PART

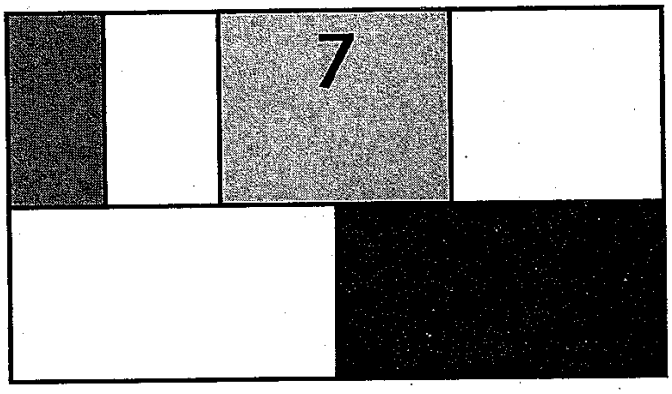
II

A SYSTEMS PERSPECTIVE

"Art, it seems to me,
should simplify.
That, indeed, is very nearly
the whole
of the higher
artistic process;
finding what conventions of form
and what of detail
one can do without
and yet preserve
the spirit of the whole."

Willa Sibert Cather
On the Art of Fiction, 1920

CHAPTER



DESIGNING ARITHMETIC BUILDING BLOCKS

Designing adders, multipliers, and shifters for performance, area, or power

Logic and system optimizations for datapath modules

Higher-level models of arithmetic modules

Introduction

After the in-depth study of the design and optimization of the basic digital gates, it is time to test our acquired skills on a somewhat larger scale and put them in a more system-oriented perspective.

We will apply the techniques of the previous chapters to design a number of circuits often used in the datapaths of microprocessors and signal processors. More specifically, we discuss the design of a representative set of modules such as adders, multipliers, and shifters. The speed of these elements often dominates the overall system performance. Hence, a careful design optimization is required. It rapidly becomes obvious that the design task is not straightforward. For each module, there exist multiple equivalent logic and circuit topologies, each of which has its own benefits and disadvantages in terms of area, speed, or power.

Although far from complete, the analysis presented helps focus on the essential trade-offs that must be made in the course of the digital design process. You will see that optimization at only one design level—for instance, through transistor sizing only—leads to inferior designs. A global picture is of crucial importance. A good digital designer focuses his attention on the gates, circuits, or transistors that have the largest impact on his goal function. The noncritical parts of the circuit can be developed routinely. We will develop first-order performance models that foster understanding of the fundamental mechanics of a module.The discussion also clarifies which computer aids can help to simplify and automate this phase of the design process.

Before analyzing the design of the arithmetic modules, a short discussion of the role of the datapath in the digital-processor picture is useful. This not only helps highlight the specific design requirements for the datapath, but also puts the rest of this book in perspective. Other processor modules, such as the input/output, controller, and memory modules, have different requirements and are discussed in subsequent chapters. After an analysis of the area-time trade-offs in the design of adders, multipliers, and shifters, we will use the same structures to illustrate some of the power-minimization approaches introduced in Chapter 4. The chapter concludes with a short perspective on datapath design and its trade-offs. Appendix E, following this chapter, discusses a number of the layout approaches used in high-speed datapaths.

EX-209-1(1)

Datapaths in Digital Processor Architectures

An analysis of the components of a simple processor puts the different classes of digital circuits and their usage in perspective. Such a processor could be the brain of a personal computer (PC) or the heart of a compact disc player. A typical block diagram is shown in Figure 7.1 and is composed of a number of building blocks that occur in one form or another in almost every digital processor.

- **The datapath** is the core of the processor; it is where all computations are performed. The other blocks in the processor are support units that store either the results produced by the datapath or help to determine what will happen in the next cycle. A typical datapath consists of an interconnection of basic combinational func-

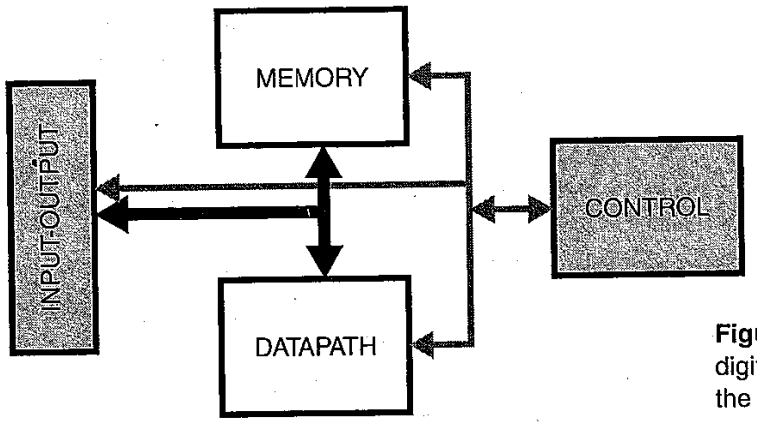


Figure 7.1 Composition of a generic digital processor. The arrows represent the possible interconnections.

tions, such as logic (AND, OR, EXOR) or arithmetic operators (addition, multiplication, comparison, shift). Intermediate results are stored in registers. The design of the arithmetic operators is the topic of this chapter.

EX-209-1(2)

- **The control module** determines what actions happen in the processor at any given point in time. A controller can be viewed as a finite state machine (FSM). It consists of registers and logic, and is hence a sequential circuit. The logic can be implemented in different ways—either as an interconnection of basic logic gates, often called *random logic*, or in a more structured fashion using programmable logic arrays (PLAs) and instruction memories.

- **The memory module** serves as centralized data storage. A broad range of different memory classes exist. The main difference between those classes is in the way data can be accessed, such as read-only versus read-write, sequential versus random access, or single-ported versus multiported access. Another way of differentiating between memories is related to their data-retention capabilities. Dynamic memory structures must be refreshed periodically to keep their data, while static memories keep their data as long as the power source is turned on. Finally, memory structures such as flash memories conserve the stored data even when the supply voltage is removed. A single processor might combine different memory classes. For instance, random access memory can be used to store data and read-only memory to store instructions.

- **The interconnect and input-output circuitry**—the interconnect network joins the different processor modules to one another as well as to the outside world. Unfortunately, the wires composing the interconnect network, are nonideal and present a capacitive, resistive, and inductive load to the driving circuitry. With growing die-sizes, the length of the interconnect wires tends to grow, resulting in increasing values for these parasitics.

Datapaths are often arranged in a *bit-sliced* organization, as shown in Figure 7.2. Instead of operating on single-bit digital signals, the data in a processor is arranged in a *word*-based fashion. For instance, a 32-bit processor operates on data words that are 32 bits wide. This is reflected in the organization of the datapath. Since the same operation has to be performed on each bit of the data word, the datapath consists of 32 identical slices, each of them operating on a single bit. Hence the word *bit-sliced*. The datapath designer can concentrate on the design of a single slice that is repeated 32 times.

