Using NOR Gate **Example 5.7**

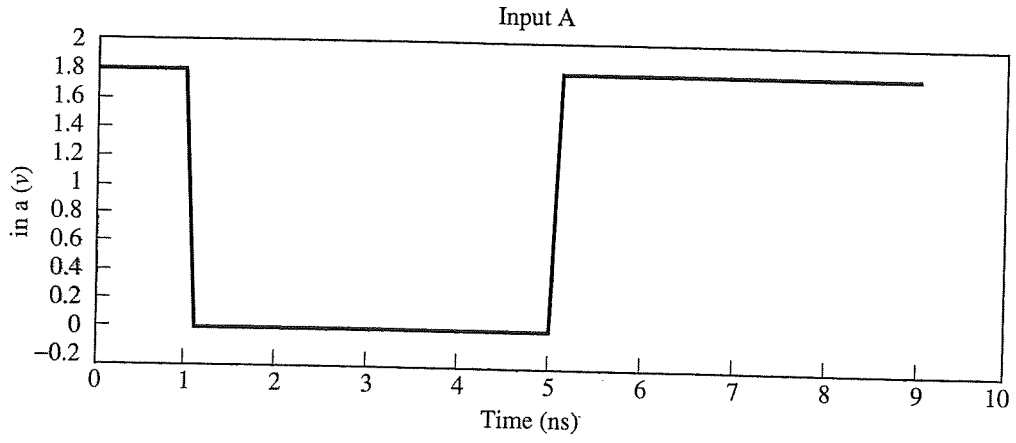
Problem:

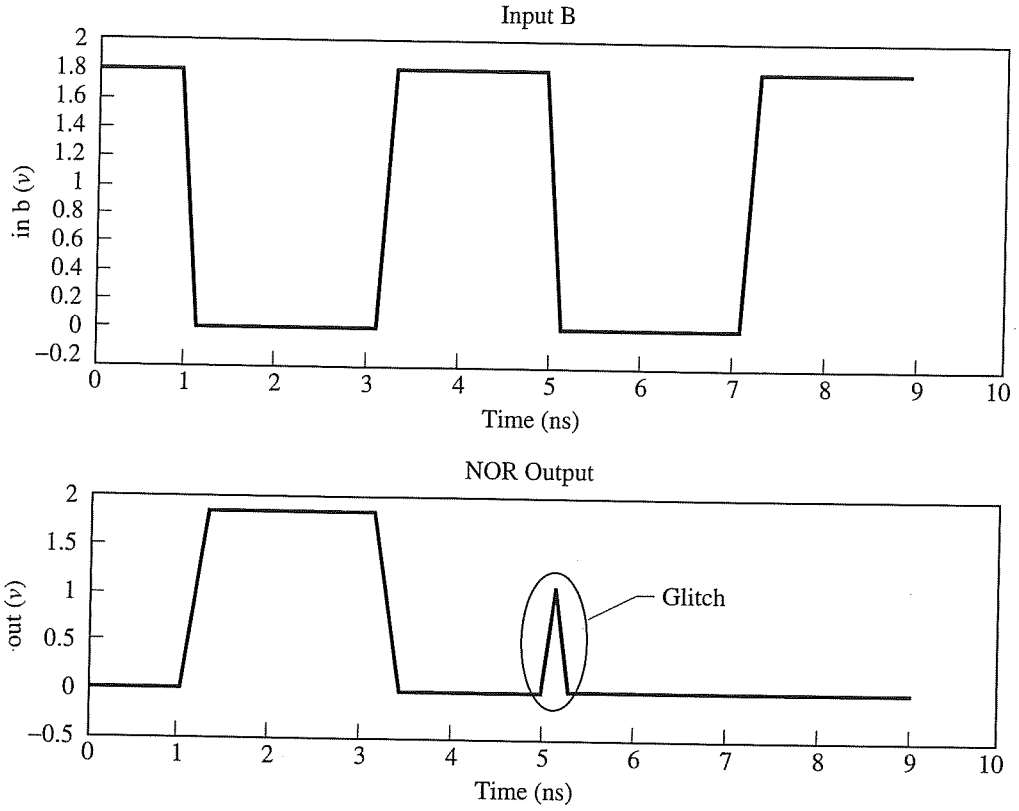
Using SPICE and a 0.18 μm technology, adjust the inputs of a 2-input NOR gate until a glitch appears at the output during a transient analysis. Under what conditions does a glitch occur? How should such a glitch be prevented?

