CHAPTER 9

STRUCTURED DIGITAL CIRCUITS AND SYSTEMS

9.0 INTRODUCTION

The purpose of Chapter 7 on basic digital building blocks was to introduce primitive logic gates and provide insight into features and limitations of digital MOS circuitry. The goal of this chapter is to extend that knowledge to the design of larger digital systems. These digital systems comprise many logic gates and may occupy a significant part of an integrated circuit chip. Such systems almost always consist of carefully repeated building blocks, with each block based on circuits such as those discussed in Chapter 7. Several different structures, including regular logic arrays, clocked structures, memories, microprocessors, and systolic arrays, are used to demonstrate the capabilities and design requirements for digital integrated circuits and systems.

This chapter begins with an examination of the general topic of structured logic forms. This includes a comparison of random versus structured logic forms, treatment of programmable logic arrays (PLAs), Weinberger arrays, gate-matrix design, and logic gate arrays. These are alternate forms used to implement combinational logic in a structured manner while maintaining control over layout area.

Clocking schemes are introduced next. Time-based signals called clocks are required to provide time order in the operation of digital circuits. In particular, clocks augment simple combinational logic to create sequential systems such as controllers or microprocessors. Following the section on clocking schemes, simple dynamic storage is discussed, a prerequisite for the subsequent treatment of clocked logic, including domino CMOS. Dynamic storage is also useful in building finite-state machines, which are described later in the chapter.

The next sections provide descriptions of several forms of memory including ROM, EPROM, SRAM, DRAM, and static and quasi-static register storage. The first four of these are introduced from a conceptual viewpoint rather than a circuit design viewpoint. The internal design of dense memory subsystems requires detailed circuit design and is outside the scope of this chapter.

With the prerequisites of combinational logic, clocking schemes, and memory available, increasingly complex digital systems such as controllers, finite-state machines, microprocessors, and systolic arrays are outlined. This final major area of the chapter provides an introduction to several practical examples of the complex digital systems that are created from an orderly composition of relatively simple digital building blocks.

9.1 RANDOM LOGIC VERSUS STRUCTURED LOGIC FORMS

A digital logic function may be realized as *random logic* or as *structured logic*. The term *random logic* describes a particular style (or lack thereof) of digital logic design. Some integrated logic circuits are placed within a layout in much the same way that small-scale logic chips are placed on a wire-wrap circuit board and then interconnected. With the many types of small-scale logic functions required, and because particular types of small-scale logic functions may be needed at irregular places within a circuit, the circuit packages and their interconnection wiring sometimes appear to have been randomly placed. Of course, for the circuit to function properly, the interconnections, and probably the package placement, were carefully designed. Nevertheless, random logic is a tag commonly used to describe digital circuits that lack regularity of circuit function, placement, and interconnection. On the other hand, *structured logic* is the term used to characterize logic forms that do demonstrate regularity in their layout and interconnection.

Many digital integrated circuits in the past were designed with large areas devoted to random logic. Early microprocessors such as the Intel 8080 and the Motorola 6800 each contained large sections of random logic. Examination of the die photo of Fig. 9.1-1 reveals that about 50% of the area for the Motorola 6809 microprocessor is devoted to random logic. Designs of this type were considered to have advantages of efficient use of silicon area and potentially fast operation. They have significant disadvantages caused by lengthy integrated circuit layout times, difficulty of testing, and costly modification steps.

Other digital integrated circuits have been designed with highly structured layouts for many years. Most notable among these are all forms of memory chips. Memory chips, such as the 1M-bit dynamic RAM (DRAM) from Texas Instruments shown in Fig. 9.1-2, are composed of many identical memory cells and are naturally structured as regular arrays of these cells. Because of the potential sales volume for memory parts, considerable effort is expended in reducing the size of the basic memory cell, causing memory chips to be among the densest of all integrated circuits.

Most of the newer, large digital integrated circuits, such as the Motorola 68030 and the Intel 80386 microprocessors, have decreased substantially the

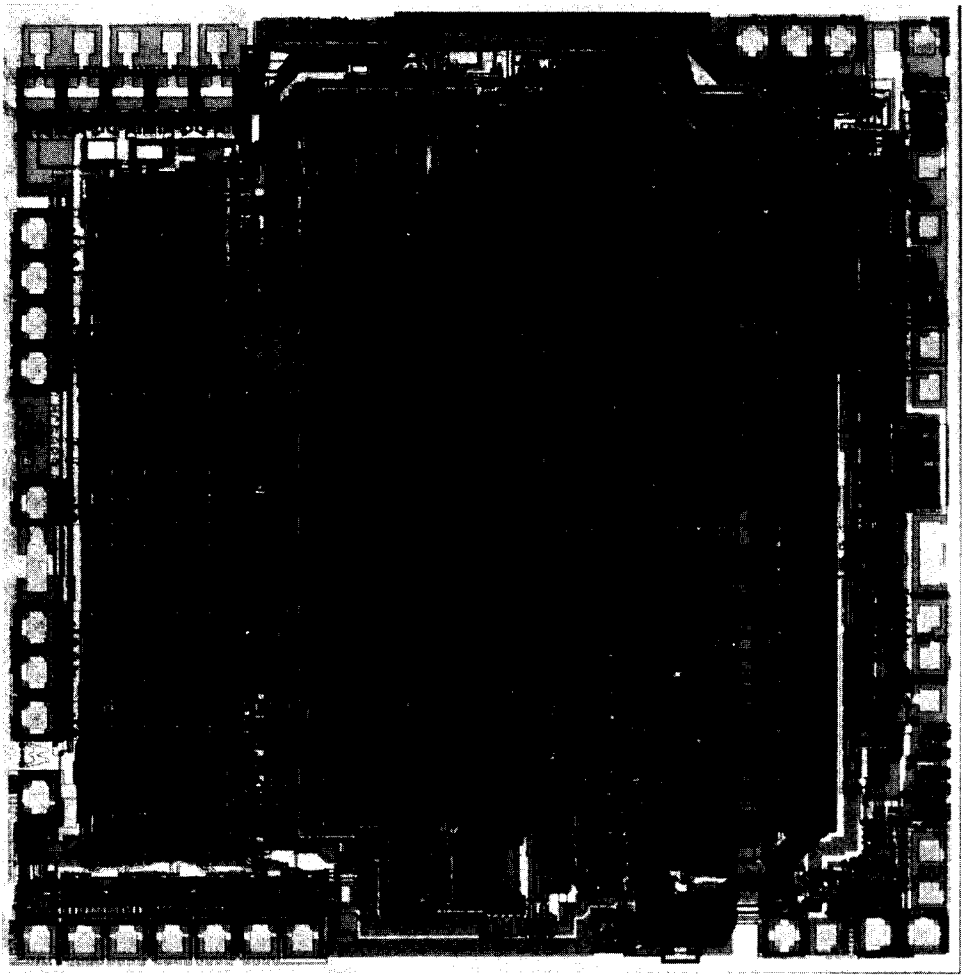


FIGURE 9.1-1
Die photo for Motorola 6809 microprocessor (Courtesy Motorola Inc.).

percentage of silicon area devoted to random logic. This is easily shown by comparing the die photo of the Motorola 68030 of Fig. 9.1-3 with the die photo of Fig. 9.1-1. In fact, because of the complexity of many new chips, random logic design is no longer feasible for large chips. The length of time to design and lay out a complex random logic chip would increase the cost of the chip prohibitively. It would also delay introduction of the chip to the market, a costly consideration. As a result, most new digital integrated circuits increasingly use structured logic forms such as PLAs, microprogram ROMs, data paths, gate arrays, and standard cells to displace random logic design.

A widely used measure introduced by Lattin in 1979[1] is helpful in describing the regularity of an integrated circuit design. This measure, called the chip regularity factor, is defined as the ratio of the total number of transistors on the

FIGURE 9.1-2
Die photo for TI 1M-bit DRAM (Courtesy Texas Instruments Inc.).

chip to the drawn transistors, where total transistors includes all possible ROM and PLA transistor placements. Thus, a design that requires a unique layout for a circuit element and then uses this circuit element n times without change would exhibit a regularity factor of n. At the other extreme, a design with unique layout for each circuit component would exhibit a regularity factor of only 1. For a given complexity of design, a higher regularity factor normally indicates reduced design and layout costs.

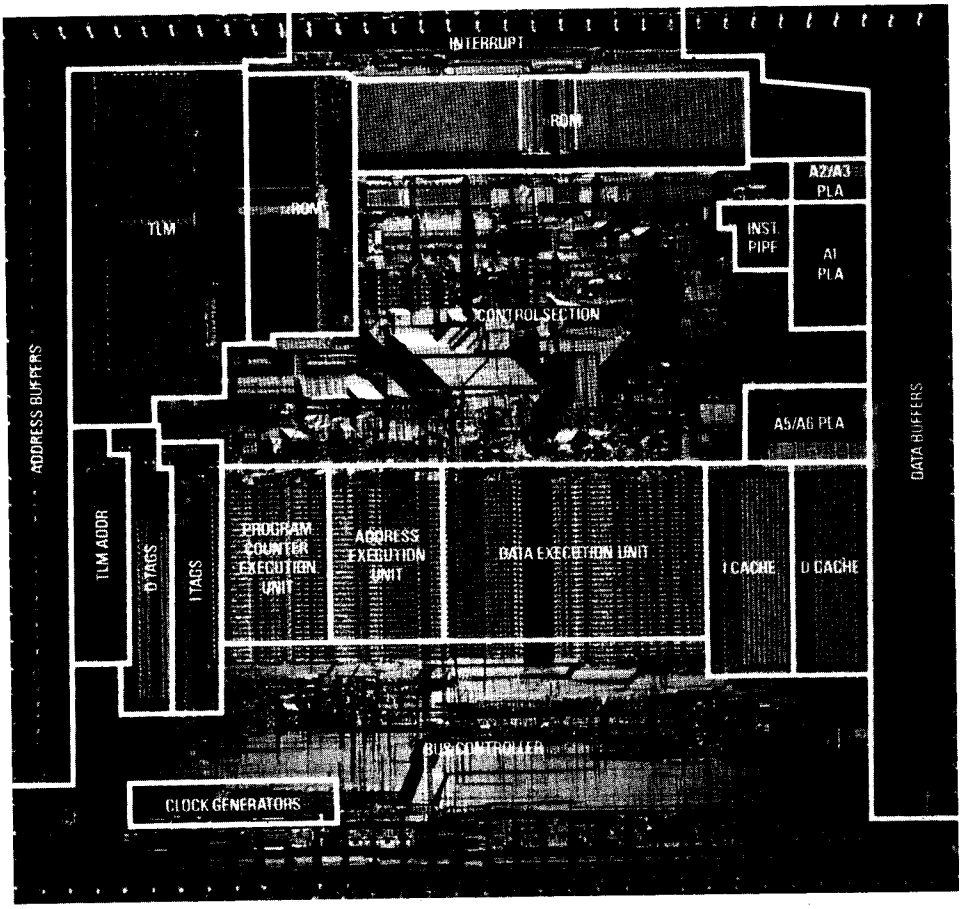


FIGURE 9.1-3
Die photo for Motorola 68030 microprocessor (Courtesy Motorola Inc.).

Some manufacturers estimate that a typical integrated circuit layout designer can lay out about 5 to 10 drawn devices per day. This includes the time to draw, check, and correct a layout. Until recently, the regularity factor for integrated circuits other than memory was not much greater than 1, indicating that almost all devices were drawn individually. Thus, a 50,000 device circuit required twenty man-years just for layout. Some of the newer integrated circuits boast regularity factors above 10. Table 9.1-1 gives the regularity factors for several microprocessor designs.[2] The higher regularity factors were obtained by using structured design forms like those to be described in this chapter.

One method to increase the regularity factor and reduce costs is to develop computer programs that produce integrated circuit layouts directly from high-level descriptions of the circuit's intended function. The term *silicon compiler* has been used to describe such programs. Research is currently underway in this area, and a few special-purpose silicon compilers have been developed. However, complete silicon compiler programs are not yet competitive with manual design using

TABLE 9.1-1
Regularity factor for microprocessor chips

Chip name	Number of devices	Regularity factor
8080	4,600	1.1
8085	6,200	3.1
8086	29,000	4.4
Z8000	17,500	5.0
68000	68,000	12.1
iAPX-432	110,000	7.9
RISC	44,000	27.5

structured logic forms. One major purpose of this chapter is to study integrated circuit forms that lead to structured, repeatable designs for integrated circuit logic.

9.2 PROGRAMMABLE LOGIC ARRAYS

The implementation of logic functions plays a key role in the design of digital systems. Thus, it is important to have a method to realize logic functions within an integrated circuit, without using random logic design. One important method of implementing logic functions in a regular, structured way is to use a PLA (programmable logic array). In this description of PLAs it is presumed that the logic functions used as the input are in the minimal representations desired by a designer. The broad topic of logic minimization is adequately covered elsewhere.[3] This section will concentrate first on describing a typical PLA organization. Then, programs to automate the generation of integrated circuit layout for a PLA from a set of logic equations will be discussed. Finally, PLA size limitations and PLA folding will be described.

Boolean logic equations can always be written in a sum-of-products form as follows.

$$K = AB + AC + BC \tag{9.2-1}$$

$$S = AB\overline{C} + A\overline{B}C + \overline{A}BC + ABC \tag{9.2-2}$$

$$X = A\overline{B} + \overline{A}B \tag{9.2-3}$$

$$Y = \overline{A}\,\overline{B} \tag{9.2-4}$$

In the sum-of-products form, each equation consists of one or more product terms composed of independent variables, for example, A, B, and C, that are ANDed together. If there are several product terms, these product terms are ORed to produce the desired dependent variable. The product terms AB, AC, and BC are ORed to produce the dependent variable K in Eq. 9.2-1. Normally, a PLA can realize several output functions concurrently by producing corresponding dependent variables from sets of independent variables. The independent variables are usually shared among the product terms used to form the dependent results. For example, Eqs. 9.2-1, 9.2-2, 9.2-3, and 9.2-4 are each functions of the independant variables A and B. These equations combine ten product terms to produce the four dependent results K, S, X, and Y.

9.2.1 PLA Organization

The PLA structure can be realized in either NMOS or CMOS technology. In either case, a PLA consists of two major subsections or planes. One is the AND plane, which requires double-rail inputs (each independent variable and its complement) to generate the product terms required by the defining logic equations. The AND plane produces each of the product terms of the right-hand side of Eqs. 9.2-1 through 9.2-4. The other is the OR plane, which forms the dependent results from these product terms. The OR plane must OR the necessary product terms to produce the dependent variables K, S, X, and Y of Eqs. 9.2-1, 9.2-2, 9.2-3, and 9.2-4, respectively.