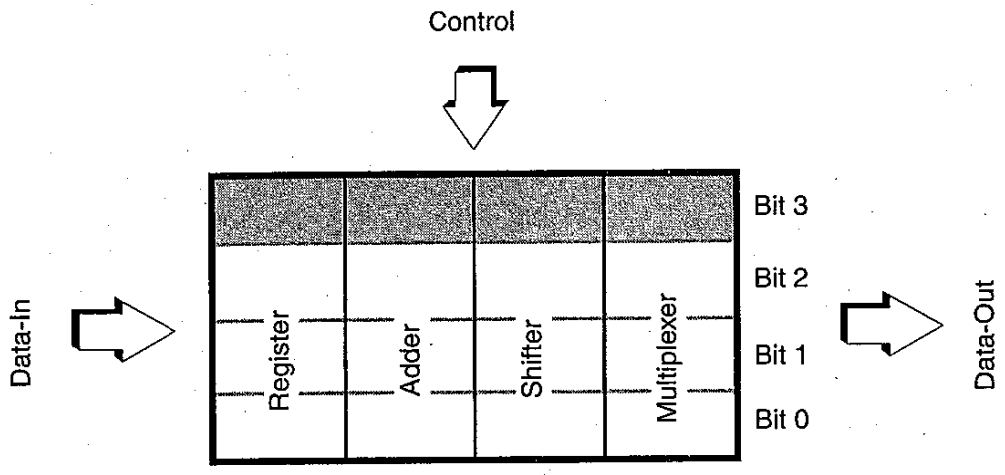


Figure 7.2 Bit-sliced datapath organization.

The Adder

Addition is probably the most commonly used arithmetic operation. Often it is also the speed-limiting element. Therefore, careful optimization of the adder is of utmost importance. This optimization can proceed either at the logic or circuit level. Typical logic-level optimizations try to rearrange the Boolean equations so that a faster or smaller circuit is obtained. An example of such a logic optimization is the *carry look-ahead adder* discussed later in the chapter. Circuit optimizations, on the other hand, manipulate transistor sizes and circuit topology to optimize the speed. Before considering both optimization processes, we provide a short summary of the basic definitions of an adder circuit (as defined in any book on logic design [e.g., Davio83]).

EX-209-2(1)

7.3.1 The Binary Adder: Definitions

The truth table of a binary full adder is given in Table 7.1. A and B are the adder inputs, C_i is the carry input, S is the sum output, and C_o is the carry output. The Boolean expressions for S and C_o are given in Eq. (7.1).

Table 7.1 Truth table for full adder.

A	B	C_i	S	C_o	Carry status
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

$$\begin{aligned} S &= A \oplus B \oplus C_i \\ &= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i \\ C_o &= AB + BC_i + AC_i \end{aligned} \tag{7.1}$$

It is often useful from an implementation perspective to define S and C_O as functions of some intermediate signals G (Generate), D (Delete), and P (Propagate). $G = 1$ ($D = 1$) ensures that a carry bit will be *generated* (*deleted*) at C_O independent of C_i, while $P = 1$ guarantees that an incoming carry will propagate to C_O. Expressions for these signals can be derived from inspection of the truth table.

EX-209-2(2)

$$\begin{aligned} G &= AB \\ D &= \bar{A}\bar{B} \\ P &= A + B \text{ (or } P = A \oplus B) \end{aligned} \tag{7.2}$$

S and C_o can be rewritten as functions of P and G (or D)

$$\begin{aligned} C_o(G, P) &= G + PC_i \\ S(G, P) &= P \oplus C_i \end{aligned} \tag{7.3}$$

Notice that G and P are only functions of A and B and are not dependent upon C_i. In a similar way, we can also derive expressions for $S(D,P)$ and $C_o(D,P)$.

An N-bit adder can be constructed by cascading N full-adder circuits in series, connecting $C_{o,k-1}$ to $C_{i,k}$ for $k = 1$ to $N-1$, and the first carry-in $C_{i,0}$ to 0 (Figure 7.3). This configuration is called a *ripple-carry adder* since the carry bit "ripples" from one stage to the other. The delay through the circuit depends upon the number of logic stages that must be traversed and is a function of the applied input signals. For some input signals, no rippling effect occurs at all, while for others the carry has to ripple all the way from the *least-significant (lsb)* to the *most-significant bit (msb)*. The propagation delay of such a structure (also called the *critical path*) is defined as the *worst-case delay over all possible input patterns*.

EX-210-1 / EX-210-2

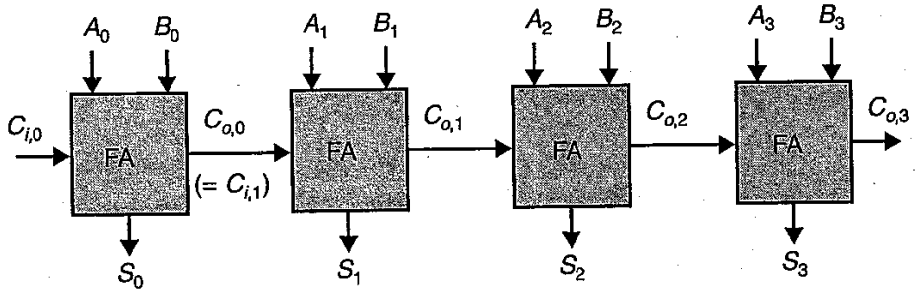


Figure 7.3 Four-bit ripple-carry adder: topology.

In the case of the ripple-carry adder, the worst-case delay happens when a carry generated at the least significant bit position propagates all the way to the most significant bit. The delay is then proportional to the number of bits in the input words N and is approximated by Eq. (7.4).

$$t_{adder} \approx (N-1)t_{carry} + t_{sum} \tag{7.4}$$

STE-102-705

where t_{carry} and t_{sum} equal the propagation delays from C_i to C_o and S, respectively.[1]

Example 7.1 Propagation Delay of Ripple-Carry Adder

Derive the values of A_k and B_k ($k = 0...N-1$) so that the worst-case delay is obtained for the ripple-carry adder.

EX-210-3(1)

The worst-case condition requires that a carry be generated at the lsb position. Since the input carry of the first full adder C_{i0} is always 0, this means that both A_0 and B_0 must equal 1. All the other stages must be in propagate mode. Hence, either A_i or B_i must be high, but not both at the same time. Finally, we would like to physically measure the delay as a transition on the msb sum-bit. Assuming an initial value of 0 for S_{N-1}, we must arrange a $0 \rightarrow 1$ transition. This is achieved by setting both A_{N-1} and B_{N-1} to 0 (or 1), which yields a high sum-bit given the incoming carry of 1.

For example, the following values for A and B trigger the worst-case delay for an 8-bit addition. The rightmost bit represents the msb in this binary representation. Observe that this is only one of the many worst-case patterns. Derive some others as an exercise.

A: 0000001; B: 01111111

Two important conclusions can be drawn from Eq. (7.4).

- The propagation delay of the ripple-carry adder is *linearly* proportional to N. This property becomes increasingly important when designing adders for the wide datapaths ($N = 16...128$) that are desirable in current and future computers.
- When designing the full adder cell for a fast ripple-carry adder, it is far more important to optimize t_{carry} than t_{sum}, since the latter has only a minor influence on the total value of t_{adder}

Before starting an in-depth discussion on the circuit design of full adder cells, an additional logic property of the full adder is worth mentioning.