Solution:

Create a SPICE file with a 2-input NOR gate. During the analysis, one input should be switching from low to high while the other is switching from high to low in the same interval to create a glitch. To prevent such a glitch, all the inputs should arrive at the same time.

```
*NOR Gate Glitching Example
*Set supply and library
.param Supply=1.8              *for 0.18 technology
.temp   25                     *Override temperature by setting it before
                                .lib
.lib    'bsim3v3.cmosp18.lib'
.opt scale = 0.1u              *Set lambda
*Power  supply
.global Vdd   Gnd
Vdd     Vdd   Gnd    'Supply' * Supply is set by .lib call
*Top level simulation netlist
mpa     out   ina    x      Vdd    PMOS l=2 w=16 ad=80 pd=16 as=80 ps=16
mpb     x     inb    Vdd    Vdd    PMOS l=2 w=16 ad=80 pd=16 as=80 ps=16
mna     out   ina    Gnd    Gnd    NMOS l=2 w=4 ad=20 pd=4 as=20 ps=4
mnb     out   inb    Gnd    Gnd    NMOS l=2 w=4 ad=20 pd=4 as=20 ps=4
Va      ina   Gnd    pulse  'Supply' 0 1ns 100ps 100ps 4.1ns 8ns
Vb      inb   Gnd    pulse  'Supply' 0 1ns 100ps 100ps 2ns   4ns
*Simulation
.tran   100ps 9ns
.plot   tran  V(out)
.end
```





5.8.2 Static (Standby) Power

Standby power is an important design consideration for low-power battery-operated, or portable, devices. There are three basic sources of static power: subthreshold leakage, *pn* junction leakage, and dc current in the output low state. Today, leakage currents are beginning to control design decisions even for high-performance non-portable applications. As a result, this component of power is gaining more attention in the IC design industry from process engineers, circuit designers, and CAD engineers.

The most important issue is the subthreshold leakage. In recent years, it has grown significantly due to the close proximity of the source and drain. This results in a bipolar transistor action, where the substrate is the base of the bipolar transistor, and the source and drain act as the emitter and collector, respectively. Subthreshold current is due to diffusion current of minority carriers across the channel region. From earlier discussions in Chapter 2, we found that the subthreshold current equation has the form

$$I_{\text{sub}} = I_s e^{[q(V_{GS} - V_T - V_{\text{offset}})/nkT]} \left(1 - e^{(-qV_{DS}/kT)}\right)$$

When we examine this expression from a power perspective, we try to identify the controlling parameters to keep I_{sub} as low as possible. Specifically, if $V_{GS} = 0$, then V_T controls the magnitude of the subthreshold current. As V_T is reduced, the

value of I_{sub} increases (see Figure 2.16). Since V_T has been kept relatively constant over the years, the subthreshold current has been kept in check. However, $V_{DD} - V_T$ controls the forward-active drive capability of the transistor; in effect, we limit the performance of the devices by keeping V_T constant while scaling V_{DD} in each technology generation. Clearly, a tradeoff exists between I_{on} and I_{off} when selecting V_T.

An alternative is to dynamically modify the threshold voltage during operation. We can accomplish this by adjusting the substrate bias of the transistors in the signal path. However, when doing so, we must ensure that the V_T is reset to a higher value when these devices are inactive. This is not a straightforward procedure for complex, high-speed designs.

Another controlling parameter is V_{DS}. If we can reduce this value, when the device is off, it will act to reduce the current. Recall that drain-induced barrier lowering causes increased drain current due to a reduction of V_T. Any approach that can limit V_{DS} will limit the subthreshold conduction. One method is to add series transistors to the pull-up and pull-down paths so that there is a smaller V_{DS} across each transistor. This is sometimes referred to as *source degeneration.*

A third parameter that can be controlled is temperature. If we can cool the transistors down to a very low operating temperature we obtain two benefits. First, since the subthreshold current is due to minority carriers, reducing the temperature will reduce the number of carriers and this will act to reduce the current. When the transistor is *on,* a reduction in temperature will actually increase the drive current since the mobility of the majority carriers will be increased. Of course, the cooling of chips requires a much more expensive package, which will increase the cost of the design.