A simplified block diagram of a PLA is given in Fig. 9.2-1. This figure shows the major AND and OR planes along with required supporting structures.

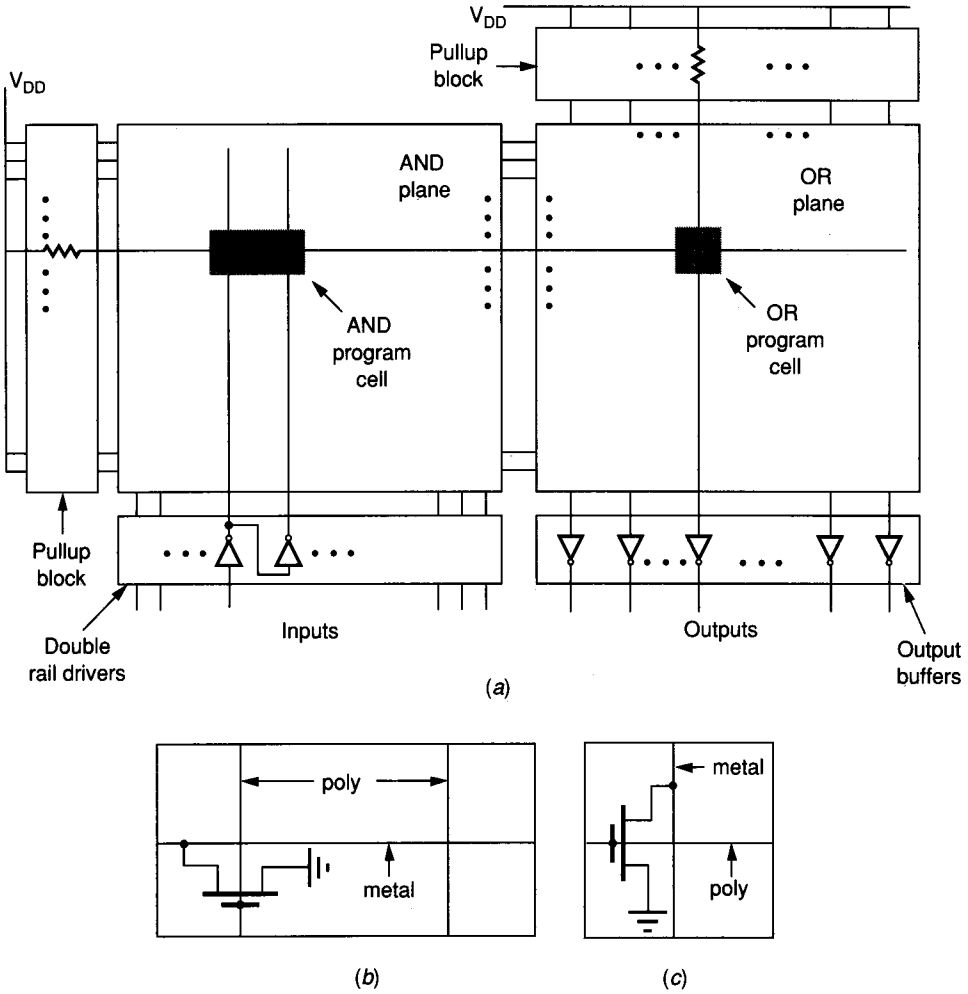


FIGURE 9.2-1
Simplified PLA architecture: *(a)* Major functional blocks, *(b)* AND program cell, *(c)* OR program cell.

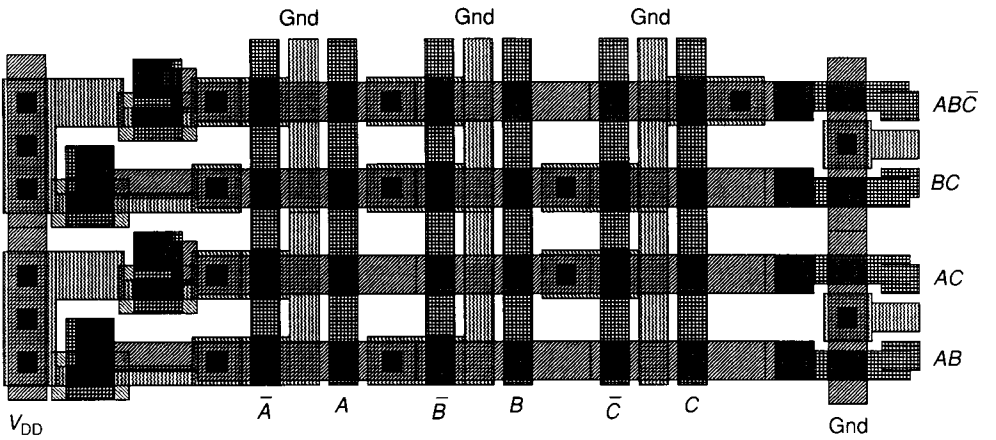


FIGURE 9.2-2
PLA AND-plane detail.

In Fig. 9.2-1, the AND plane is placed on the left with double-rail drivers at the bottom and pullup devices to the left. The OR plane is on the right with pullup devices at the top and output buffers at the bottom. Each double-rail driver accepts an input variable. This input variable is complemented and buffered to drive the vertical polysilicon lines in the AND plane shown in Fig. 9.2-2. Vertical ground lines separate each variable and its complement. The vertical ground lines are run in the diffusion level. Horizontal metal lines provide the path for each product term. The left end of each metal line is connected to a pullup circuit and the right end connects to the OR plane. The metal product lines provide area for contacts to diffusion islands at points between the vertical polysilicon lines associated with different input variables as shown in Fig. 9.2-2. This allows creation of a pulldown transistor to OR the effect of an input variable into the product term. ORing input terms to realize the AND function will be explained in the following paragraph. The product terms of the AND plane are "programmed" by selectively connecting pulldown transistors between the horizontal product term path in metal (the diffusion islands provide the drain connection) and the vertical ground path in diffusion. The vertical signal lines in polysilicon form the gates of these transistors, as is shown in Fig. 9.2-2. Because of the choice of polysilicon for these signal lines, they can directly gate the programming transistors, thereby minimizing layout area. With vertical polysilicon lines, the horizontal product lines must be metal to prevent shorts or unwanted transistors that would occur if the product lines were polysilicon or diffusion, respectively.

Operation of the AND plane of the PLA can be explained with the help of the NOR structure of Fig. 9.2-3, in which the resistor R is used to designate a pullup device. Note that in NMOS technology, the pullup device is generally a depletion transistor as shown in Fig. 9.2-2. In CMOS the pullup device is either a p-channel transistor with its gate grounded or a clocked p-channel pullup. For illustrative purposes, Fig. 9.2-3 shows the realization of the product term $M = A\bar{B}\bar{I}$. The output of a horizontal product term line of a PLA should be high when

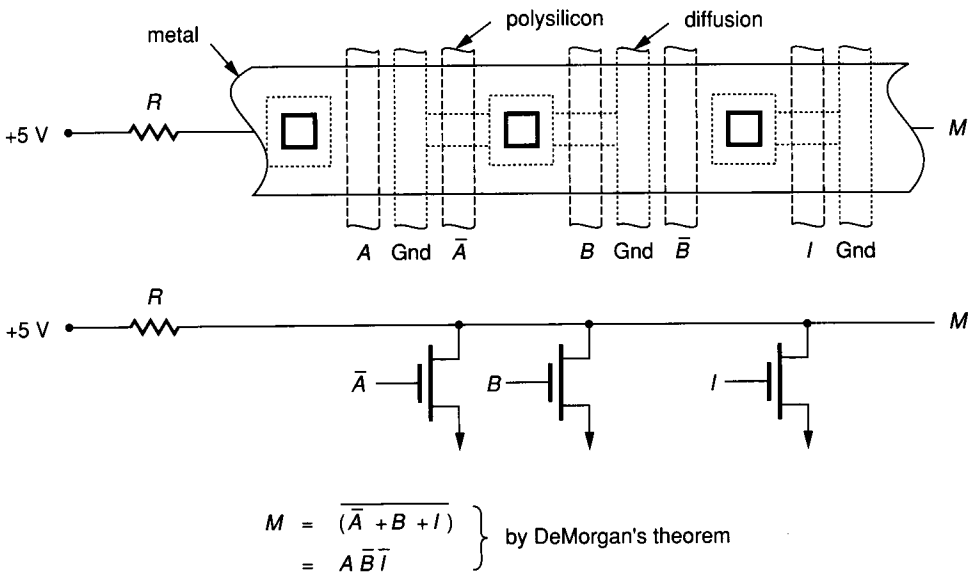


FIGURE 9.2-3
NOR structure of PLA AND Plane.

the individual variables of the product are all true. The pullup device forces the product term output to a logic high voltage unless one of the programming transistors connecting the metal product term line and ground is activated by a high input. If the programming transistors are gated by the complements of the inputs that form the product term, then the product line will be pulled low if one or more of the inputs are low (the complement of the input will be high causing the gate of the programming transistor to be high). This structure is identical to the multi-input NOR gate of Fig. 9.2-3 and requires a sizing ratio between the pullup device and the programming transistor corresponding to the multi-input NOR gate discussed in Sec. 7.4.1. By DeMorgan's theorem, a NOR gate with all inputs inverted logically realizes the AND function, as required for the AND plane.

The OR plane will have the same construction as the AND plane except that everything is rotated 90° clockwise. This may be observed from the right-hand side of Fig. 9.2-1*a*. For the OR plane, the inputs are the product terms from the AND plane. These are available horizontally at the right side of the AND plane in metal, where they are converted to the polysilicon level as they enter the OR plane from the left as shown in Fig. 9.2-2. The outputs from the OR plane are vertical metal lines. These metal lines connect to pullup devices at the top and output drivers at the bottom of the OR plane. Horizontal ground lines in diffusion are placed between alternate pairs of horizontal polysilicon lines in the OR plane. Programming transistors are formed with the drain connected to a vertical metal output line, the source connected to a horizontal diffusion ground line, and the gate formed by a horizontal polysilicon signal line. Once again,

the choice of direction for the polysilicon lines is defined by the need to gate the programming transistors directly to minimize layout area. These transistors OR the proper product terms for each output line. As in the AND plane, this structure realizes a multi-input NOR gate; the outputs must be inverted to realize the OR function. Inverting buffers placed at the bottom of the OR plane achieve this inversion. The structure for the OR plane, including transistor sizing, is usually identical to the AND plane. A complete PLA that realizes Eqs. 9.2-1 through 9.2-4 is shown in Fig. 9.2-4.

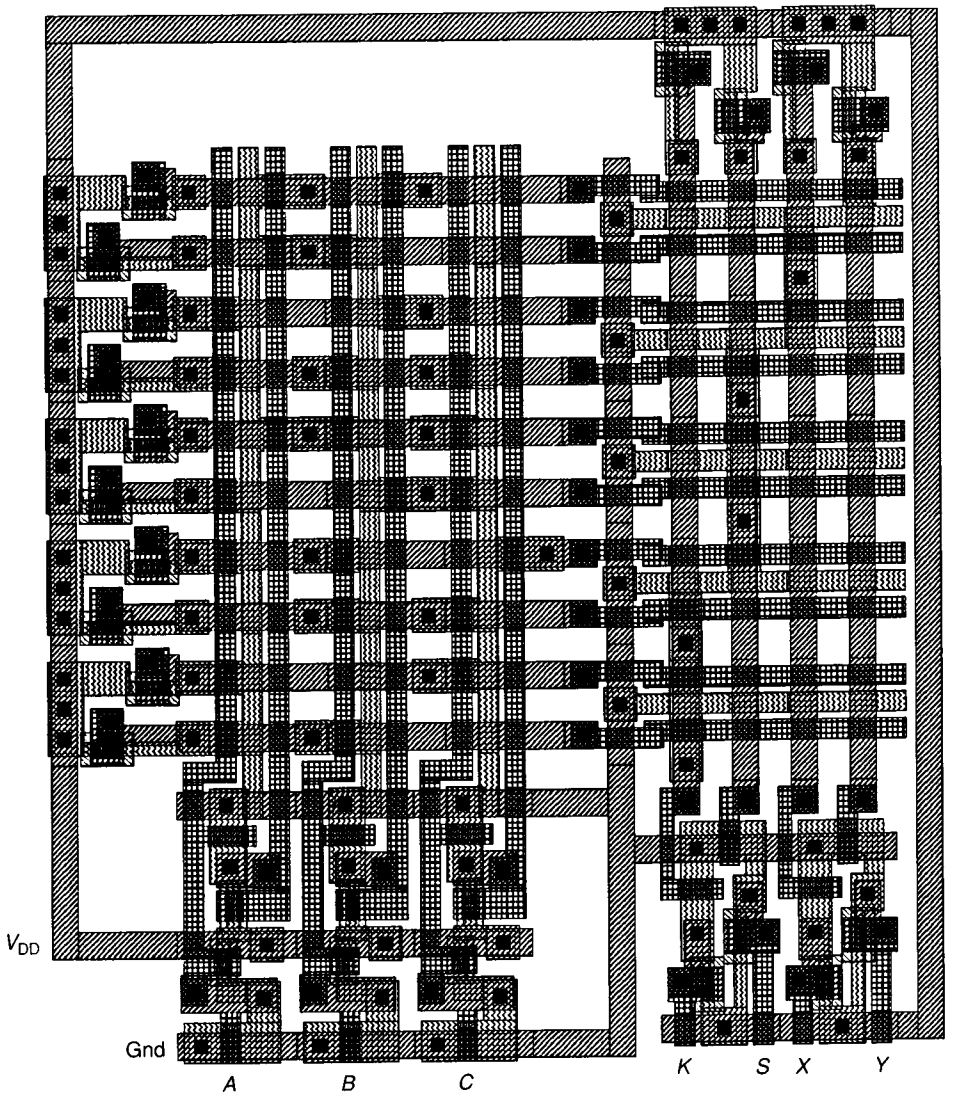


FIGURE 9.2-4
PLA layout for Eqs. 9.2-1 through 9.2-4.