Inverting all inputs to a full adder results in inverted values for all outputs.

This property, also called *the inverting property*, is expressed in Eq. (7.5) and will be extremely useful when optimizing the speed of the ripple-carry adder. It states that the circuits of Figure 7.4 are identical.

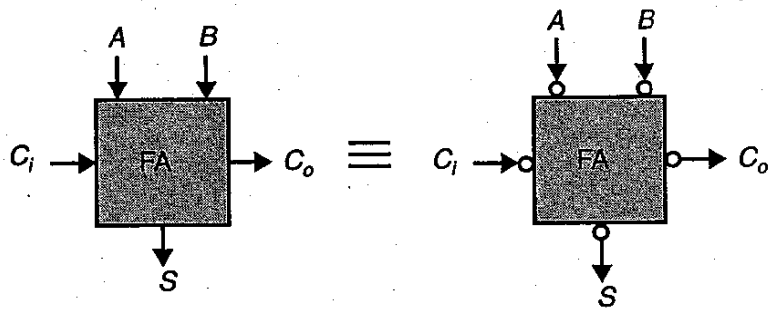


Figure 7.4 Inverting property of the full adder. The circles in the schematics represent inverters.

[1] Eq. (7.4) assumes that for the lsb the delay from the input signals A_0 (or B_0) to $C_{o,0}$ is equal to t_{carry} Although not completely correct, this approximation is acceptable and helps to simplify the expression.

EX-210-3(2)

$$\bar{S}(A, B, C_i) = S(\bar{A}, \bar{B}, \bar{C}_i)$$
$$\overline{C_o}(A, B, C_i) = C_o(\bar{A}, \bar{B}, \bar{C}_i) \tag{7.5}$$

7.3.2 The Full Adder: Circuit Design Considerations

Static Adder Circuit

One way to implement the full adder circuit is to take the logic equations of Eq. (7.1) and translate them directly into complementary CMOS circuitry. Some logic manipulations can help to reduce the transistor count. For instance, it is advantageous to share some logic between the sum- and carry-generation subcircuits, as long as this does not slow down the carry generation, which is the most critical part, as stated previously. An example of such a reorganized equation set is given in Eq. (7.6). The equivalence with the original equation set is easily verified.

$$C_o = AB + BC_i + AC_i$$
$$S = ABC_i + \overline{C_o}(A + B + C_i) \tag{7.6}$$

The corresponding adder design, using complementary static CMOS, is shown in Figure 7.5 and requires 28 transistors. Besides consuming a large area, this circuit is also slow:

- Long chains of series PMOS transistors are present in both *carry*- and *sum* generation circuits.
- The intrinsic load capacitance of the C_o signal is large and consists of two diffusion and six gate capacitances plus the wiring capacitance.

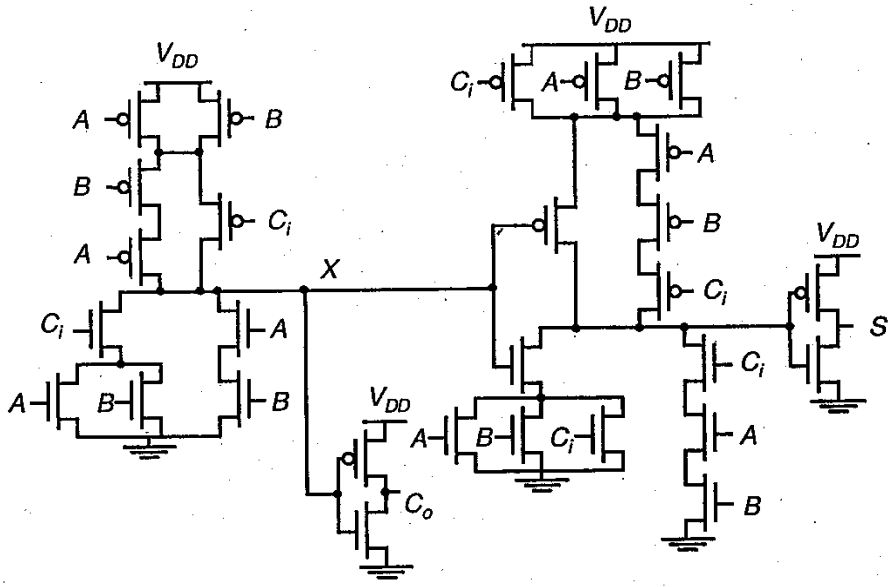


Figure 7.5 Complementary static CMOS implementation of full adder.

- The carry-generation circuit requires two inverting stages per bit. As mentioned above, minimizing the carry-path delay is the prime goal of the designer of high-speed adder circuits.
- The sum generation requires one extra logic stage, but this is not that important, since this factor appears only once in the propagation delay Eq. (7.4).

Although slow, the circuit includes some smart design tricks. Notice that in the first gate of the carry-generation circuit, the NMOS and PMOS transistors connected to C_i are placed as close as possible to the output of the gate. This is a direct application of a circuit-optimization technique discussed in Section 4.2—transistors on the critical path should be placed as close as possible to the output of the gate. For instance, in stage k of the adder, signals A_k and B_k are available and stable long before $C_{i,k}$ (= $C_{o,k-1}$) arrives after rippling though the previous stages. In this way the capacitances of the internal nodes in the transistor chain are precharged or discharged in advance. On arrival of $C_{i,k}$, only the capacitance of node X has to be (dis)charged. Putting the $C_{i,k}$ transistors closer to V_{DD} and *GND* would require not only the (dis)charging of the capacitance of node X but also of the internal capacitances.

The speed of this circuit can now be improved gradually by using some of the adder properties discussed in the previous section. First of all, the number of inverting stages in the carry path can be reduced by exploiting the inverting property—inverting all the inputs of a full adder cell also inverts all the outputs. This rule allows us to eliminate an inverting gate in a carry chain, as demonstrated in Figure 7.6. The only disadvantage is that this design needs different cells for the even and odd slices of the adder chain.

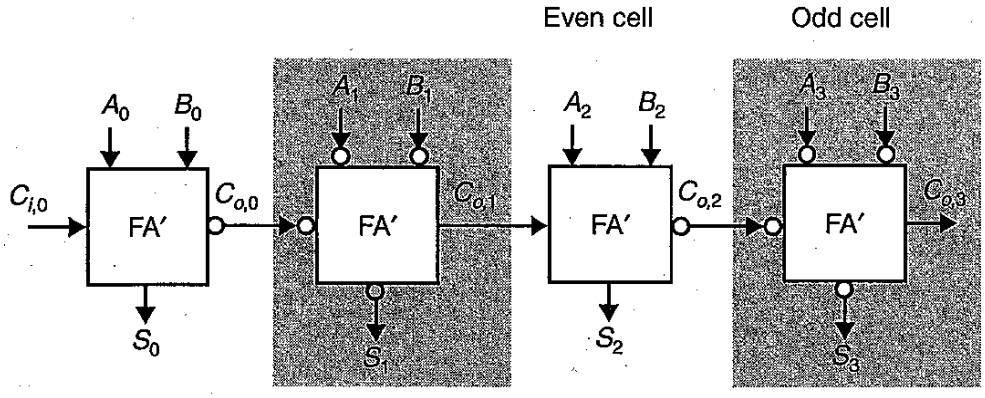


Figure 7.6 Inverter elimination in carry path. FA′ stands for a full adder FA without the inverter in the carry path.