Another source of leakage is due to the reverse-bias of source and drain junctions. The percentage of the overall power dissipation is rather small but we should still understand its origin. The basic diode equation is given by

$$I_{pn} = I_0 \left(e^{(q/kT)V_{SB}} - 1\right) \qquad \text{where } I_0 = A \cdot J_S$$

In reverse-bias mode, the current is $I_{pn} = -A\,J_s$, where A is the area of the junction and J_s is the current density. The areas of interest are the bottom of the junction and the channel-facing sidewall. Therefore, we need to keep these areas as small as possible by minimizing the source and drain areas. The currents are typically in the *pA* to *nA* range and are negligible in most digital designs. The combination of subthreshold current and the *pn* junction current is referred to as leakage current:

$$I_{leak} = I_{sub} + I_{pn} \tag{5.14}$$

Since the NMOS and PMOS have different leakage currents, I_{leak} can be based on the average leakage of the pull-up and pull-down paths. The total static power dissipation due to these two components is

$$P_{static} = (I_{sub} + I_{pn})\,V_{DD} \tag{5.15}$$

One final form of static power is found in the pseudo-NMOS gates when the output is low. In this case, the current, I_{DC}, through the gate is computed by setting the

output to V_{OL} and determining the current flow through any device in the current path. This current is multiplied by V_{DD} to obtain the power

$$P_{DC} = I_{DC}V_{DD} \tag{5.16}$$

This power can be relatively large. If we design a circuit with exclusively pseudo-NMOS gates, the power budget would be exceeded very quickly, much like the NMOS technology on which it is based. Recall that NMOS technology gave way to CMOS technology due to the dc standby power when the output of a gate was low. Imagine that you have 2 million logic gates and half of them are dissipating power. You quickly realize that you will exceed your power budget no matter what you do since the mere existence of the gates implies power dissipation. Therefore, pseudo-NMOS should be used sparingly.

5.8.3 Complete Power Equation

For a standard CMOS gate, the power equation is given by

$$P = \alpha\, C\, V_{DD}^2 f_{\text{clk}} + I_{\text{leak}} V_{DD} \tag{5.17}$$

where the short-circuit current effects are included in the α term.

For a pseudo-NMOS gate, the complete power equation is given by

$$P = \alpha\, C\, V_{DD}^2 f_{\text{clk}} + I_{\text{leak}} V_{DD} + \frac{I_{DC}V_{DD}}{2} \tag{5.18}$$

where the last term considers the fact that the dc current is flowing only half the time.

DC Power Dissipation for Pseudo-NMOS Inverter in 0.13 μm Technology **Example 5.8**

Problem:

Compute the dc power dissipated by the pseudo-NMOS inverter when the output is low, given that $V_{OL} = 0.1$ V, $W_n/L_n = 9$. Use 0.13 μm technology parameters.

Solution:

The current flowing through the inverter can be computed using the current through the pull-down transistor that is in the linear region of operation:

$$I_{DC} = \frac{W_N}{L_N} \frac{\mu_n C_{ox}}{\left(1 + \dfrac{V_{OL}}{E_{CN}L_N}\right)} \left[(V_{DD} - V_{TN})(V_{OL}) - \frac{(V_{OL})^2}{2} \right]$$

Using the parameter values for 0.13 μm CMOS:

$$I_{DC} = 9 \times \frac{430\ \mu\text{A/V}^2}{(1 + (0.1/0.6))} \left[(1.2 - 0.4)(0.1) - \frac{(0.1)^2}{2} \right] = 250\ \mu\text{A}$$

$$\therefore P = I_{DC}V_{DD} = 250\ \mu\text{A} \times 1.2\ \text{V} = 300\ \mu\text{W}$$

5.9 Power and Delay Tradeoffs

In most designs today, the primary tradeoff is between power and delay. We have already seen that equalizing the input arrival times will reduce glitches and equalizing the edge rates will minimize short-circuit current. Clearly, it is not sufficient to optimize one without considering the other. If we improve a design relative to power but it slows down the circuit, the results may not be acceptable. Similarly, an improvement in speed may only be achieved at the expense of power. Therefore, a metric is needed that allows us to balance the two design objectives in a meaningful way.

One such metric that has been popular for many years is the *power-delay product (PDP)*. The rationale is that, if we are interested in reducing power and reducing delay, then why not take the product of the two and try to minimize them together? At an intuitive level, this makes sense. If we follow this thought process to establish a metric for gates, then:

$$\text{PDP} = P_{\text{avg}} t_P \tag{5.19}$$

where P_{avg} is the average power and t_P is the average propagation delay of a gate. Considering only the dominant source of power for a gate, we have

$$P_{\text{avg}} = CV_{DD}^2 f$$

and the propagation delay is

$$t_P = \frac{1}{2f}$$

assuming that the switching period is twice the propagation delay. Combining these results, we obtain

$$\text{PDP} = CV_{DD}^2 f \frac{1}{2f} = \frac{CV_{DD}^2}{2} \tag{5.20}$$

This is an energy quantity and, as such, the PDP represents the energy required to perform a specific operation. The reason why we view this as an energy quantity is that energy is the time integral of power. If we compute the energy stored in the capacitor after a charging operation, we obtain

$$E_C = \int_0^\infty i_c(t) v_{\text{out}}(t) dt = \int_0^\infty C \frac{dv_{\text{out}}}{dt} v_{\text{out}}(t)(dt) = \int_0^{V_{DD}} C v_{\text{out}}(t) dv_{\text{out}} = \frac{1}{2} CV_{DD}^2$$

In this case, the operation is to switch from low to high. In a similar fashion, Equation (5.20) defines the PDP as the energy per toggle operation for a gate.