From the previous description, it is easy to see that a PLA is constructed by repeated use of a few simple cell layouts. The primary cells include a double-rail input driver, pullup cell, AND (OR) plane section, AND-OR connect, programmable transistor section, and an inverting output buffer. Examples of these basic cells from the PLA of Fig. 9.2-4 are given in Fig. 9.2-5. In creating the layout for PLA cells, the design and pitch of the basic AND (OR) plane cell must be considered carefully because this cell has the greatest influence on the total area of most PLA implementations. The input drivers, pullup devices, and inverting buffers are designed to match the pitch of the AND (OR) plane cells.

A convenient measure of a PLA's size is the triplet (i, p, o) where i is the number of inputs, p is the number of product terms, and o is the number of outputs. The number of potential transistors in the AND and OR planes is given by the expression $(2i + o)p$. Increasing i or o adds to the width of the AND plane or OR plane, respectively. Increasing p adds to the height of both the AND and OR planes. A relative measure of PLA size is given by the calculation $(2i + o)p$.

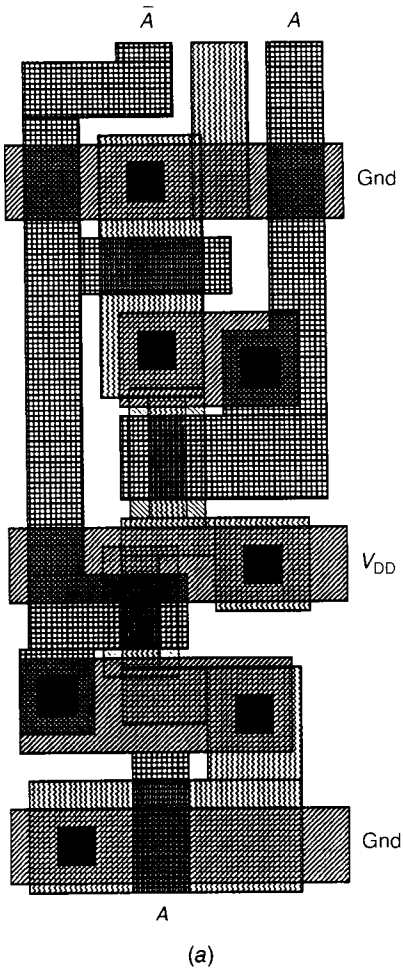
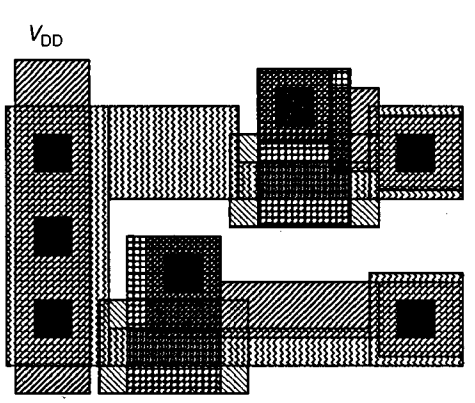
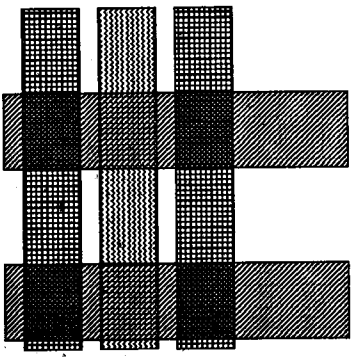


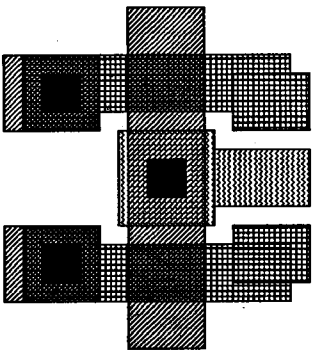
FIGURE 9.2-5
Basic PLA cells: *(a)* Double-rail driver, *(b)* Pull-up pair, *(c)* AND-plane section, *(d)* AND-OR plane connection, *(e)* Inverting buffer pair, *(f)* Programming plug.



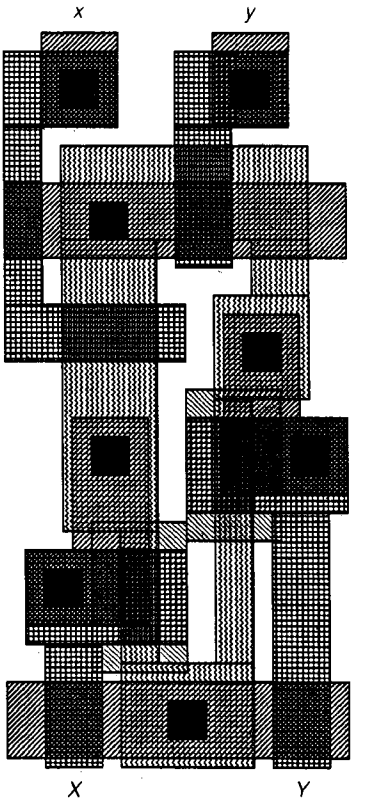
(*b*)



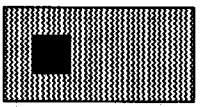
(*c*)



(*d*)



(*e*)



(*f*)

FIGURE 9.2-5
(*Continued*)

This measure neglects a constant area factor for the drivers, pullup devices, and buffers. The i term is doubled because each input produces two vertical lines in the AND plane. The PLA of Fig. 9.2-4 is a (3, 10, 4) PLA.

The speed of a PLA is determined by its size and by the characteristics of the drivers and gates employed. From input to output, a PLA requires five levels of logic, including two for the double-rail driver, one for the AND plane, one for the OR plane, and one for the output driver. Thus, a PLA is likely to be slower than a direct two-level realization for the simplest logic functions. The speed of a PLA can be estimated by analyzing the individual logic stages as explained in Chapter 7. A discussion of PLA size constraints is given in Sec. 9.2.4.

9.2.2 Automatic PLA Generation

An important consequence of the structured form for a PLA is the ability to generate a PLA layout automatically from a set of logic equations. The detailed layout of the standard cells comprising a PLA needs to be accomplished only once for a given set of design rules and technology. A PLA generator program can be written to accept a list of input variables, logic equations based on those variables, and the layout definition of the standard cells. From this information, the PLA generator can formulate the complete layout of the PLA along with size, power, and delay estimates. A PLA generator is a proven example of automatic layout generation based on higher-level functional definitions.

If a PLA generator program has been written for a particular technology, the required input from a designer is quite simple, consisting primarily of the logic equations to be realized by the PLA. As an example of how a PLA generator program might be used, consider the following computer/designer interaction, with computer input from a designer shown in upper-case letters and computer output shown in lower-case letters.

RUN PLAGEN
input variables: A, B, C
logic equations:
$C = AB + AC + BC$
$S = ABC\& + AB\&C + A\&BC + ABC$
$X = AB\& + A\&B$
size: $x = 260$ μm, $y = 320$ μm
power estimate: 2 mW
delay estimate: 10 ns
output file name: pla.cif

In this example, the designer must specify the input variables and the corresponding logic equations. The program creates a geometrical layout description file (pla.cif) and size, power, and delay estimates. If the PLA generator program has

been written in a technology-independent manner, then the designer must also specify the library containing the standard PLA cells for the technology. The availability of automatic PLA generation simplifies the design of many digital systems. A most important feature is that the designer can be confident the PLA layout is free from human layout or programming errors and is therefore correct for the particular set of logic equations input to the program.

9.2.3 Folded PLAs

The standard PLA form just described is ideal for some sets of logic equations; however, an improved form called a *folded PLA* is used under the following conditions. If two product terms are functions of disjoint sets of input variables, and these disjoint input sets can be spatially segregated, then it is possible for two distinct input terms and their complements to share the same AND-plane columns. This reduces the width of the PLA by two columns and is called AND-plane folding. If two output terms are functions of disjoint sets of product terms, and these disjoint product terms can be spatially segregated, then it is possible for two distinct output terms to share the same OR-plane column. This reduces the width of the PLA by one column and is known as OR-plane folding. Either of these folding operations reduces the area required by the PLA. Unfortunately, the folding of different groups of variables interact so that optimal PLA folding is a difficult problem. Heuristics are normally used to find a good folded structure for a PLA.[4,5] Both standard PLA and folded PLA implementations for a set of logic equations are discussed in the following example.

Example 9.2-1. Folded PLA Structure Implement the following logic equations with a standard PLA structure and with a folded PLA structure if possible.

$$S = AB\overline{C} + A\overline{B}C + \overline{A}BC + ABC$$

$$K = AB + BC + AC$$

$$R = \overline{A}B + A\overline{B}$$

$$W = \overline{D}E + D\overline{E}$$

$$X = DEF + \overline{D}\,\overline{E}\,\overline{F}$$

$$Y = ABC + DEF + D\overline{E}$$

Solution. These equations can be implemented as a standard (6,13,6) PLA, as shown symbolically in Fig. 9.2-6. Only the programming planes are shown; an x is used to locate each programming transistor.

The equations can also be implemented by the folded PLA of Fig. 9.2-7. In the folded AND-plane (3,13,6) PLA of Fig. 9.2-7, the y's represent product terms associated with the input variables at the top of the AND plane while the x's represent product terms associated with input variables at the bottom of the AND plane. If x and y input terms appear on the same column, the line between them must be disconnected. These product terms are ORed to produce the dependent outputs.

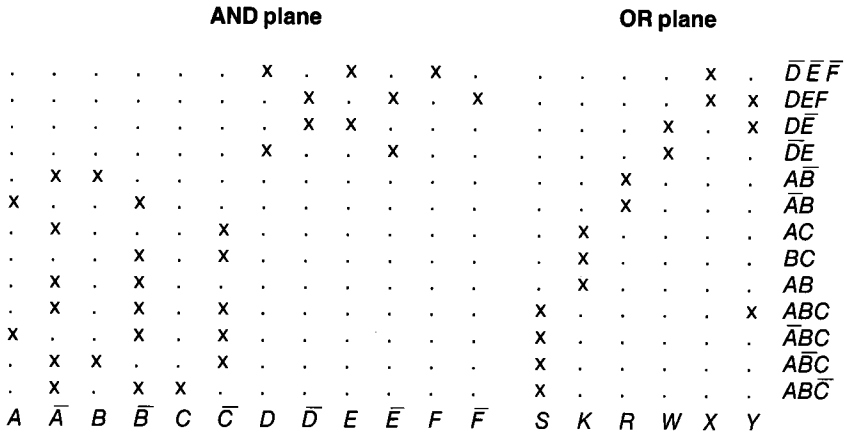


FIGURE 9.2-6
Standard PLA implementation (6,13,6).

In the folded AND-plane, folded OR-plane (3,13,4) PLA, shown in Fig. 9.2-8 the x's represent programming transistors associated with the lower input variables to the AND plane or the lower output terms of the OR plane; the y's represent programming transistors associated with the upper input variables to the AND plane or the upper output terms of the OR plane. According to the PLA size measure introduced in Sec. 9.2.1, the relative sizes are 936 for Fig. 9.2-6, 468 for Fig. 9.2-7, and 312 for Fig. 9.2-8. The area reduction using the dually folded PLA structures is substantial.

9.2.4 Large PLAs

Although PLAs provide an excellent means to organize sets of logic equations, large PLA structures are not necessarily desirable. Extremely large PLA structures suffer from two related disadvantages. As the size of a PLA grows, increased

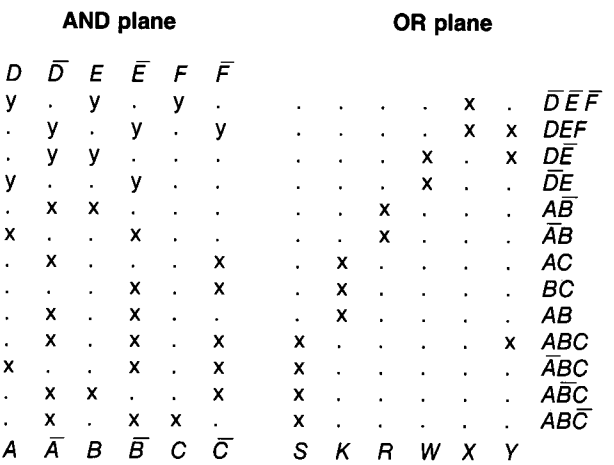


FIGURE 9.2-7
Folded AND-plane PLA implementation (3,13,6).

AND plane						OR plane				
D	$\bar{D}$	E	$\bar{E}$	F	$\bar{F}$	W	X			
y	.	y	.	y	.	.	y	.	.	$\bar{D}\bar{E}\bar{F}$
.	y	.	y	.	y	.	y	.	x	DEF
.	y	y	.	.	.	y	.	.	x	$D\bar{E}$
y	.	.	y	.	.	y	.	.	.	$\bar{D}E$
.	x	x	.	.	.	.	.	x	.	$A\bar{B}$
x	.	.	x	.	.	.	.	x	.	$\bar{A}B$
.	x	.	.	.	x	.	x	.	.	AC
.	.	.	x	.	x	.	x	.	.	BC
.	x	.	x	.	.	.	x	.	.	AB
.	x	.	x	.	x	x	.	.	x	ABC
x	.	.	x	.	x	x	.	.	.	$\bar{A}BC$
.	x	x	.	.	x	x	.	.	.	$A\bar{B}C$
.	x	.	x	x	.	x	.	.	.	$AB\bar{C}$
A	$\bar{A}$	B	$\bar{B}$	C	$\bar{C}$	S	K	R	Y	

FIGURE 9.2-8
Folded AND-plane, folded OR-plane PLA implementation (3,13,4).

interconnection capacitances within the larger AND and OR planes slow the operation of the circuit considerably. Increasing the size of the drivers will reduce the delay, but additional power and area are then required. A second disadvantage of a large PLA is that PLAs with many inputs and outputs tend to be sparsely populated with programming transistors. In these PLAs each output is likely to be a function of only a few inputs; thus, substantial area may be wasted within the AND and OR planes to bypass unused signals for a given product term or output term. Some unused area may be recaptured through the use of folded PLA structures. The foregoing observations cause many designers to group related logic signals into separate smaller PLAs rather than into one large PLA.

The standard PLA organization was explained in this section. A typical cell based structure for a PLA allows creation of programs to generate automatically the PLA layout from logic equations. The ability to fold PLA structures can greatly reduce the area required for the PLA. Multiple smaller PLAs are sometimes used in place of single large PLAs to reduce area, power, and delay.