Improved Adder Design

An improved adder circuit, also called the *symmetrical, or mirror adder*, is shown in Figure 7.7 [Weste85]. Its operation is based on Eq. (7.3). The carry generation circuitry is worth analyzing. First, the carry-inverting gate is eliminated, as dictated by the previous section. Secondly, the PDN and PUN networks of the gate are not dual. Instead, they form a smart implementation of the propagate/generate/delete function: when either D or G is high, $\overline{C}_o$ is set to V_{DD} or *GND*, respectively. When the conditions for a *Propagate* are valid

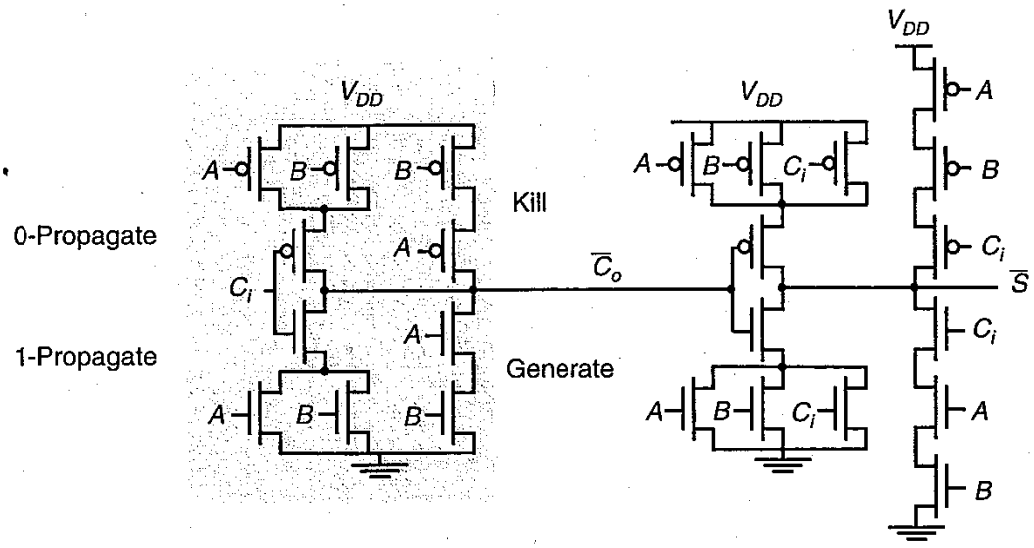


Figure 7.7 Mirror adder—circuit schematics.

(or P is 1), the incoming carry is propagated (in inverted format) to $\overline{C}_o$. This results in a considerable reduction in both area and speed. The analysis of the output circuitry is left to the reader. Some other observations are worth considering.

- This full adder cell requires only 24 transistors.
- The NMOS and PMOS chains are completely symmetrical. This guarantees identical rising and falling transitions if the NMOS and PMOS devices are properly sized. A maximum of two series transistors can be observed in the carry-generation circuitry.
- When laying out the cell, the most critical issue is the minimization of the capacitance at node $\overline{C}_o$. The reduction of the diffusion capacitances is particularly important.
- The capacitance at node $\overline{C}_o$ is composed of four diffusion capacitances, two internal gate capacitances, and six gate capacitances in the connecting adder cell, or a total of ±12 gate capacitances assuming that the diffusion capacitance is approximately equal to a gate capacitance (see Chapter 3). This is identical to the fully complementary implementation of Figure 7.6.
- The transistors connected to C_i are placed closest to the output of the gate.
- Only the transistors in the carry stage have to be optimized for speed. All transistors in the sum stage can be minimum-size.

Example 7.2 Static Adder Design

Consider a slight modification of the static ripple-carry cell of Figure 7.5. A differential approach is used; that is, every full adder cell generates both C_o and $\overline{C}_o$. The crucial gates of the cell are depicted in Figure 7.8. It is left as an exercise for the reader to fill in the rest of the cell (use transmission-gate EXORs as much as possible). The transistor sizes for our 1.2 µm CMOS process are annotated on the schematics (in λ). Observe how a progressive sizing is used. Explain why the PMOS transistor connected to P is smaller than the one connected to C_{in} in the first gate.

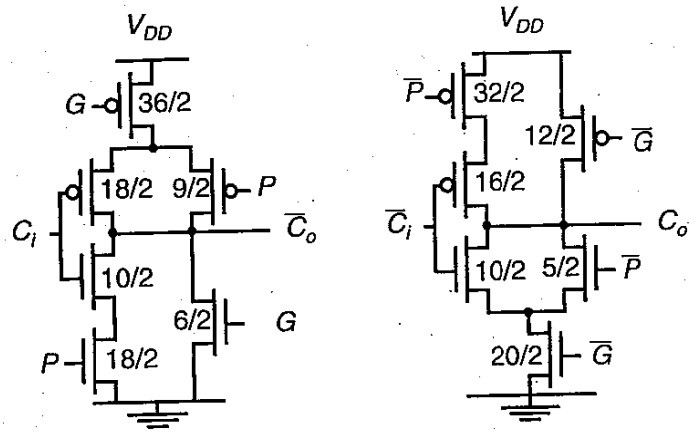


Figure 7.8 Carry-generation circuitry of full-adder cell.

From simulations, we can derive a delay model in the style of Eq. (7.4).

$$t_{add} = t_{A,B \to co} + (N-2)t_{ci \to co} + t_{ci \to s}$$

with $t_{A,B \to co} = 1.63$ nsec, $t_{ci \to co} = 0.32$ nsec, and $t_{ci \to s} = 1$ nsec, for a rise time of 2 nsec at the inputs. This yields the following expression for t_{add}

$$t_{add} = 1.99 + 0.32N \text{ nsec}$$

A 32-bit addition thus takes 12.23 nsec.

Dynamic Adder Design

Most of the adder circuits discussed above can also be designed using dynamic circuit styles such as DOMINO CMOS or *np*-CMOS. An implementation of an *np*-CMOS adder is shown in Figure 7.9. The basic cell requires only 17 transistors, ignoring the extra inverters required for the input or output signals. The alternating even and odd carry stages are realized using NMOS and PMOS networks respectively. The sum-generation networks are also implemented in alternating device types. While it is a direct implementation of Eq. (7.3), the reduced capacitance of the dynamic circuitry results in a substantial speed-up over the static implementation. The load capacitance on the carry bit approximately equals seven equivalent gate capacitances—three diffusion and four gate capacitances.