The PDP can be minimized by reducing the capacitance C, the voltage swing ΔV_{swing} or the power supply voltage, V_{DD}. However, the metric has some limitations. If the PDP of two designs performing the same operations are compared, it would still be difficult to tell which one is better. For example, if one used a lower supply voltage to reduce PDP, the metric would not capture the fact that the delay is also

reduced. Also, using smaller transistors would reduce C, but may also slow down the circuit. Because of the cancellation of the delay term when deriving the PDP, we have lost valuable timing information.

A solution to this problem is to define another metric that multiplies the PDP by the delay. This is termed the *energy-delay product (EDP)*. It is given as follows:

$$\text{EDP} = \text{PDP} \times t_P \tag{5.21}$$

We already have the equation for PDP, but we need a new formulation for t_P:

$$\begin{aligned} I &= C\frac{dV}{dt} \\ \therefore \Delta t &= \frac{C\Delta V}{I} \\ \therefore t_P &= \frac{C\Delta V}{I_{\text{sat}}} \approx \frac{CV_{DD}}{K_2(V_{DD} - V_T)} \end{aligned} \tag{5.22}$$

where K_2 is a constant that depends on device sizes. Combining Equations (5.20) and (5.22), we obtain:

$$\text{EDP} = \frac{C^2V_{DD}^3}{2K_2(V_{DD} - V_T)} \tag{5.23}$$

Now, whenever power supply values or device sizes are altered, they are captured in the EDP. This is a better metric for low-power design.

If the goal of the overall design is to reduce the EDP, then it is instructive to examine how a number of designs would compare using this metric. For this purpose, we plot the EDP of several designs (*a*, *b*, *c*, and *d*) on a qualitative graph of energy vs. 1/delay in Figure 5.31. The lines shown in the graph have equal EDP and are therefore equivalent in terms of this metric. Designs that lie closer to the *x*-axis

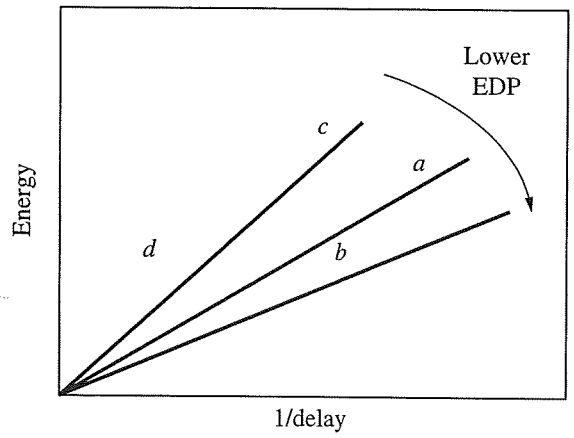


Figure 5.31
Energy versus 1/delay for four designs: *a*, *b*, *c*, *d*.

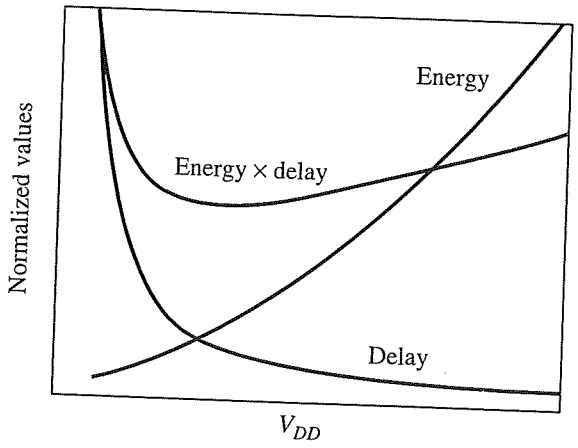


Figure 5.32
Energy-delay product versus supply voltage.

are better since their energy-delay product is lower. Surprisingly, for the four cases shown, design *b* is the best and design *d* is the worst in terms of EDP.