9.3 STRUCTURED GATE LAYOUT

The realization of logic functions is so basic to digital design that many forms of structured layout have been developed. The PLA, introduced previously, has achieved the widest usage of these forms, and because of its importance a separate section was devoted to the PLA. Other structured layout forms, including Weinberger arrays and gate matrix layout, have received attention and are covered together in this section. By design, the PLA is used to realize two-level logic functions. Yet many digital designs are simplified through the use of multilevel logic with intermediate functions used as inputs to subsequent logic functions. The structured logic forms discussed here allow direct realization of multilevel logic functions. The design, operation, and limitations of the Weinberger array and the gate matrix layout styles are explored.

9.3.1 Weinberger Arrays

The Weinberger array is perhaps the earliest example of structured logic. Originally reported in 1967[6], this structure was first based on the PMOS NAND logic gate. More recent implementations have been based on NMOS NOR gates with depletion pullups. This structure can also be realized in CMOS using p-channel pullups with their gates grounded. Because any two-level Boolean logic function can be realized in the NOR-NOR form, the Weinberger structure is completely general. This logic form is easily extended to multiple levels of logic where this is desirable. In addition, the array is easily augmented with new logic functions without change to the original structure.

The basic form for the Weinberger NOR array is shown in Fig. 9.3-1. Vertical columns with a pullup device at one end provide the outputs for logic functions. Alternate pairs of output columns are separated by ground columns. The pullup device is the load for the NOR gate, while a pull-down transistor is placed between the vertical output column and a ground column for each input to the logic function. Device sizing is determined by the NOR gate structure as explained in Sec. 7.4.1. Gate inputs are received horizontally in polysilicon. Because pullup devices are placed at the top of the vertical columns, this structure allows inputs to be provided from the left with outputs available at the bottom. Horizontal lines connected to a vertical output line can serve as inputs to subsequent logic gates. The horizontal lines can also provide the output of the matrix on the right side opposite the inputs.

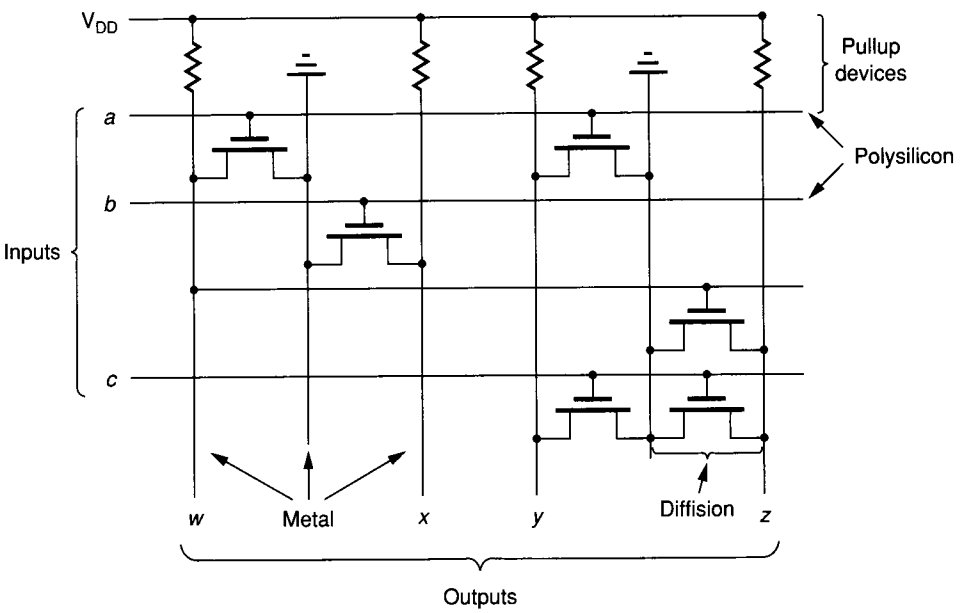


FIGURE 9.3-1
Weinberger NOR array organization with $w = \overline{a}$, $x = \overline{b}$, $y = \overline{a + c}$, $z = \overline{w + c}$.

Example 9.3-1. Exclusive-OR function Use the Weinberger NOR array structure to implement the exclusive-OR function.

Solution. First, the exclusive-OR must be written in product-of-sums form. This is easily accomplished by analyzing the following truth table for exclusive-OR.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

The required logic equation is obtained by writing the complement of the logic function in terms of the zero output rows from the truth table as

$$\overline{X} = \overline{A}\,\overline{B} + AB \tag{9.3-1}$$

and converting to the NOR-NOR form as

$$X = \overline{(\overline{A + B}) + (\overline{A} + \overline{B})} \tag{9.3-2}$$

Figure 9.3-2 shows a schematic layout for this function. Note that two vertical output columns are used to generate the complements of the input variables; two output columns generate the first-level NOR functions; and a final output column generates the desired dependent variable, X.

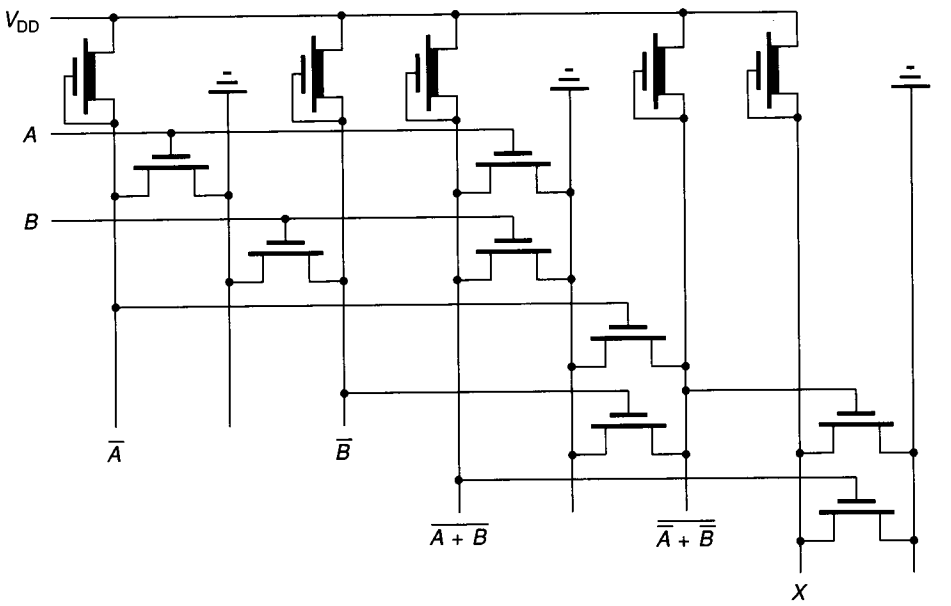


FIGURE 9.3-2
NMOS Weinberger array implementation of exclusive-OR function.

The structure shown in the example is easily expanded by adding horizontal input terms at the bottom and vertical output columns at the right. The output of the exclusive-OR example above is available as an input to expanded logic functions, potentially resulting in multilevel logic. For logic functions with many inputs and outputs, the structure becomes unwieldy because of sparse distribution of the pulldown transistors. This structure is most useful in depletion-load NMOS, CMOS with a grounded-gate p-channel pullup, or clocked logic forms because of the single pullup device per column. It does not map well to complementary gate structures. Because of the simplicity of its form, a Weinberger array is easily generated by a computer program that transforms input logic equations into a layout description. In fact, one of the first silicon compilers[7] used this structure for logic functions in its control section.

9.3.2 Gate Matrix Layout

More recently, a second form of structured logic layout that is suitable for CMOS was described. Called gate matrix layout, this organization was used in the development of an early 32-bit microprocessor.[8] Like the Weinberger NOR array, *gate matrix layout* is composed of a matrix of intersecting rows and columns as shown in Fig. 9.3-3. Transistors are instantiated along the rows, while inputs and outputs primarily use the columns. The rows are mostly diffusion, while the columns are polysilicon. Metal is available in both horizontal and vertical directions for interconnections. A polysilicon column is required for each logic input and each logic output.

Figure 9.3-3 shows a gate matrix schematic layout for a 2-input NAND gate. The layout is separated into two partitions with the upper partition containing n-channel transistors and the lower partition populated by p-channel transistors. Except for the short metal connection to V_{SS}, all other vertical lines represent polysilicon. The polysilicon lines serve a dual function, acting both as a vertical connection medium and as the gates of transistors. The horizontal n-diffusion segment in the upper partition of Fig. 9.3-3 is gated by the two polysilicon NAND-

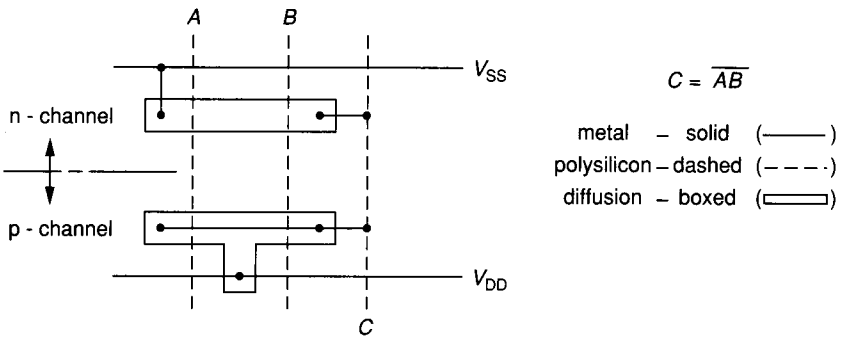
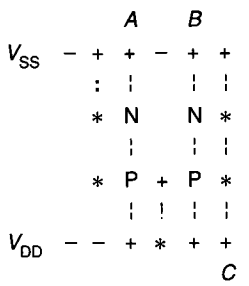


FIGURE 9.3-3
Gate matrix layout for NAND gate.

gate input lines resulting in a series pulldown path. The left end is connected to V_{SS} while the right end is connected to the vertical polysilicon output line, C. The horizontal p-diffusion segment, implementing the parallel pullup transistors of the NAND gate, is bisected by a p-diffusion connection to V_{DD}. This segment is gated by the two polysilicon NAND-gate input lines. Opposite ends are connected horizontally to the vertical polysilicon output line by metal. Note that this output line could gate other transistors to implement a more complex logic function. In general, more complex logic functions are created by adding transistors and connections. The gate matrix layout structure is suitable for implementing logic equations using NAND gates, NOR gates, and inverters in classical CMOS logic form. Device sizing is determined as explained in Sec. 7.6 for CMOS logic. The number of inputs to multi-input CMOS gates is limited by asymmetry of the pullup/pulldown paths as the number of inputs (termed *fan-in*) increases. The fan-in limit depends on the application.

To begin a gate matrix layout, the designer can draw a series of vertical polysilicon lines corresponding to the circuit inputs. The number of lines must be greater than or equal to the number of inputs because some lines may be required for outputs. Associated transistors for a gate are placed along the same row as shown in Fig. 9.3-3. Connections between transistors on different rows are accomplished with metal or with diffusion that runs between the polysilicon columns. Metal can also run horizontally to connect transistors across polysilicon columns as it did in the NAND-gate example.

Because of the structured form for gate matrix layout, symbolic representation of logic functions is possible. In fact, a layout can be defined by a line drawing using a small set of symbols and a few simple rules. This can be created by hand or with computer assistance. Figure 9.3-4 shows an early form



N n–channel transistor
P p–channel transistor
\+ metal–polysilicon or metal–diffusion crossover
* contact
¦ polysilicon or n–diffusion wire
! p–diffusion wire
: vertical metal
– horizontal metal

FIGURE 9.3-4
Symbolic gate matrix layout description for two-input NAND gate.

of symbolic description using only standard symbols from an alphanumeric CRT to describe the 2-input NAND gate of Fig. 9.3-3. These symbols are defined at the bottom of the figure. Modern CAD tools use high-resolution graphics CRTs and allow symbolic input of transistors and interconnections to form a gate matrix layout. Interconnections are symbolized by single lines, and transistors are symbolized by small layout icons. The following rules allow a symbolic description of a logic function that is easily updated as technology advances.

1. Polysilicon runs in one direction only with constant width and pitch.
2. Diffusion runners may exist between polysilicon columns.
3. Metal runs in either direction and is of constant width.
4. Transistors exist only on polysilicon columns.
5. Transistor width can be increased by using multiples of the symbol vertically.

A more complex example of symbolic representation to better demonstrate this technology-independent layout style is shown in Fig. 9.3-5.

The symbolic layout methodology outlined here does not specifically consider geometrical design rules. An advantage of gate matrix layout is that a topology that realizes a particular logic function can be defined independent of layout rules. Ultimately, the pitch of the rows is determined by the minimum separation between two discrete transistors. The pitch of the columns is set to leave room

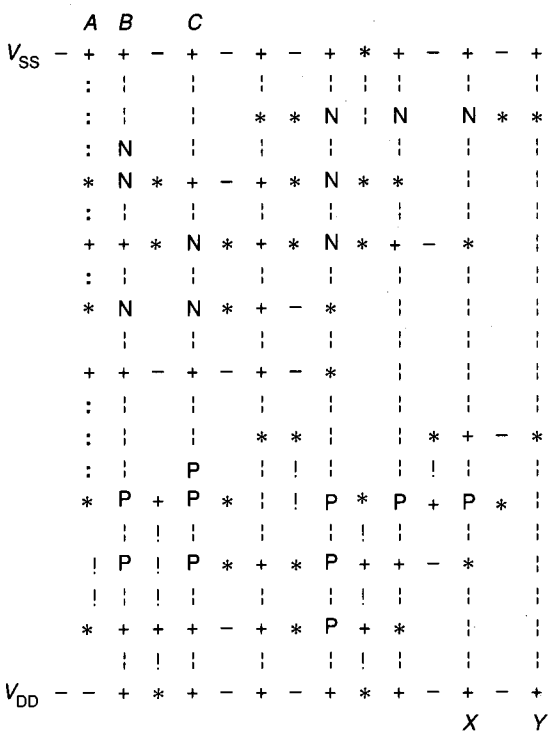


FIGURE 9.3-5
Symbolic gate matrix layout description for 3-input, 2-output logic function (see Prob. 9.8).

to place a diffusion region with contact between polysilicon columns. The matrix pitch for rows and columns is set for minimum-size transistors. The widths of the power and ground buses are set by current requirements.