Example 7.3 Dynamic Adder Design

A layout of the dynamic adder design of Figure 7.9 is plotted in Figure 7.10. Observe the bit-sliced organization. Close to minimum-size transistors are used virtually everywhere except for the carry-generation circuitry. The large devices used there (up to 48/1.2) are easily recognizable in the layout. In the 1.2 μm CMOS technology, the total area per bit equals 120 μm × 31 μm, for a very small carry delay of 200 psec/bit. This delay number ignores the precharge time, which is at least of a similar length.

In addition, the dynamic approach allows for the conception of alternative circuit diagrams that are hard to realize in a static way. An example of such a circuit is shown in Figure 7.11. This adder is called a *Manchester Carry-Chain Adder,* and uses a cascade of pass-transistors to implement the carry chain. *Propagate* and *Generate* signals are gener-

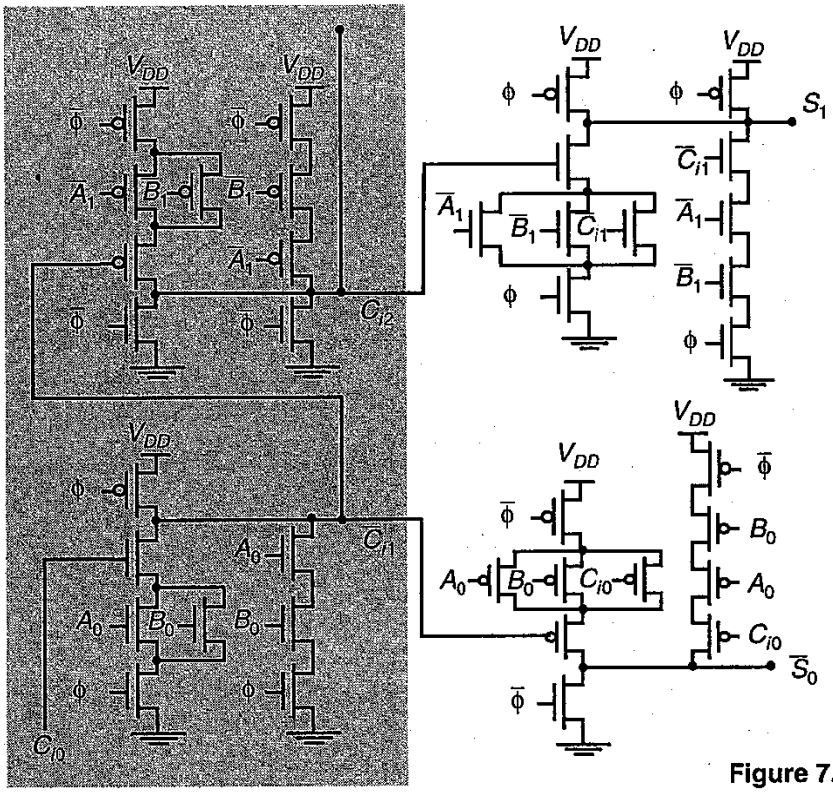


Figure 7.9 Dynamic full adder using the *np* CMOS logic style.

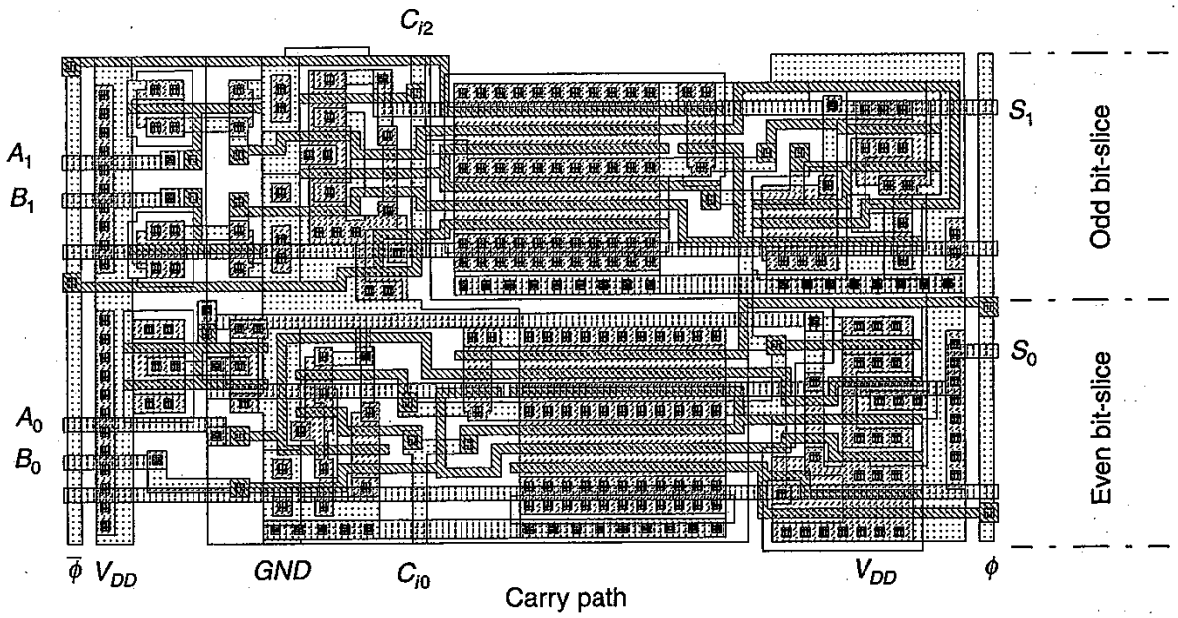


Figure 7.10 Layout of dynamic full adder using the *np*-CMOS logic style (from [Kleinfelder91]).

ated in the traditional way using, for instance, pass-transistor logic. During the precharge phase ($\phi = 0$), all intermediate nodes of the pass-transistor chain are precharged to V_{DD}. During the evaluation phase, the A_k node is discharged when there is an incoming carry and the *Propagate* signal P_k is high, or when the *Generate* signal for stage k (G_k) is high.

The capacitance per node on the carry chain is very small and equals only four diffusion capacitances. Unfortunately, the distributed *RC*-nature of the carry chain results in a

propagation delay that is quadratic in the number of bits N. To avoid this, it is necessary to insert signal-buffering inverters. The optimum number of stages per buffer depends on the buffer delay t_{pbuf} and the resistance and capacitance of the pass-transistors, as was discussed in Section 4.2. An additional speed-up is obtained by a careful sizing of the transistors in the pass-transistor chain. During a discharge of the complete chain, transistor M_0 has to sink the largest amount of current, or $I_{M0} > I_{M1} > \ldots > I_{M4}$. Therefore, it is advantageous to size the transistors progressively. The same is true for the discharge transistors connected to the *Generate* signals and the *Evaluate* transistors. Short propagation delays/bit can be obtained using this technique. For instance, for our 1.2 µm technology, we measured a delay of 192 psec per bit. This delay is very sensitive to any parasitic capacitances at the precharged nodes.