Another way to view this metric is to examine the energy, delay, and EDP on the same normalized graph as a function of the supply voltage, V_{DD}. This is shown in Figure 5.32. As the power supply voltage decreases, delay increases, but the energy per operation decreases. Therefore, an optimum EDP must exist and it can be obtained by differentiating Equation (5.23) with respect to V_{DD} and setting the result to zero:

$$\frac{\partial \text{EDP}}{\partial V_{DD}} = 0$$

$$\therefore V_{DD}^{*} = \frac{3}{2} V_T$$

This optimum value can be used to set the supply voltage for a given technology. It is interesting to note that increasing the supply voltage beyond the optimum does not affect EDP very much. Therefore, higher values of V_{DD} are acceptable and may be necessary for the target performance and noise margins of the design. A decrease in V_{DD} from the optimum rapidly increases the EDP. Therefore, it is important to keep the supply value higher than the optimal value, and perhaps this is the best way to interpret the above results.

Example 5.9 **Comparison of Two Designs**

Problem:

Compare the chip power for two cases. In one case, a chip has 10M gates, an activity factor of 10%, $V_{DD} = 1.8$ V, a clock frequency of 500 MHz and an average capacitance per node of 20 fF. In the second case, a chip has 50M gates, an activity factor

of 5%, $V_{DD} = 1.2$ V, a clock frequency of 1 GHz and an average capacitance per node of 10 fF. Which design is better and why?

Solution:

Case 1: 10M gates, $\alpha = 0.1$, $C = 20$ fF, $V_{DD} = 1.8$ V, $f_{clk} = 500$ MHz $\rightarrow P = 32.4$ W

Case 2: 50M gates, $\alpha = 0.05$, $C = 10$ fF, $V_{DD} = 1.2$ V, $f_{clk} = 1$ GHz $\rightarrow P = 36$ W

The two power levels are about the same, so it appears that the two designs are about equivalent. Now compute the energy-delay product.

Case 1: EDP $= Pt_p^2 = 32.4\ (2\ \text{ns})^2 = 130(10^{-18})J\text{-}s$

Case 2: EDP $= Pt_p^2 = 36\ (1\ \text{ns})^2 = 36(10^{-18})J\text{-}s$

The second design has a smaller energy-delay product and therefore is superior.

5.10 Summary

For a series stack with three devices, the width of a single equivalent device is

$$W_{eq} = \frac{W_1 W_2 W_3}{W_1 W_2 + W_2 W_3 + W_3 W_1} \qquad \text{(series stack)}$$

For a parallel set of devices, the equivalent width with all devices turned on is

$$W_{eq} = W_1 + W_2 + W_3 \qquad \text{(parallel devices)}$$

Average propagation delay

$$t_P = \frac{t_{PHL} + t_{PLH}}{2}$$

Dynamic and static power:

$$P_{\text{dynamic}} = \alpha_{0\rightarrow 1} C_L V_{DD}^2 f + \alpha_{sc} C_L V_{DD}^2 f = \alpha C_L V_{DD}^2 f$$

$$P_{\text{static}} = (I_{\text{sub}} + I_{pn}) V_{DD}$$

For a standard CMOS gate, the complete power equation is given by

$$P = \alpha\ C\ V_{DD}^2 f_{\text{clk}} + I_{\text{leak}} V_{DD}$$

For a pseudo-NMOS gate, the complete power equation is given by

$$P = \alpha\ C\ V_{DD}^2 f_{\text{clk}} + I_{\text{leak}} V_{DD} + \frac{I_{DC} V_{DD}}{2}$$

Energy-delay product:

$$\text{EDP} = \text{PDP} \times t_P = P \times t_P \times t_P$$

REFERENCES

A long list of references exists for this material. Here are a few recent ones.

1. S. M. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits, Analysis and Design*, 3rd ed., McGraw-Hill, 2003.
2. J. Rabaey, *Digital Integrated Circuits: A Designer Perspective*, 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2003.
3. H. Veendrick, *Deep-Submicron CMOS ICs*, 2nd ed., Kluwer Academic Publishers, Boston, MA, 2000.
4. D. D. Gajski, *Principles of Digital Design*, Prentice-Hall, NJ, 1997.
5. J. P. Uyemura, *CMOS Logic Circuit Design*, Kluwer Academic Publishers, Boston, MA, 1999.
6. A. Bellaouar and M. Elmasry, *Low-Power Digital VLSI Design, Circuits and Systems*, Kluwer Academic Publishers, Boston, MA, 1995.
7. A. Chandrakasan, R. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Boston, MA, 1995.
8. T. Burd and R. Brodersen, *Energy Efficient Microprocessor Design*, Kluwer Academic Publishers, Boston, MA, 2002.