The CPU for the BellMac 32-bit microprocessor was laid out first as custom logic and then later as a gate matrix layout. This CPU contains about 20,000 transistors. The final gate matrix layout achieved a respectable density of about 2 square mils per transistor in a 4 μ technology. This was slightly denser than an earlier hand-packed version. A 10% to 15% area improvement was reported by hand optimizing early line drawings for the gate matrix layout.[9]

Two examples of structured gate layout to implement general logic equations were described in this section. These differ from the PLA design style, which requires its defining equations in the sum-of-products form. The Weinberger array can be generated algorithmically but may be less dense than the gate matrix style. The gate matrix style provides near handcrafted density while allowing technology-independent layout specification. Both Weinberger arrays and gate matrix layout have been used successfully in many commercial circuits.

9.4 LOGIC GATE ARRAYS

Logic gate arrays provide a simplified means to implement digital integrated circuit designs. This implementation form is consistent with the logic design process using small-scale and medium-scale integrated circuits. Gate arrays incorporate logic building blocks that are familiar to many digital system designers without the high circuit complexity and long turnaround times that are typical for custom integrated circuits. Building blocks such as logic gates, flip-flops, decoders, and counters are available and can be combined into an integrated circuit that has many of the density, power, speed, and reliability characteristics of custom integrated circuits.

Gate arrays are manufactured as regular arrays of patterned blocks of transistors. The transistors in one or more blocks can be interconnected to form logic elements such as gates, flip-flops, and decoders. Figure 9.4-1 shows the topology for a typical gate array. The array consists of columns of transistor blocks separated by wiring channels and surrounded by I/O circuitry. This patterned array of transistors remains unchanged while allowing many different interconnection possibilities. As a result, most integrated circuit fabrication steps can be completed before the interconnections are defined. Ideally, a gate array manufacturer can stock partially fabricated wafers awaiting customer specification of a particular design. The interconnections for a particular design are formed by adding one or more levels of metal interconnection. This is done after the logic design and computer simulation for the desired circuit are complete.

Because all processing steps before metalization are identical regardless of the application, a gate array manufacturer can produce uncommitted gate array wafers as standard, high-volume parts instead of as custom parts. Therefore, the manufacturer can afford to expend considerable effort on maximizing the yield and performance of the gate array chip. After a customer provides the definition of the logic blocks and interconnections for his application, the metalization and overglassing steps are completed and the circuit can be tested.

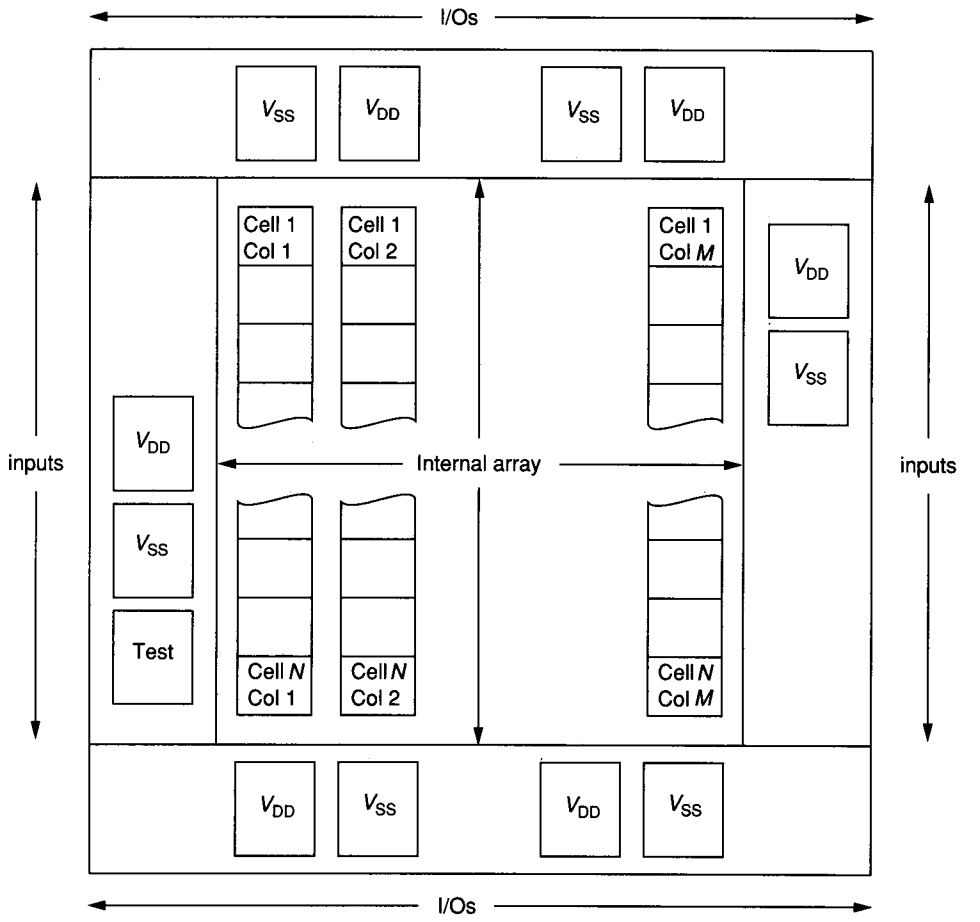


FIGURE 9.4-1
Topology for typical gate array.

Figure 9.4-2 shows that spaces or channels for interconnection wires form an important part of a gate array chip. Proper placement and sizing of the wiring channels are important in obtaining high utilization of the transistors within a gate array. If the wiring channels are too narrow, it may be impossible to interconnect all the logic circuits from different parts of the chip. For small, fixed wiring channels, this wiring problem may be minimized by leaving some of the potential logic gates unused in order to reduce wiring needs. This lessens the density of logic circuits used within the gate array, and thus wastes some of the transistor resources. If the wiring channels are widened, the interconnection wiring problem becomes simpler. However, with wide wiring channels, substantial integrated circuit area may be left unutilized in the wiring channels, increasing the cost of the final circuit. Between these two extremes, a compromise must be found. Wiring channels must be wide enough to allow acceptable transistor utilization for the gate array, without requiring excessive area for the channels. A gate array containing 6000 potential logic gates might have about 50% of

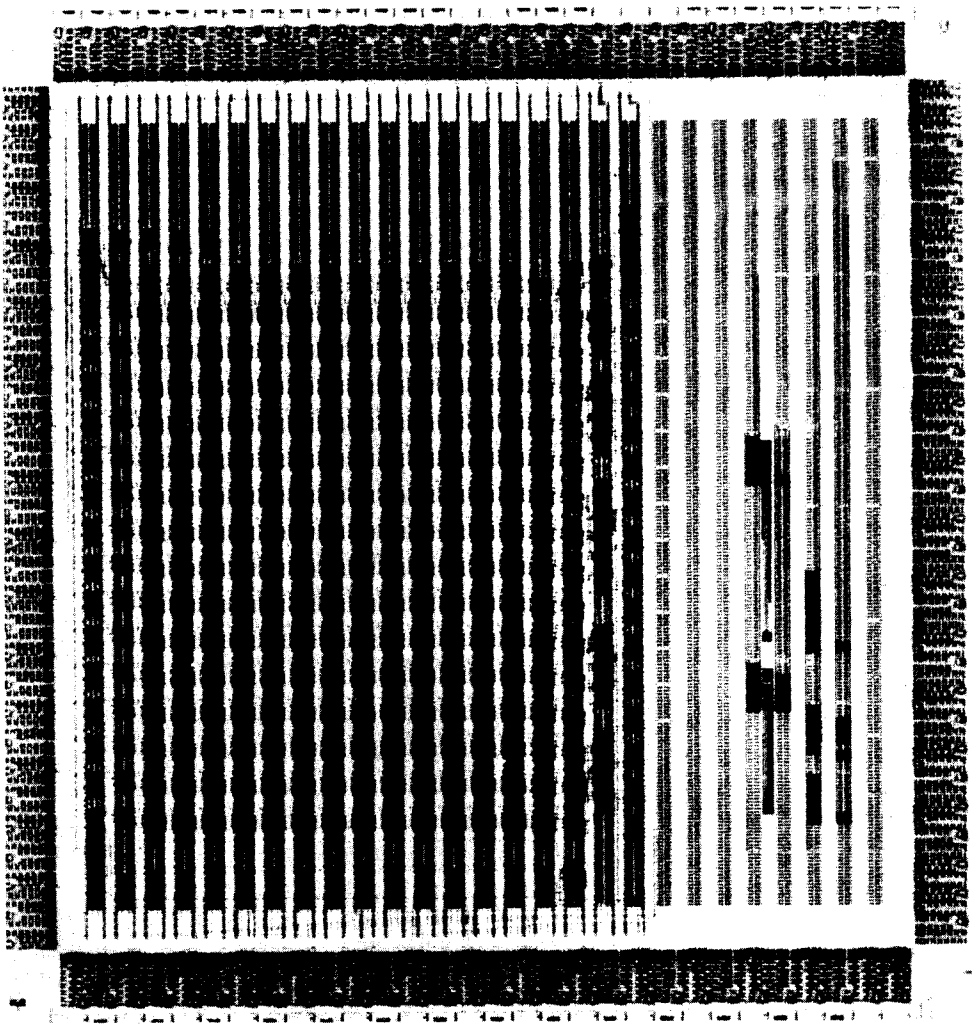


FIGURE 9.4-2
CMOS gate array chip (©IEEE 1982, Kobayashi et. al., ISSCC Proc., p. 316).

its area devoted to wiring channels, with a resulting gate utilization of 80% for a typical circuit design.

A recently introduced strategy to minimize the area dedicated to interconnections within gate arrays is to eliminate the transistor-free wiring channels. Instead, the gate array is completely patterned with transistor resources. To picture this possibility, consider Fig. 9.4-2 with each vertical wiring channel replaced by a column of transistor resources. The resulting structure is called a *channelless gate array*, or *sea-of-gates array*. In this structure, groups of transistors are interconnected to form logic building blocks as they are with the channeled gate array structure. Interconnection wiring is placed over selected transistor resources, rendering them unusable. With this strategy, logic building blocks can be created anywhere that interconnection wiring is not

required. The only area dedicated to wiring channels is that specifically required for interconnections. This minimizes the previously wasted area in channeled gate arrays where dedicated wiring channels were not fully utilized.

The sea-of-gates structure has two potential disadvantages. First, the capacitance of metal interconnections over the more heavily doped transistor source and drain diffusions is greater than the capacitance of metal interconnections over the lightly doped substrate. However, this increased capacitance per unit area is more than offset by the decrease in required interconnection area resulting from less restricted placement of interconnections in a sea-of-gates array. Second, the increased freedom in placement of logic blocks and routing of interconnections complicates the CAD tools that are used to place and route the gate arrays. This is a small price to pay for the increased utilization of die area and is rapidly being overcome by new programs designed for channelless gate arrays.

After a gate array is designed and fabricated by a particular manufacturer, the wiring channel size and transistor array characteristics are fixed. It might seem that a designer could simply provide logic block and interconnection definitions to finalize a digital logic design. This design would be implemented on the fixed gate array chip by interconnecting suitable logic blocks. An important step remains, however, before the design can be released for fabrication of the interconnection layers. The placement of individual logic blocks within the gate array must be specified before these blocks are interconnected. Unfortunately, arbitrary positioning of the logic blocks is unsatisfactory because the ability to properly interconnect the blocks depends heavily on their placement. Substantial effort has been expended on developing placement and routing algorithms that provide high density for the logic blocks while retaining the ability to interconnect those blocks through allowable wiring channels. Optimal placement and routing is an unsolved research problem, and many investigators are seeking improved placement and routing algorithms.[10,11]

The design process with logic gate arrays is less complex than custom integrated circuit design because the designer works at a higher level of abstraction. Manufacturers of gate arrays provide significant support to the logic designer with definitions of standard logic elements such as NAND, NOR, D flip-flop, latches, buffers, and compound logic gates. A typical list of logic elements available from a manufacturer is given in Table 9.4-1. Providing a set of logic blocks such as these supports a design style that is closely akin to TTL logic design with 74XX devices. This allows many of today's logic designers to use microelectronic circuits in their designs without mastering the details of MOS transistor circuit design.

A typical logic block might consist of three two-input NAND gates, as shown in Fig. 9.4-3. Each NAND gate uses two p-channel and two n-channel transistors to realize its logic function. Three of these gates are grouped into a single logic block, reminiscent of a TTL 7400, quad two-input NAND package. For a typical 3 μ gate array family, high-to-low and low-to-high propagation delays average 1.4 ns with no load and 4.6 ns with a 1 pF load. The 1 pF load is equivalent to a fan-out of three and includes a 100 mil length of metal conductor. A slightly more complex circuit, a D flip-flop, is shown in Fig. 9.4-4. This circuit requires two logic blocks and a total of 24 transistors for its realization.

TABLE 9.4-1
CMOS gate array library

Typical logic functions	
Triple 2-in NAND	2-2 O–A–I
Dual 3-in NAND	2-2 A–O–I
Triple 4-in NAND	D flip-flop
5-in NAND	D flip-flop with set, reset
Dual 2-in NAND/AND	2-to-1 mux
Triple 3-in NAND/AND	1-of-4 decoder with enable (act "L")
Triple 2-in NOR	2-bit magnitude comparator
Dual 3-in NOR	Mux D flip-flop with reset
Triple 4-in NOR	D flip-flop with preset, reset
5-in NOR	Toggle enable flip-flop with reset
Dual 2-in NOR/OR	D latch with reset
Triple 3-in NOR/OR	4-bit S–I/P–O SR
Triple clock buffer	Noninverting Schmitt
Quad inverter	1-bit ALU
Dual tri-state buffer	Full-adder
Tri-state noninv. buffer	4-to-1 data mux
EX-OR	4 bit parity checker
NAND latch plus 2-in NOR	4-bit S–I, P–I/P–O SR
Triple NAND latch	Presettable down counter with reset
NOR latch plus 2-in NOR	4-in mux with enable tri-state
Triple NOR latch	J-K flip-flop with set, reset

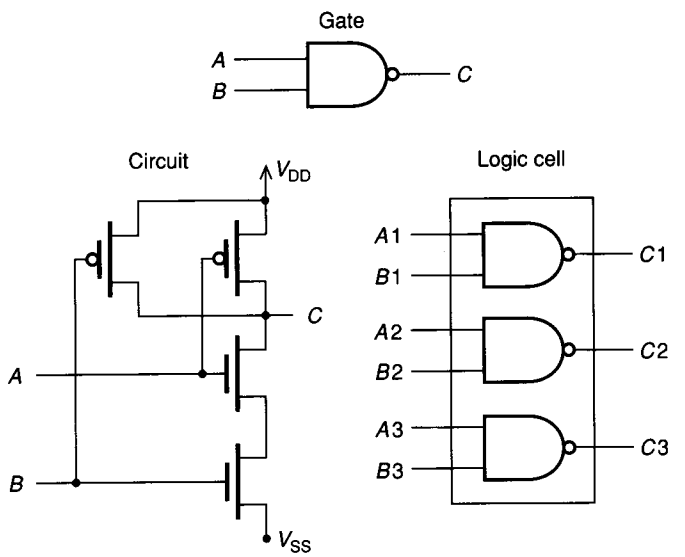


FIGURE 9.4-3
Triple two-input NAND logic block.