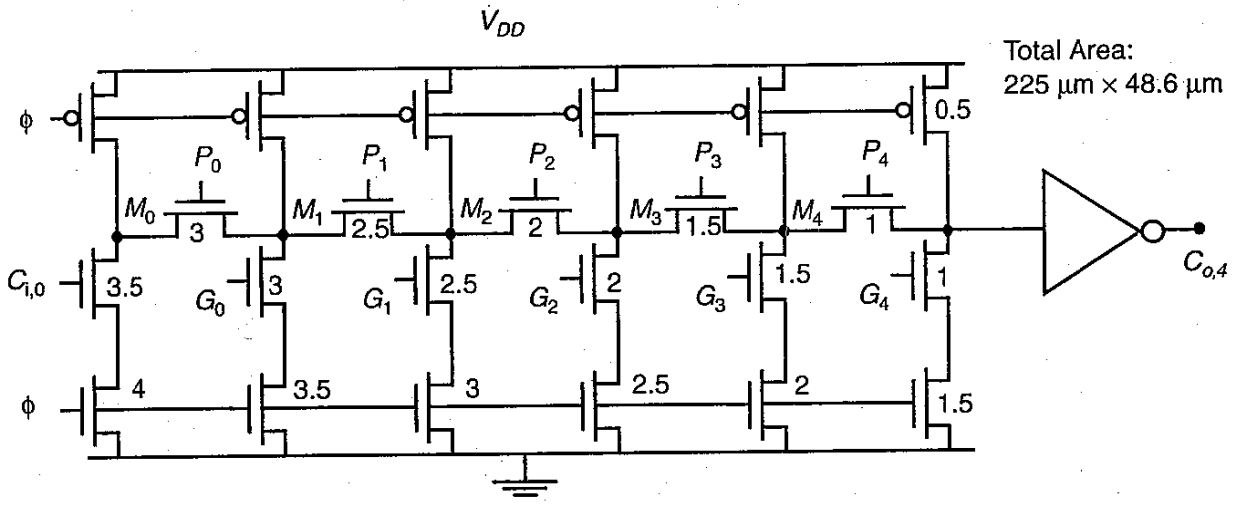


Figure 7.11 Manchester carry-chain adder (5-bit section). The annotated numbers indicate the relative transistor sizes. A unit-size transistor measures (6/1.2).

Problem 7.1 Manchester Carry Chain

The carry chain uses only NMOS transistors in the pass-transistor network (instead of full transmission gates). Discuss why this is an acceptable approach in this configuration.

Example 7.4 Transistor Sizing in the Manchester Carry Chain

The worst-case delay of the carry chain of the adder in Figure 7.11 can be modeled by the linearized *RC* network of Figure 7.12. The linearized on-resistance of the minimum-size transistor is assumed to equal 20 kΩ, and the linearized diffusion capacitance contributed by each minimum-size device is set to 5 fF. The diffusion capacitance at each node in the chain is the sum of the capacitive contributions of two pass-transistors, a pull-down and a precharge device.

The propagation delay of this *RC* network is expressed by the following equation:

$$t_p = 0.69 \sum_{i=1}^{N} C_i \left(\sum_{j=1}^{i} R_j \right) \tag{7.7}$$

where N is the number of nodes in the network and C_i the capacitance of node i to ground. The last term in the equation (the resistive sum) represents the total resistance between node i and

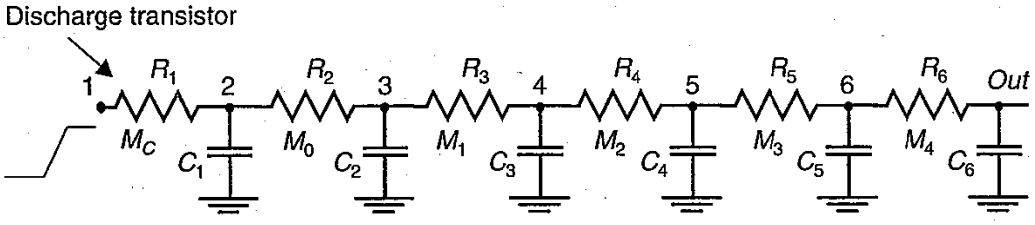


Figure 7.12 Equivalent network to determine propagation delay of carry chain.

the source node (node 1). This expression, called the *Elmore delay*, is discussed further in Chapter 8. The delay of the *RC* network of Figure 7.12 can now be determined

$$t_p = 0.69(C_1R_1 + C_2(R_1 + R_2) + C_3(R_1 + R_2 + R_3) + C_4(R_1 + R_2 + R_3 + R_4) + C_5(R_1 + R_2 + R_3 + R_4 + R_5) + C_6(R_1 + R_2 + R_3 + R_4 + R_5 + R_6))$$

Since R_1 occurs six times in the expression, it makes sense to minimize this contribution by making the first transistor larger than the other ones, or to use progressive scaling.

First consider the case where all transistors are minimum-size. The capacitance at each node is estimated to equal 20 fF, and all resistances are set to 20 kΩ

$$t_p = 0.69C(6R + 5R + 4R + 3R + 2R + R) = 0.69 \times 21 \times RC = 5.8 \text{ nsec}$$

Assume now that the stages are made progressively larger, starting from a minimum-size transistor at the output. The (W/L) of the next-to-last transistor is scaled by a factor k (> 1), which means that its resistance is divided by k, while its associated capacitances are approximately increased by a factor k. The following expressions hold

$$C_i = kC_{i+1};\ R_i = R_{i+1}/k$$

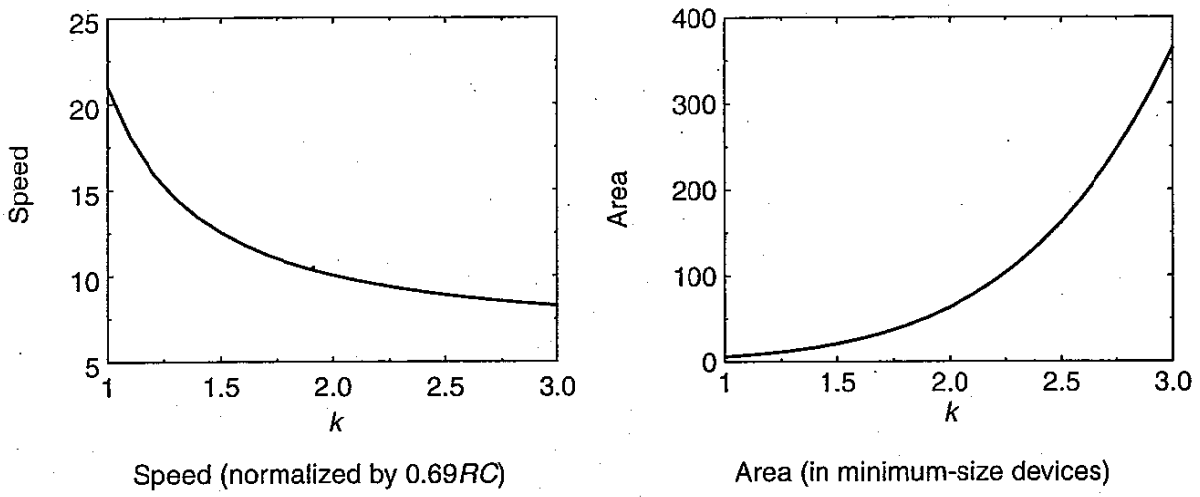
$$C_6 \approx 20 \text{ fF};\ R_6 = 20 \text{ k}\Omega$$