PROBLEMS

P5.1. Design static CMOS gates that have the following outputs. You may assume the availability of true and complement forms of each signal for the input. Note that you should be able to design each function as a single gate. Device sizing is not required for this problem.

$$\text{(a) Out} = ABC + BD$$

$$\text{(b) Out} = AB + \overline{A}C + BC$$

$$\text{(c) Out} = \overline{A + B + CD} + A$$

P5.2. Implement the functions associated with a full adder in static CMOS using the fact that these are majority functions. That is, if most of the inputs are high, the output will be high, and if most of the inputs are low, the output will be low. A useful property of majority functions is that they are self-duals. The full adder has three inputs, A, B, and C_{in}, and two outputs, Sum and C_{out}. The expressions for the outputs are as follows:

$$C_{out} = AB + C_{in}(A + B)$$

$$\text{Sum} = \overline{C_{out}}(A + B + C_{in}) + ABC_{in}$$

P5.3. Design a static CMOS gate that performs the Boolean function $F = (A \oplus B)C + BC$. You can use inverters to generate any complementary inputs needed. Sizing is not required here.

P5.4. Find the logic function for the circuit in Figure P5.4. What is the worst case V_{OH} and V_{OL}? (Note: this is a contrived circuit that would not be used in a real design.)

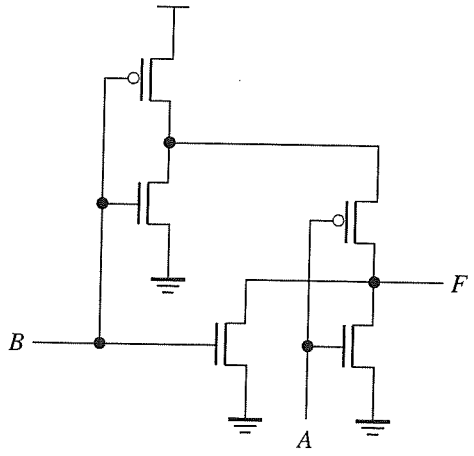


Figure P5.4

P5.5. The circuits of Figure P5.5 are pseudo-NMOS and pseudo-PMOS gates. What function is performed by each gate? Size the transistors so that the pseudo-NMOS gate has $V_{OL} = 0.1V_{DD}$ and the pseudo-PMOS gate has $V_{OH} = 0.9V_{DD}$. Can you suggest why pseudo-PMOS is not favorable (compared to pseudo-NMOS)? All transistors have a length of 2λ.

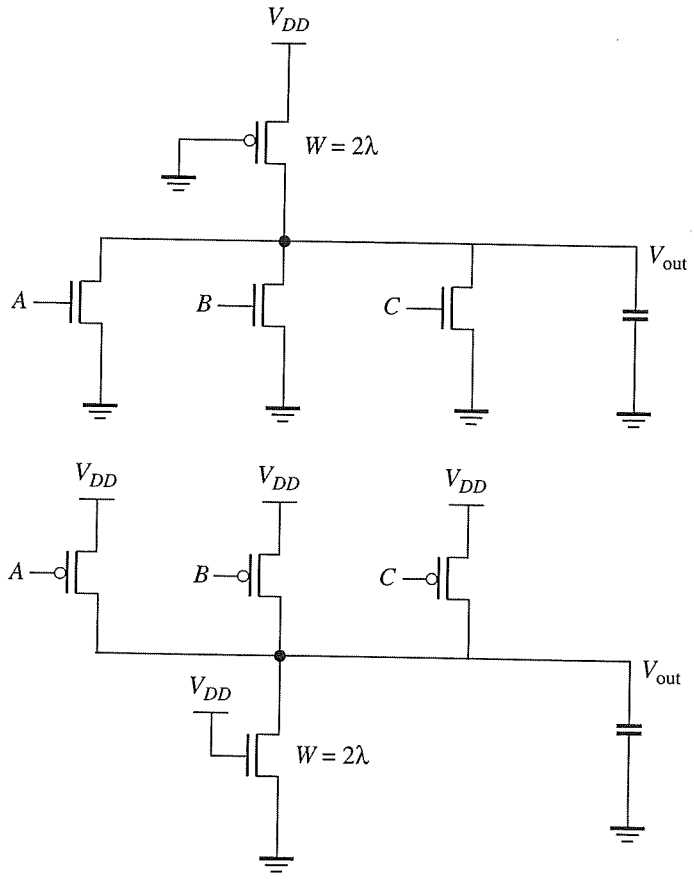


Figure P5.5

P5.6. The circuit of Figure P5.6 is a saturated-enhancement load NMOS NOR gate. Calculate V_{OL} and V_{OH}, and the dc current and power when the output is low. Assume that all the transistors are $2\lambda/2\lambda$ devices. Assume 0.13 μm technology parameters.