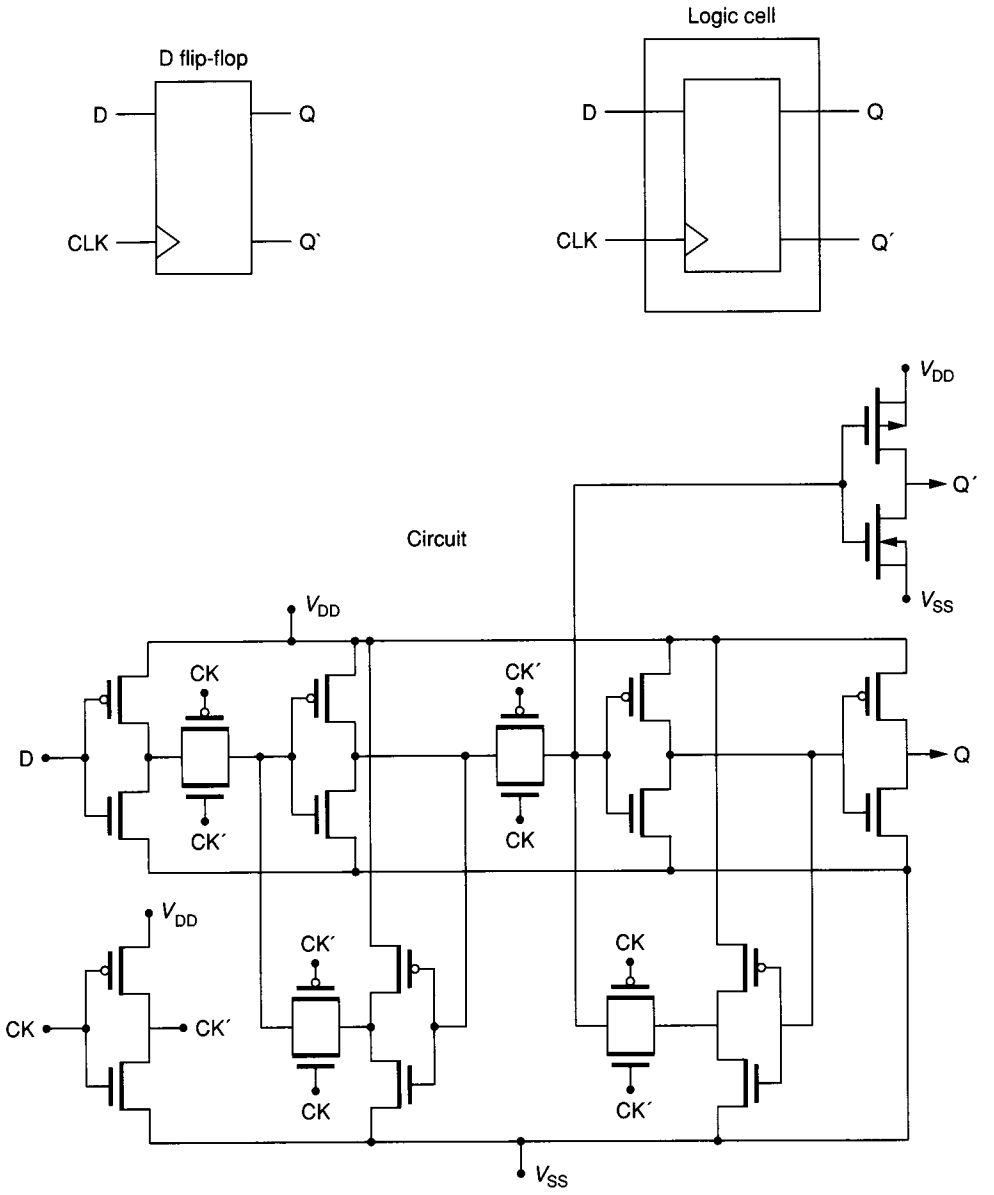


FIGURE 9.4-4
D flip-flop logic block.

Vertical metal interconnection between devices is accomplished through channels provided between each vertical row of blocks. Horizontal metal interconnection is accomplished through the blocks and around the ends of the rows. Ten horizontal wiring tracks are available per block for the typical circuits of Figs. 9.4-3 and 9.4-4. Most gate array manufacturers provide software tools to assist with placement of the logic modules and interconnection wiring. They also pro-

vide computer simulation tools for designers to verify the logical correctness and timing characteristics of proposed designs. There is interaction between placement and timing because interconnection lengths and, therefore, load capacitance change with logic module placements. Accurate simulation is a necessity for gate array designs as with all microelectronic circuits; it is not feasible to patch an integrated circuit with jumper wires, as is customary in correcting printed circuit board wiring errors.

Gate arrays are widely used to customize application-specific logic that would otherwise require many more IC packages. This results in considerable savings in area for many products. At least two levels of metal interconnect are generally required to implement designs with gate arrays. As the capability of technology has increased, the number of wiring layers for gate arrays has increased to three and even four layers of interconnect. Gate arrays are offered in sizes ranging from a few thousand gates to more than 100,000 gates at the present time.

In this section, logic gate arrays including both channeled gate arrays and the newer channelless, or sea-of-gates, structures were introduced. With gate arrays, logic designers can access the advantages of microelectronic circuits without becoming experts in the details of MOS circuit design. Both the global structure of gate arrays and examples of logic building blocks used in gate arrays were discussed. Finally, the requirement for CAD tools such as placement and routing programs, logic simulators, and accurate timing simulators was highlighted.

9.5 MOS CLOCKING SCHEMES

In preparation for the introduction of more complex digital systems containing storage devices and finite-state machines, the concept of clocking methods to control the movement of information is presented here. The combinational logic devices discussed previously do not require time-based control signals for their operation. The output of an ideal combinational logic circuit is completely defined at any time by the binary signals present as inputs to that circuit. For many applications, it is expedient to cause the output of a digital circuit to depend on both present and past inputs. For example, the output of a hand-held calculator in response to the "=" key depends on previous data and function inputs to the calculator. Digital circuits whose outputs depend on both present and past inputs are called *sequential* circuits; synchronous sequential circuits require a control signal to mark the passage of time and thereby delineate present inputs from past inputs. A digital signal called a *clock* serves this purpose by controlling the transfer of binary variables from one storage location to another.

An ideal clock signal is simply a periodic alternation of logic high and low voltage levels, as shown in Fig. 9.5-1*a*. Figure 9.5-1*b* shows a typical clock signal waveform as it might be observed on an oscilloscope. For 5 V logic, a single clock cycle is defined by

$$\text{clk} = 5 \text{ V} \qquad \text{for } 0 \leq t < t_1 \tag{9.5-1}$$

and

$$\text{clk} = 0 \text{ V} \qquad \text{for } t_1 \leq t < T \tag{9.5-2}$$

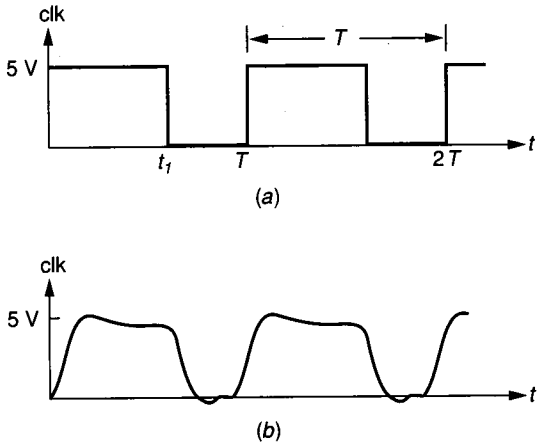


FIGURE 9.5-1
(a) Ideal clock signal, *(b)* Typical oscilloscope display of clock.

The time T represents the *period* of the clock signal; the inverse relationship $f = 1/T$ gives the *frequency* of the clock; and t_1/T is defined as the *duty cycle* of the clock. The square-wave output of a signal generator is an example of a clock signal with a 50% duty cycle. Circuits that operate from a single clock signal are said to use a *single-phase* clock. Much digital circuitry in the past was designed using single-phase clocks. Single-phase clocks could be used because of readily available binary storage devices (clocked flip-flops) that used one edge (either rising or falling) of the single-phase clock to update their stored value.

For reasons to be shown later, MOS logic circuits typically use *multiphase* clocks. Two-phase clocking schemes and four-phase clocking schemes derived from two-phase clocks are common. A *two-phase* clock is composed of two related sequences of alternating high and low voltage levels with the same frequency. A two-phase clock may be supplied to an integrated circuit from an external source or may be generated within the integrated circuit itself by special clock generation circuitry. Normally, a two-phase clock is composed of two nonoverlapping single-phase clock signals. Figure 9.5-2 shows nonoverlapping two-phase clock signals. The two-phase clock signals, ϕ_1 and ϕ_2 are said to be

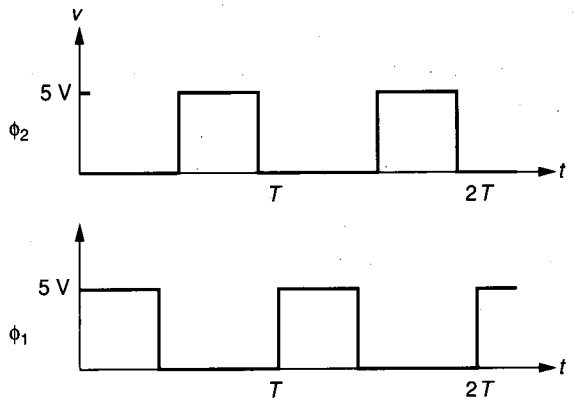


FIGURE 9.5-2
Nonoverlapping two-phase clock signals.

nonoverlapping because they are never both in the high state at the same time. This is represented by the logical AND function as

$$\phi_1 \cdot \phi_2 = 0 \qquad (9.5\text{-}3)$$

It is frequently desirable to derive a nonoverlapping two-phase clock from a single-phase clock signal. It might seem that this could be accomplished with the simple inverter circuit shown in Fig. 9.5-3*a*. However, a physical inverter circuit exhibits delay between its input and output signals, causing the derived two-phase clock signals to overlap when the input clock signal changes from a low to a high voltage level. This overlap condition is shown in Fig. 9.5-3*b*. The length of the overlap condition depends on the inverter delay. For practical inverter circuits, an overlap condition will always occur with this circuit.

Although a simple inverter cannot be used to generate a nonoverlapping two-phase clock from a single clock input, a slightly more complicated connection of simple logic gates will generate a nonoverlapping two-phase clock from a single clock input. The circuit in Fig. 9.5-4*a* generates a nonoverlapping two-phase clock with a nonoverlap time of at least one gate delay at each clock change. Figure 9.5-4*b* shows ideal waveforms for this circuit assuming identical, symmetric gate delays of length Δ for the inverter and the NOR gates. The reader should verify that the delays shown in Fig. 9.5-4*b* are correct. Asymmetric delays will change the time between clock edges, but will not cause overlap of the clock signals (see Prob. 9.12).

Clock waveforms have been presented in this section without restrictions on such clock characteristics as frequency, nonoverlap time, duty cycle, or rise and fall times. In fact, the clock signals shown have been idealized with zero rise and fall times. In practical circuits, clock signals are frequently required to drive a large number of gate inputs. The resulting capacitive loading can seriously degrade the rise and fall characteristics of clock signals, as was shown in Section

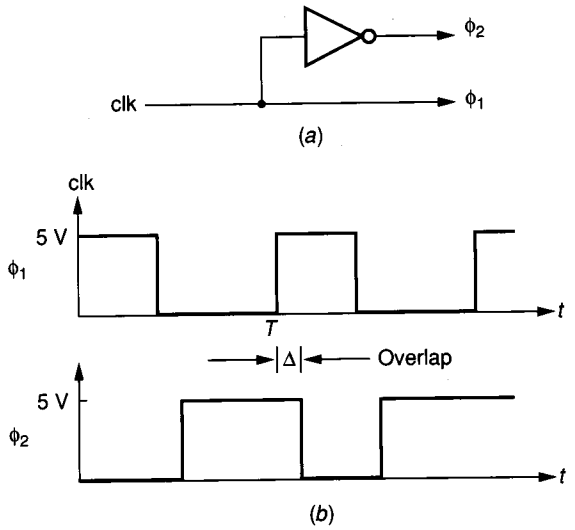


FIGURE 9.5-3
(a) Inverter used to generate two-phase clock, *(b)* Resulting clock waveforms showing overlap condition.

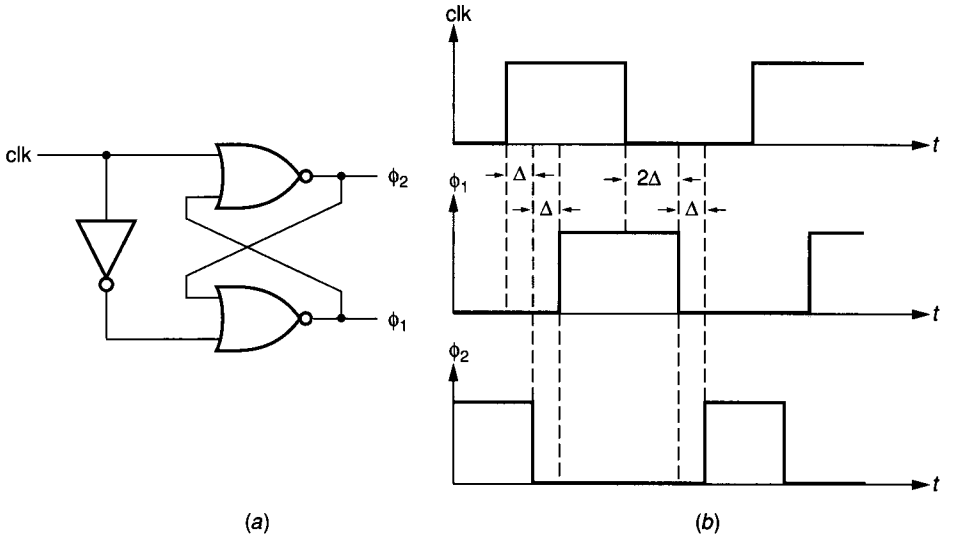


FIGURE 9.5-4
(a) Circuit used to generate a nonoverlapping two-phase clock from a single-phase clock input, *(b)* Ideal clock waveforms from the circuit of *(a)* assuming symmetrical clock delays.