This yields the following expression for t_p

$$t_p = 0.69CR(1 + 2k + 3k^2 + 4k^3 + 5k^4 + 6k^5)/k^5$$

Figure 7.13 plots the propagation delay (normalized with respect to $0.69RC$) and the area of the transistor chain (normalized with respect to a minimum-size transistor) as a function of k. Observe that the area increases dramatically with k, which effectively excludes the use of large scaling factors. The delay starts from $21RC$ for the nonscaled version and decreases sharply for k between 1 and 1.5. For instance, a k-factor of 1.5 reduces the delay by 40% (to $0.69 \times 12 \times RC$) at the expense of a 3.5-fold increase in area. A smaller scaling factor (e.g., $k = 1.2$) does not effect the area much (only 1.65 times larger), but still yields a reasonable speed-up of 20%. Notice that the minimum possible delay for very large values of k equals six time-constants. This means that a linear dependence on the number of RC-stages is achieved in contrast to the quadratic dependence of the nonscaled implementation.

WARNING: Be aware that the above analysis represents only a first-order model. Extensive simulation on extracted layouts is necessary to fine-tune the transistor sizes.



Speed (normalized by 0.69*RC*) Area (in minimum-size devices)

Figure 7.13 Speed and area of Manchester carry chain as a function of scaling factor *k*.

Pipelined Adder

One way to break the dependency between the addition time and the number of bits is to employ the pipelining technique introduced in Chapter 6. Pipeline registers are inserted on the carry path of the adder so that the subsequent sum bits are produced in different time intervals. The pipelining reduces the critical path and the clock period of the adder to a single carry- and sum-generation stage. An N-bit addition now takes N clock cycles, but a result is produced every clock cycle because N additions are performed simultaneously. This extreme usage of pipelining is called *bit-level pipelining* and has been used effectively in signal-processing applications.

An example of a pipelined adder based on the NORA-CMOS dynamic circuit approach is shown in Figure 7.14. The even and odd bits are implemented as ϕ-blocks and $\overline{\phi}$-blocks, respectively. One stage is evaluating, while the next one is in the precharging mode. Observe also how within a single stage, the *np*-CMOS approach is used to cascade gates. Further pipelining can be achieved by inserting an extra register between the carry- and sum-generating circuitry, reducing the critical path to a single carry generation. This incurs an extra cost, because all inputs to the sum-generating circuitry (A_i, B_i, and C_i) must be delayed as well by inserting extra registers.

7.3.3 The Binary Adder: Logic Design Considerations

The ripple-carry adder is only practical for the implementation of additions with a relatively small word length. Fast computers such as mainframes or supercomputers require additions with a word length up to 128 bits. The linear dependence of the adder speed on the number of bits makes the usage of ripple adders rather impractical. Therefore, logic optimizations are necessary, resulting in adders with $t_p < O(N)$. A number of those are discussed briefly below. We concentrate on the circuit design implications, since most of these structures are well-known from traditional logic design.

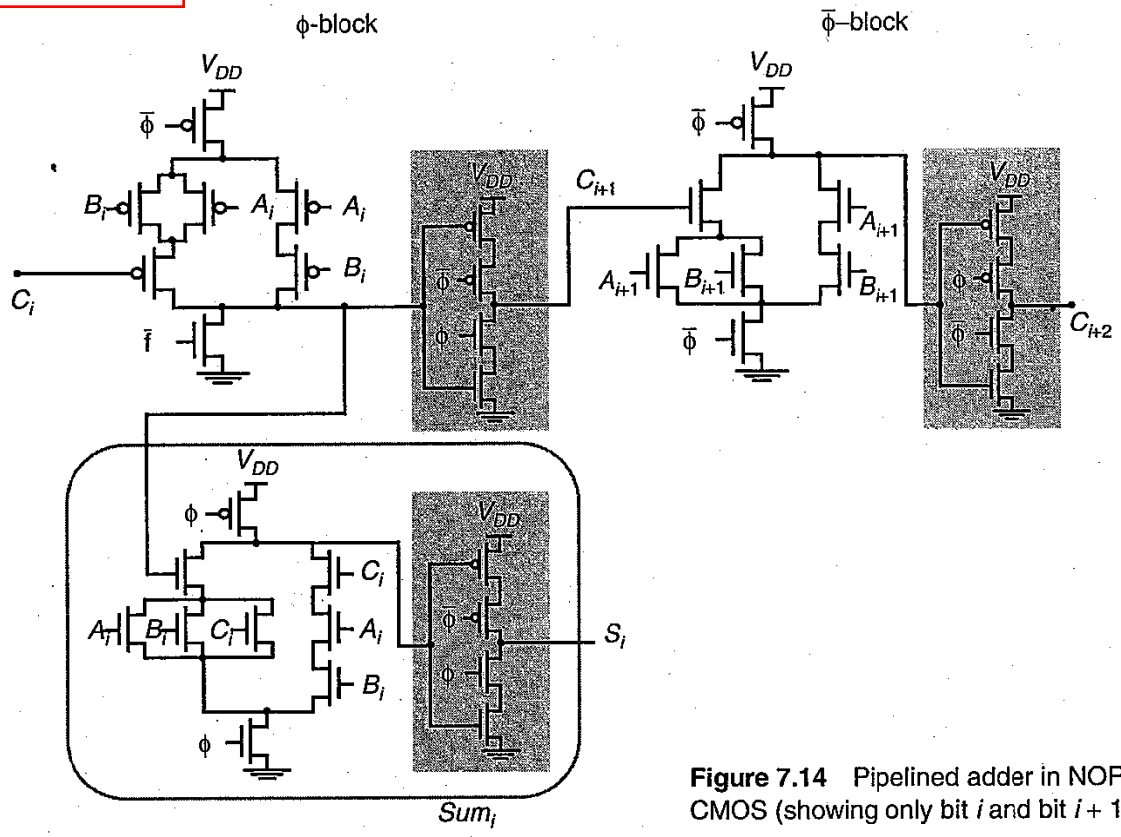


Figure 7.14 Pipelined adder in NORA-CMOS (showing only bit *i* and bit *i* + 1).

The Carry-Bypass Adder

Consider the four-bit adder block of Figure 7.15a and suppose that the values of A_k and B_k ($k = 0\ldots3$) are such that all *Propagate* signals P_k ($k = 0\ldots3$) are high. An incoming carry $C_{i,0} = 1$ propagates under those conditions through the complete adder chain and causes an outgoing carry $C_{o,3} = 1$. In other words,

$$\text{if } (P_0P_1P_2P_3 = 1) \text{ then } C_{o,3} = C_{i,0} \qquad (7.8)$$
$$\text{else either DELETE or GENERATE occurred.}$$

This information can be used to speed up the operation of the adder, as shown in Figure 7.15b. When $BP = P_0P_1P_2P_3 = 1$, the incoming carry is forwarded immediately to the next block through the bypass transistor M_b. Hence the name *carry-bypass adder* [Turrini89]. If this is not the case, the carry is obtained via the normal route.