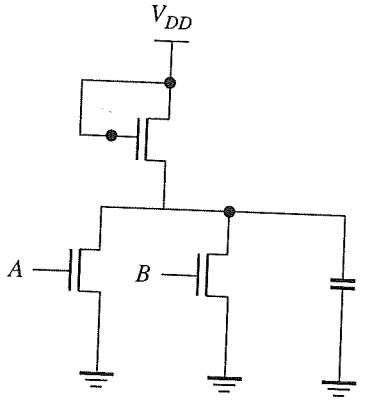


Figure P5.6

P5.7. Calculate V_S of a 2-input NOR gate when one input is switching and with both inputs tied together. The devices sizes are $W_p = 16\lambda$ and $W_n = 4\lambda$. Then use SPICE to find the VTC when switching only input A, only input B, and then AB together. The results for the two inputs switched separately vary slightly. Explain why. Use 0.18 μm technology parameters.

P5.8. What is the minimum value of V_{GG} in the circuit of Figure P5.8 to obtain $V_{OH} = V_{DD} = 1.8$ V? Assume 0.18 μm technology parameters, with $L = 200$ nm.

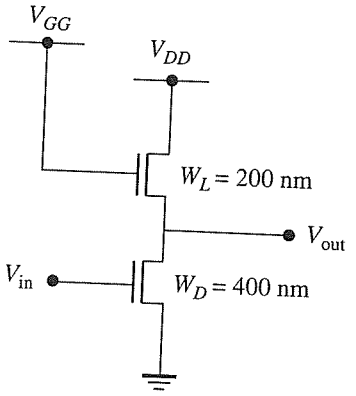


Figure P5.8

P5.9. In the Figure P5.9 circuit, size the transistors so that $t_{PLH} < 50$ ps; $V_{OH} = 1.2$ V and $V_{OL} = 0.1$ V. (The length of all transistors is 2λ.) Ignore parasitic capacitances of the devices. Use 0.13 μm technology parameters.

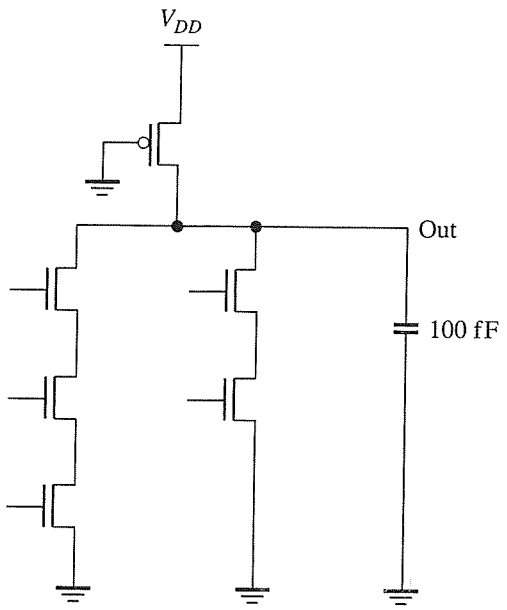


Figure P5.9

P5.10. Design a CMOS gate whose output is Out = $AB + \overline{A}\overline{B}C$. If the minimum length of the transistors is 0.1 μm and the gate is connected to 75 fF load, calculate the width of the transistors so that $t_{PHL} = t_{PLH} < 50$ ps.

P5.11. Calculate the dc static power and dynamic switching power in the circuit in Figure P5.11 if the average switching frequency is $f_{avg} = 100$ MHz. What is the combined average power due to these two components. Use the 0.13 μm technology parameters with $2\lambda = 100$ nm. Ignore device parasitic capacitances.

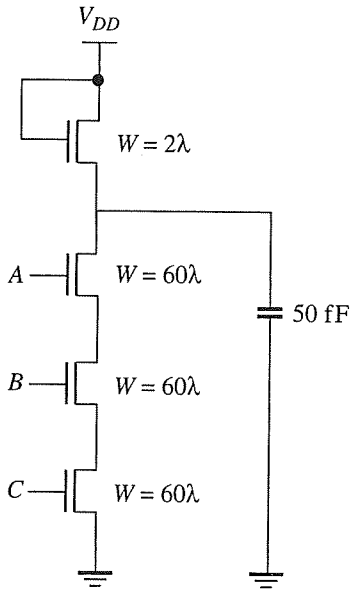


Figure P5.11

P5.12. Compute the static and dynamic power consumption of the two inverters in Figure P5.12. Use the 0.13 μm technology parameters with 2λ = 100 nm.