7.9. To speed system operation and to minimize rise and fall times, clock signals are usually buffered and/or regenerated as they are used throughout the circuit. Exact device sizing for clock drivers depends on the size of the capacitive load and the speed with which the load must be driven. Techniques such as those discussed in Secs. 7.8 and 7.9 are used to calculate delays and set device sizes. Other characteristics of clock signals will be discussed in this chapter as they apply to the circuit under discussion.

9.6 DYNAMIC MOS STORAGE CIRCUITS

With the inverter, pass transistor or transmission gate, and multiphase clocks introduced in previous sections, the tools are in place to look at a useful storage mechanism within MOS circuits. This simple storage mechanism is termed *dynamic storage* and is widely used for momentary storage of data in digital circuits. The following subsections outline the structure and operation of dynamic MOS storage devices, particularly dynamic shift registers.

9.6.1 Dynamic Charge Storage

Among the technologies widely used for digital design, MOS technology provides two unusual features that lead to a particularly efficient way to store data momentarily. These two features are the MOS transistor's extremely high input resistance and the ability of an MOS transistor to function as a nearly ideal electrical switch. The circuit combination of these features with the source terminal

of one MOS transistor connected to the gate terminal of a second MOS transistor allows electrical charge to be stored momentarily on or removed from the gate terminal of the second transistor.

The three circuits of Fig. 9.6-1 show typical circuit configurations used to achieve dynamic charge storage. The pass transistor and transmission gate devices are often designed with minimum-size transistors to reduce layout area. The inverters are the simple inverters of Secs. 7.3 and 7.5 except for a higher sizing ratio k as explained in the next paragraph. Figure 9.6-1*a* is useful with NMOS circuits, while the other two are examples from CMOS circuits. Operation of the NMOS circuit will be explained to demonstrate dynamic charge storage, and then the changes required for CMOS will be noted.

Operation of the circuit of Fig. 9.6-1*a* depends on whether the pass transistor is off or on. If the gate of the pass transistor is at a high logic voltage, then the

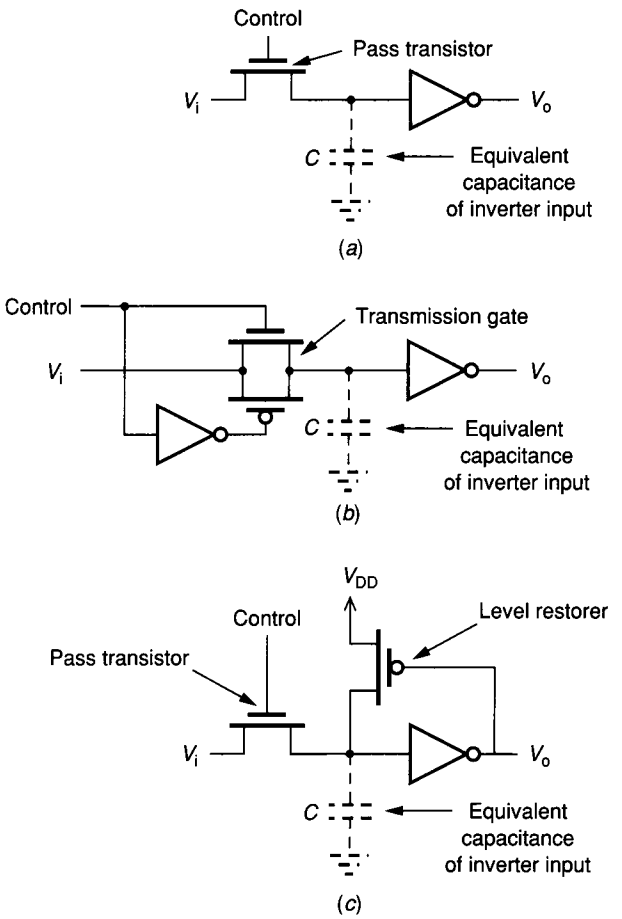


FIGURE 9.6-1
(a) NMOS dynamic storage circuit, *(b)* CMOS dynamic storage circuit, *(c)* CMOS pass transistor storage circuit with level restoration.

pass transistor conducts. In this case, the gate terminal of the inverter input transistor will be charged or discharged according to the logic voltage level at the input to the pass transistor. The time required to charge or discharge the gate terminal will depend on the gate capacitance, the pass transistor resistance, and the signal source. The gate can be discharged to 0 V, or can be charged to $V_{DD} - V_{TN}$. This sets the gate terminal to either a logic 0 or a logic 1 value, respectively. Because the inverter input voltage range has been reduced from the normal range of V_L to V_{DD} to a smaller range of V_L to $V_{DD} - V_{TN}$ by the pass transistor, the switching threshold voltage V_M should be lowered by increasing the inverter sizing ratio k from 4 to about 8. This can be shown through an analysis like that of Sec. 7.3.2.

When the gate of the pass transistor is at a logic low voltage, the pass transistor is off, thereby isolating any charge on the gate capacitance of the inverter input transistor. This charge (or the lack thereof) represents the stored logic value. If the stored charge were perfectly isolated, the logic value would be stored indefinitely. However, the isolation is less than perfect, primarily because of leakage through the reverse-biased diode created between the pass transistor source diffusion and the substrate. Leakage also occurs through the pass transistor switch. Because the stored charge will leak away over time, this circuit is termed a *dynamic storage circuit*. The following example examines the temporal characteristic of a typical dynamic storage node.

Example 9.6-1. For the NMOS circuit of Fig. 9.6-1*a* assume a pass transistor source diffusion area of 4 $\mu \times 5$ μ and an inverter input gate area of 9 μ^2. If the gate terminal capacitance is 1 fF/μ^2 and diffusion leakage current to substrate is 0.2 fA/μ^2, how long does it take for the stored voltage to change by 2.5 V?

Solution. The approximate capacitance is given by

$$C = 9\ \mu^2 \times 1\ \text{fF}/\mu^2 + 20\ \mu^2 \times 0.12\ \text{fF}/\mu^2 + 18\ \mu \times 0.2\ \text{fF}/\mu' = 15\ \text{fF}$$

and the leakage current is

$$I_r = 2\ \mu^2 \times 0.2\ \text{fA}/\mu^2 = 4 \times 10^{-3}\ \text{pA}$$

Then the time it takes to discharge the capacitance by 2.5 V is given by

$$t_{2.5} = 2.5C/I_r = 2.5\ \text{V} \times 15\ \text{fF}/4 \times 10^{-3}\ \text{pA} = 9.38\ \text{s}$$

This is clearly a long time compared to the clock periods of most digital circuits.

For dynamic storage with present MOS devices, the primary charge leakage path occurs through the diode between the source diffusion and the substrate. As MOS processes are created with linearly scaled-down devices, subthreshold leakage through the pass transistor's channel will become the predominant leakage factor.[12]

Dynamic storage can be implemented in CMOS by replacing the pass transistor with a transmission gate, as shown in Fig. 9.6-1*b*. Note the increase

in circuit complexity caused by the addition of the p-channel transistor in the transmission gate and the requirement for a dual-polarity control signal. This situation can be alleviated somewhat with the circuit of Fig. 9.6-1*c*. In this circuit, called a *level-restoring inverter*, an n-channel pass transistor is followed by a special inverter with a weak p-channel feedback transistor to restore the logic high level. The p-channel transistor must be sized to have an equivalent resistance much greater than the series pulldown resistance of the pass transistor and any circuit that drives the pass transistor input. The pass transistor can discharge the inverter gate to 0 V to give a good low logic level. In this case, the inverter output is high and the p-channel feedback transistor is off. As explained in Chapter 7, an n-channel pass transistor cannot raise the voltage high enough to ensure that the p-channel transistor of the inverter is off. Nevertheless, the pass transistor can pull the inverter input voltage high enough to force the inverter's output to a low logic voltage. This low voltage turns on the p-channel feedback transistor, thereby pulling the inverter input to the upper supply voltage and holding it there.

Dynamic storage is widely used within MOS circuits because of the simplicity of the required circuitry. The NMOS version of Fig. 9.6-1*a* requires only three transistors, while the CMOS version of Fig. 9.6-1*c* requires just four transistors. Thus, dynamic storage is area-efficient compared to the static storage circuits to be discussed later. A frequent use of dynamic storage circuits is to create shift registers. The following discussion shows shift registers that are built upon a generic MOS dynamic storage stage consisting of a pass transistor and a simple inverter for NMOS or a level-restoring inverter for CMOS.

9.6.2 Simple Shift Register

Figure 9.6-2 shows a multistage MOS shift register with each stage composed of a pass transistor and an inverter. The operation of this shift register can be described as follows based on a nonoverlapping two-phase clock. Assume that a logic signal is placed at the input of shift register stage A while the ϕ_1 clock

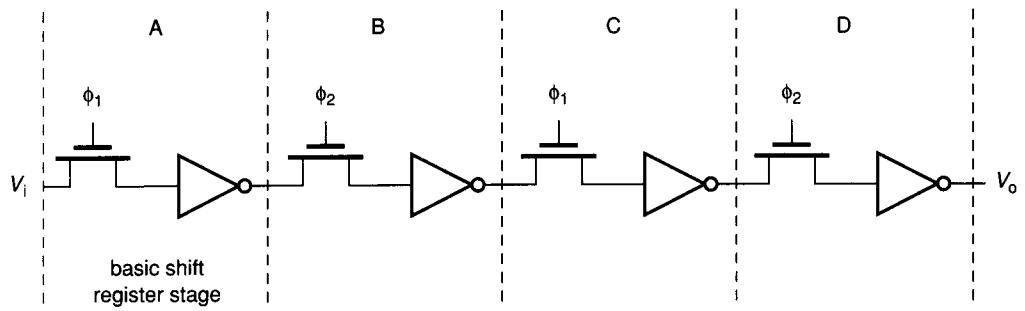


FIGURE 9.6-2
A linear shift register.

is low. Because the ϕ_1 clock is low, the pass transistor in stage A is off. Next, when the ϕ_1 clock goes high, if the signal at the input to stage A is held constant, it will be propagated to the input of inverter A. After a short delay, the output of inverter A will provide the inverted logic signal to the input of shift register stage B. At this time, the ϕ_2 clock is low and the pass transistor in stage B will not pass this input. When the clocks change so that ϕ_2 is high, the pass transistor of stage B will propagate the output signal of stage A to inverter B and then to the output of stage B. The signal will be stopped by the pass transistor of stage C because ϕ_1 is low while ϕ_2 is high. This sequence continues through the shift register chain as the clock signals alternate, causing the original input signal to propagate through the shift register stages.

At this point, a question should arise about the input to the inverter of shift register stage A when the ϕ_1 clock signal is low. In this instance, the input pass transistor of stage A is off, and the logic value is held by the dynamic storage of the input of the inverter. While the input to inverter A remains at its stored logic value, the output of inverter A will actively drive the input of stage B to the complementary logic level.

Each time the ϕ_1 clock changes to a high level, the shift register input signal will propagate to the gate of inverter A and on to the output of stage A. A sequence of alternating ϕ_1 and ϕ_2 clock signals will cause an input signal to propagate or shift through the structure at the rate of two stages of the shift register for each complete cycle of the clock signals. After N clock cycles, a logic input value will have shifted through $2N$ stages of the shift register. When a two-phase clock is used to control a shift register, it is important that the two clock phases do not overlap. If both phases of the clock were high simultaneously, a data value could propagate through multiple stages during the clock overlap time. This would result in uncontrolled operation of the shift register circuit and erratic movement of stored information.

Shift registers such as the one just described are used frequently within integrated circuits to provide temporary storage of digital signals. Such shift register storage can be used as a simple way to delay the arrival of a signal for a specific number of clock cycles. Shift register storage is also frequently used as the temporary memory for a sequential logic circuit. It will be shown later that a shift register can be combined with a PLA to provide a regular, expandable sequential machine. In general, shift registers provide dense, limited access memory for many applications within digital integrated circuits.

Figure 9.6-3 shows a parallel set of shift registers used to shift a group of signals in lock step fashion. As an example, such a parallel shift of 8, 16, or 32 data bits is sometimes required in microprocessor circuits. The basic structure of this set of shift registers demonstrates two principles important for the efficient geometrical layout of digital circuits. In Fig. 9.6-4, a symbolic layout diagram showing the geometrical topology for this circuit shows that the data for the shift register flows from left to right while the control signals (ϕ_1 and ϕ_2 clocks) flow from top to bottom. Such an orthogonal structure of data paths and control signals within a subsystem is widely used to provide a regular organization of logic

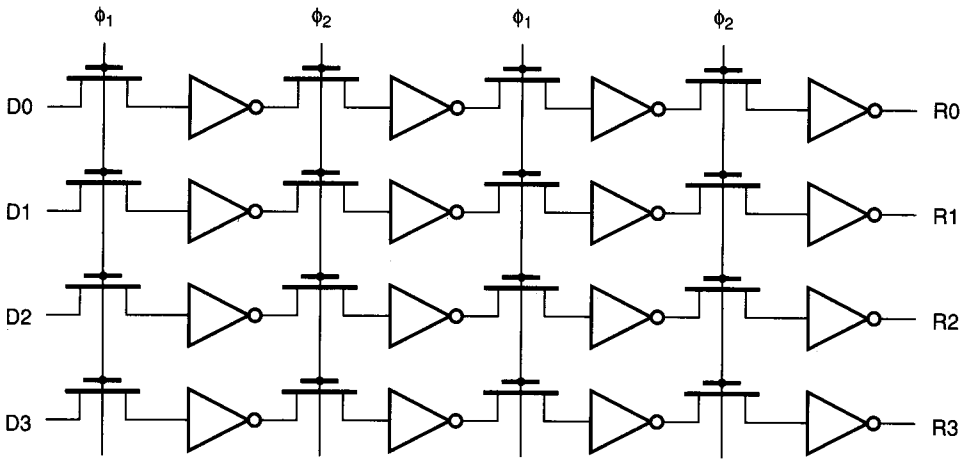


FIGURE 9.6-3
A parallel set of linear shift registers.