Figure 7.16 shows the possible carry-propagation paths when the full adder circuit is implemented in Manchester-carry style. This picture demonstrates how the bypass speeds up addition: either the carry propagates through the bypass path, or a carry is generated somewhere in the chain. In both cases, the delay is smaller than the normal ripple configuration. The area overhead incurred by adding the bypass path is small and typically ranges between 10 and 20%.

Let us now compute the delay of an *N*-bit adder. At first we assume that the total adder is divided in (*N*/*M*) equal-length bypass stages, each of which contains *M* bits. An

10.3.3 Read-Write Memories (RAM)

Providing a memory cell with roughly equal read and write performance requires a more complex cell structure. While the contents of the ROM and NVRWM memories is ingrained in the cell topology or programmed into the device characteristics, storage in RAM memories is based on either positive feedback or capacitive charge, similar to the ideas introduced in Chapter 6. These circuits would be perfectly suitable as R/W memory cells but tend to consume too much area. In this section, we introduce a number of simplifications that trade off area for either performance or electrical reliability. They are labeled as either SRAMs or DRAMs depending upon the storage concept used.

Static Random-Access Memory (SRAM)

The generic SRAM cell is (re)introduced in Figure 10.24 and turns out to be virtually identical to a register cell shown in Figure 6.16, deemed of limited use at that time. It requires six transistors/bit. Access to the cell is enabled by the word line, which replaces the clock and controls the two pass-transistors M_5 and M_6. In contrast to the ROM cells, two bit lines transferring both the stored signal and its inverse are required. Although providing both polarities is not a necessity, doing so improves the noise margins during both read and write operations, as will become apparent in the subsequent analysis.

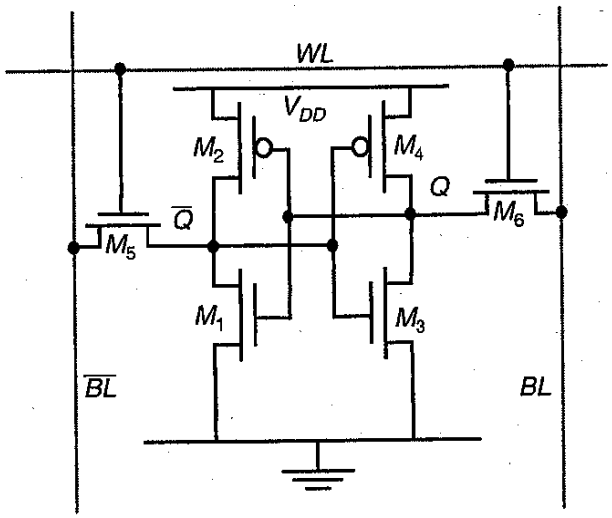


Figure 10.24 Six-transistor CMOS SRAM cell.

Problem 10.5 CMOS SRAM Cell

Does the SRAM cell presented in Figure 10.24 consume stand-by power? Explain. Draw an equivalent pseudo-NMOS implementation. How about the stand-by power in that case?

Operation of SRAM cell. To understand the operation of the memory cell, let us consider the write and read operations in sequence. While doing so, we also derive the transistor-sizing constraints.

Example 10.6 CMOS SRAM Write Operation

In this example, we derive the device constraints necessary to ensure a correct write operation. Assume that a 1 is stored in the cell (or $Q = 1$). A 0 is written in the cell by setting BL to 0 and $\overline{BL}$ to 1, which is identical to applying a reset pulse. This causes the flip-flop to change state if the devices are sized properly.

During the initiation of a write, the schematic of the SRAM cell can be simplified to the model of Figure 10.25 It is reasonable to assume that the gates of transistors M_1 and M_4 stay at V_{DD} and GND respectively as long as the switching has not commenced. While this condition is violated once the flip-flop starts toggling, the simplified model is more than accurate for hand-analysis purposes. It is sufficient that node Q can be pulled below the switching threshold of the cross-coupled inverter, which is assumed to be located at $V_{DD}/2$, to ensure that the flip-flop will toggle. Node $\overline{Q}$ must be raised above $V_{DD}/2$.

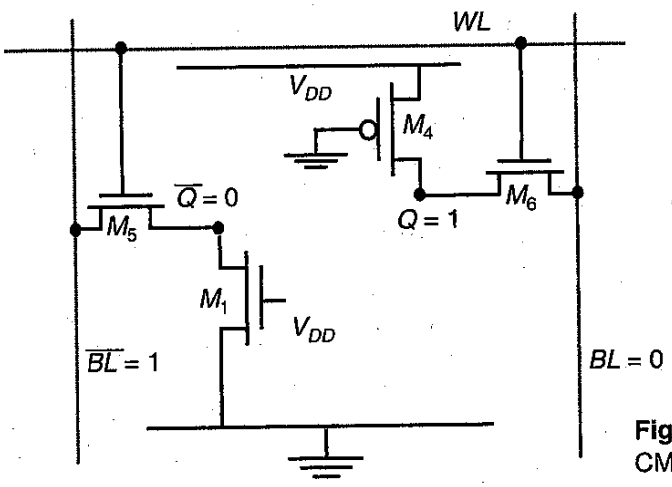


Figure 10.25 Simplified model of CMOS SRAM cell during write ($Q = 1$).

The conditions under which this occurs can be derived by considering the *dc* current equations at the switching threshold point. This is similar to the analysis of Example 6.1.

$$k_{n,M6}\left((V_{DD} - V_{Tn})\frac{V_{DD}}{2} - \frac{V_{DD}^2}{8}\right) = k_{p,M4}\left((V_{DD} - |V_{Tp}|)\frac{V_{DD}}{2} - \frac{V_{DD}^2}{8}\right) \tag{10.1}$$

and

$$\frac{k_{n,M5}}{2}\left(\frac{V_{DD}}{2} - \left(V_{Tn}\left(\text{for } V_{SB} = \frac{V_{DD}}{2}\right)\right)\right)^2 = k_{n,M1}\left((V_{DD} - |V_{Tn}|)\frac{V_{DD}}{2} - \frac{V_{DD}^2}{8}\right) \tag{10.2}$$

The first equation simply expresses that $k_{n,M6}$ should be equal to (or larger than) $k_{p,M4}$. When the cross-coupled inverter is implemented using minimum-size devices, it is acceptable to use a minimum device for the pass-transistor as well. A wider device can help increase the noise margin (as already shown in Figure 6.14).

Evaluating Eq. (10.2) for the 1.2 μm CMOS technology and $V_{DD} = 5$ V yields

$$(W/L)_{n,M5} \geq 10\,(W/L)_{n,M1} \tag{10.3}$$

Pulling up node Q requires a large NMOS pass-transistor, since pulling up a node with a saturated NMOS device is not very effective. This is detrimental in two ways: (1) it increases the size of the cell, and (2) it presents a larger capacitive load to the bit line, which hampers the performance.