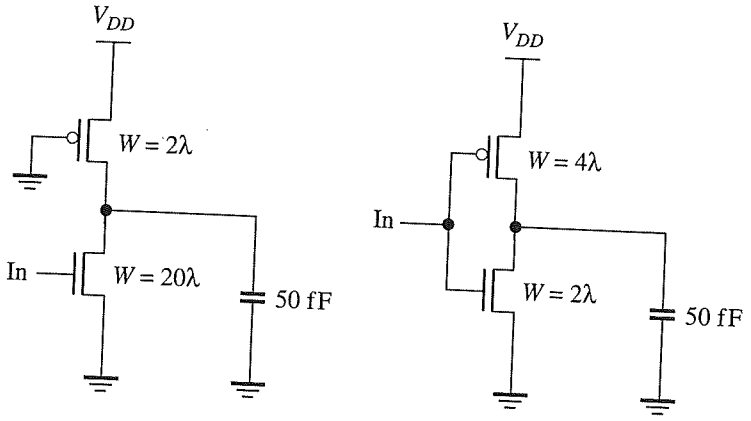


Figure P5.12

Sketch the VTC of the inverters and show on the graph in which region we have power due to dc current, subthreshold current, and short-circuit current.

P5.13. In this question, the idea is to develop a simple equation for the total switching power for 0.18 μm technology that also incorporates the short-circuit power. To do this, use HSPICE to compare the amount of power due to CV^2f versus power due to short-circuit current, $I_{SC}V_{DD}$ in a 4X inverter driving three different load capacitances: 20 fF, 40 fF, and 80 fF. For the same three load capacitances determine the two components of power when input slopes are 0.01 ns, 0.1 ns, and 0.2 ns. Build a table of ratios between CV^2f and $I_{SC}V_{DD}$ as a function of capacitive load and input ramp. What factor α_{sc} would you multiply the CV^2f power by to incorporate the short-circuit power based on this table? (Assume that f= 200 Mhz.)

P5.14. For the RC circuit of Figure P5.14, determine the total energy delivered by the voltage source to the circuit when a step input from 0 to V_{DD} is applied. Next compute the total energy that is stored by the capacitor.

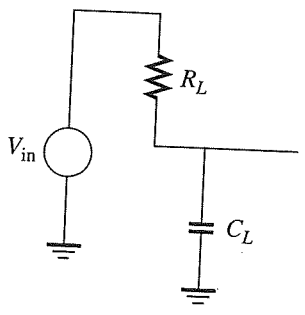


Figure P5.14

What happened to the rest of the energy that was delivered by the voltage source?

P5.15. Do you expect the overall power consumption of a synchronous digital system to increase or decrease when temperature increases (assume the clock frequency does not change)? Explain.

P5.16. Design an SR latch in 0.13 μm CMOS using NAND gates to deliver a delay of 400 ps from S to Q and from R to $\overline{Q}$. Assume that the total load to be driven by Q and $\overline{Q}$ is 100 fF.

P5.17. The voltage waveforms shown in Figure P5.17 are applied to the JK master-slave flip-flop illustrated in Figure 5.20. With the flip-flop initially reset, show the resulting waveform at the Q output of the master and slave latches.

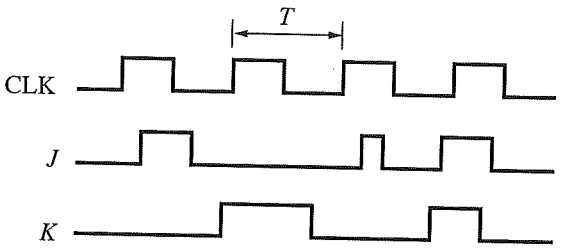


Figure P5.17

P5.18. Repeat the above problem for the JK edge-triggered flip-flop shown in Figure 5.21. Assume the flip-flop is initially set.

P5.19. Implement a positive-edge triggered D flip-flop with set and reset inputs using only NAND3 gates (i.e., no inverters). The inputs should be S, R, D, and CK, while the outputs are Q and $\overline{Q}$. Number the NAND gates 1 through 6. Then, answer the following questions:

(a) With CK = D = low and S = R = high, determine the output state of each gate (1 or 0). Assume the flip-flop is initially set.

(b) Repeat part (a) after CK = high.