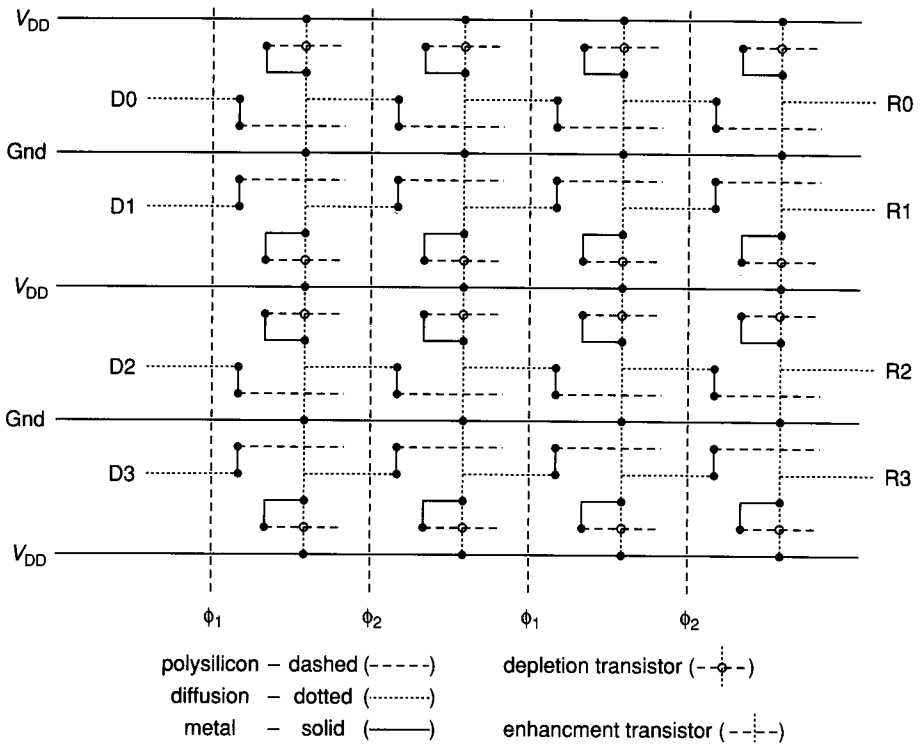


FIGURE 9.6-4
Symbolic layout diagram for a parallel set of linear shift registers.

circuits within an integrated circuit chip. The shift register stages are mirrored vertically about the ground and V_{DD} lines. This mirroring technique allows shared power and ground connections and reduces required circuit layout area. It is important to minimize the size of the basic shift register stage because this stage is repeated many times in a large shift register.

9.6.3 Other Shift Registers

A slightly different connection of the basic shift register cell is used to demonstrate another useful operation on a group of data signals. The need to shift the entire contents of a data word in one direction or the other is common in digital systems. If a data word is shifted toward the less significant bits, the equivalent binary weight of each bit is halved by each shift. If a data word is shifted toward its more significant bits, the equivalent binary weight of each bit is doubled by each shift. This doubling operation is useful in the conventional "shift and add" algorithm for binary multiplication.

Figure 9.6-5 shows a simple connection of shift register stages that allows a data word to be shifted toward higher significance or shifted directly along the data path according to a control signal. The control signal is ANDed with one

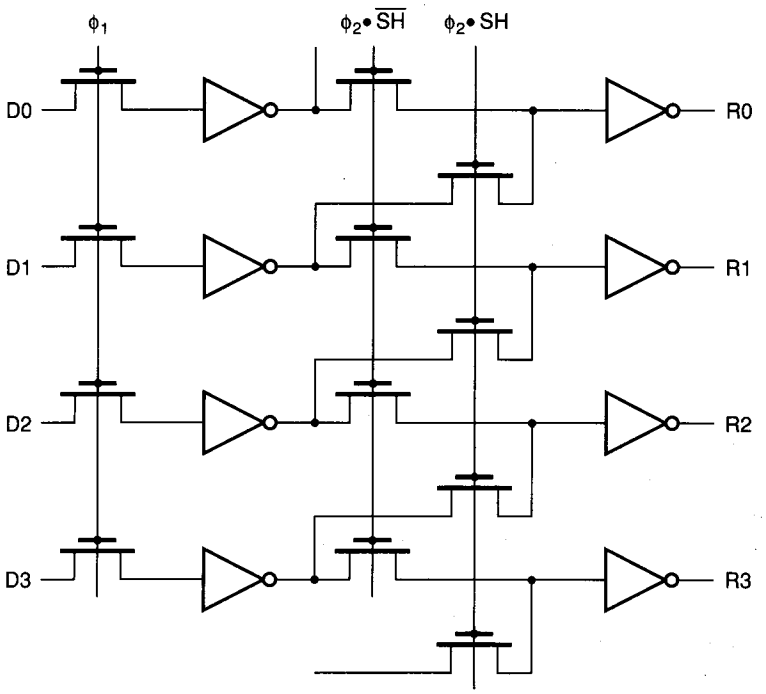


FIGURE 9.6-5
Four-bit shift-over, shift-up shift register.

clock phase to determine whether the data word is shifted or just passed along the data path. The layout of this circuit is similar to that of the parallel group of shift registers in Fig. 9.6-4 because the data and control signals are introduced orthogonally.

The purpose of this section has been to introduce dynamic storage, a most useful capability of MOS circuits. The ability to momentarily store logic values with minimal transistor and area requirements is widely used in digital ICs. The transient nature of the storage, typical storage circuits, and an application of dynamic storage to implement shift registers were all described.

9.7 CLOCKED CMOS LOGIC

Clocked logic in various forms has been used within digital MOS designs for many years.[13] Early use of clocked logic circuits was intended to minimize power dissipation in PMOS or NMOS logic. Present use of CMOS clocked logic circuits allows reduction of the number of transistors required within a design as compared with complementary static logic. Classical static CMOS logic gates require $2N$ transistors for an N-input gate, while NMOS logic gates require only $(N + 1)$ transistors. Clocked logic circuits for CMOS reduce the number of transistors to $(N + k)$ where k is a small constant overhead. This is accomplished by requiring dynamic storage of logic values within the gate structure (see Sec. 9.6). Clocked logic styles retain the desirable CMOS property of essentially zero static power dissipation. For this purpose transistors gated by clock signals, instead of complementary transistors gated by logic signals, are used to break the path between power and ground. Three styles of clocked CMOS logic are described here. The latter two have found wide application in large-scale digital circuits such as microprocessors and signal processors.

9.7.1 C²MOS

A dynamic shift register in CMOS is complicated by the need for a transmission gate and complementary clock signals rather than a pass transistor as described in Sec. 9.6. Figure 9.7-1 shows the four transistors and two clock signals required to construct a CMOS dynamic shift register stage. This construction can be

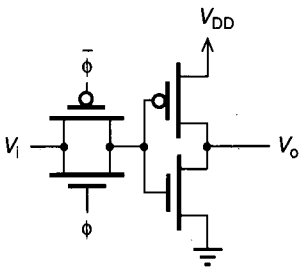


FIGURE 9.7-1
CMOS dynamic shift register stage.

simplified somewhat by the circuit configuration of Fig. 9.7-2. This form of dynamic shift register is called *clocked CMOS logic* or *C^2MOS*. In this circuit, the clocked transistors are placed in series with the p-channel and n-channel transistors of a standard inverter. The primary use of C^2MOS is within dynamic shift registers. All transistors can normally be sized as minimum-size devices because each stage is only required to drive the capacitance of an identical shift register stage.

Although the C^2MOS circuit requires the same number of transistors, external connections, and clock phases as the standard CMOS dynamic shift register of Fig. 9.7-1, the layout is simplified because the source/drain regions of the two p-channel transistors can be merged, and the corresponding regions of the two n-channel transistors can be merged. This reduces circuit capacitance, number of contacts, and layout area.

Operation of the C^2MOS circuit is quite simple. The gates of the p-channel pullup transistor and the n-channel pulldown transistor of the inverter are both connected to the input signal. For a valid logic input, one of these transistors will be off while the other is on. Clocked transistors placed in series with the pullup and pulldown transistors serve to connect these transistors to the output when the clock is high. For a high logic input, the output storage node will be discharged when the clock is high; for a low logic input, the output storage node will be charged when the clock is high. Otherwise, the output node will remain in its present state. In contrast to other clocked logic circuits, which are introduced in the following sections, the output of C^2MOS is available during the entire clock cycle, although it is actively driven only when the clock is high.

A problem with the C^2MOS circuit is that the load capacitance is the storage node for the dynamic charge. In the standard dynamic shift register, the storage node is inherently buffered from the output because the inverter gate is the storage node. Thus, the C^2MOS circuit is more susceptible to interference from the load circuit attached to the stage. If the load is an identical C^2MOS stage, the gates of the next stage can provide sufficient capacitance for the dynamic charge storage.

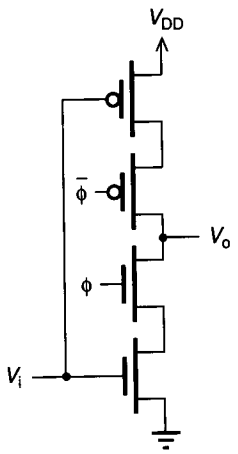


FIGURE 9.7-2
C^2MOS dynamic shift register stage.

9.7.2 Precharge-Evaluate Logic

A more general form of clocked logic, called *precharge-evaluate logic*, or *P-E logic*, provides low power dissipation like that obtainable with CMOS logic, and yet requires a transistor count comparable to NMOS logic. A basic tenet of such clocked logic is a tradeoff of output availability against power dissipation caused by the resistive pullup device of NMOS logic. If the path between power and ground is broken by two series transistors that are on at mutually exclusive times, no dc current path from power to ground will exist, nor will static power be consumed. Also, because there is no dc current path to place constraints on device sizing, minimum-size transistors can be used throughout to conserve layout area. The path to V_{DD} is used to precharge the output node during part of the clock cycle, and the path to ground is used to selectively discharge the output node during another part of the clock cycle. The output is taken high during the precharge time and is logically valid during the discharge cycle only after time is allowed to selectively discharge the output. Thus, for a square-wave clock signal, valid output availability is less than 50%.

The circuit of Fig. 9.7-3 shows a three-input NAND gate in P-E form. If the precharge transistor is a p-channel device and the discharge enable transistor is an n-channel device, a single clock signal will suffice. When the clock is in the low state, the p-channel transistor conducts and the output node is precharged to V_{DD}. When the clock signal goes to the high state, the n-channel transistor will enable discharge of the output node depending on the logic condition at the circuit's inputs. For the circuit of Fig. 9.7-3, the output is discharged only if all three inputs A, B, and C are in the high logic state. If any of these inputs is in the low logic state, the discharge path is broken and the output node is left charged to a logic high value. Thus, the gate realizes the NAND logic

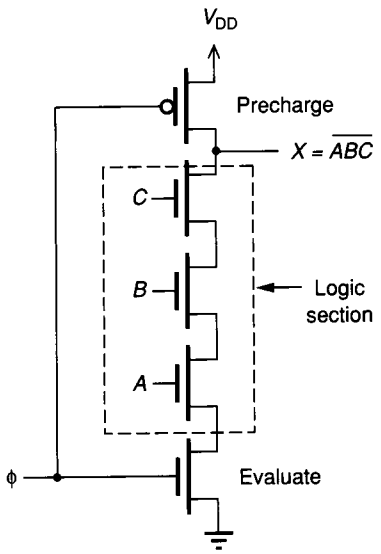


FIGURE 9.7-3
Three-input NAND gate in P-E form.

function. The P-E logic form allows realization of complex logic functions, as demonstrated in Fig. 9.7-4.

P-E CMOS logic has both advantages and disadvantages compared with classical static CMOS logic. In general, P-E logic requires less area than classical static CMOS logic because it does not require complementary transistor structures. The logic structure is ratioless, allowing use of minimum-size transistors throughout the gate logic. Because there is no dc pulldown current, a large number of transistors can be placed in series within the logic section. As another plus, it is possible for P-E logic to be faster than static logic because of lower gate loading on logic signals.

On the negative side, P-E logic has several disadvantages. The logic output value can be affected by a phenomenon called *charge sharing*. If a discharged node internal to the logic section is connected to the output node when the logic function is not satisfied, the output node charge will be shared with the discharged internal node, thereby degrading the output voltage level. Care must be exercised in circuit design to ensure that the output capacitance is larger than the internal node capacitances. P-E logic requires the addition of clock signals. There is a minimum clock rate because of the dynamic nature of the output signal, and the maximum clock rate is limited by circuit characteristics. The inputs must be stable during evaluation; otherwise, an incorrect value on the input could erroneously discharge the output node. Finally, the outputs must be stored during precharge if they are required during this phase of operation. These disadvantages are overcome by placing limits on allowable clock frequencies, and by careful selection of the types of circuits that are connected to P-E logic.

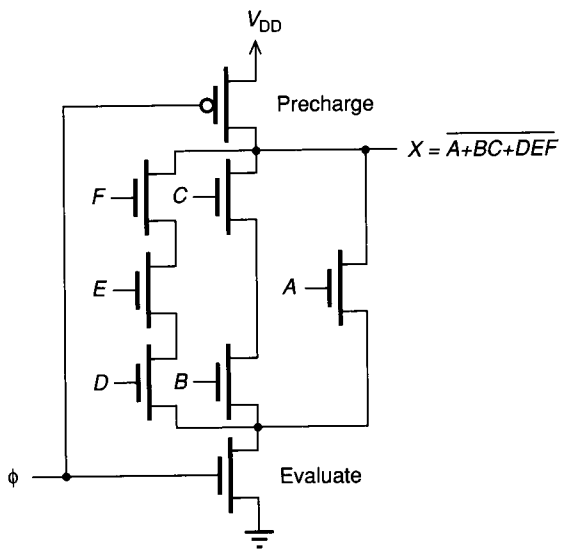


FIGURE 9.7-4
Complex logic gate in P-E form.

Multiple stages of P-E logic based on the same clock signal cannot be cascaded. Because the output of each stage of logic is driven to a logic high during the precharge phase, use of this output to drive secondary stages of P-E logic could erroneously satisfy their logic conditions immediately after the clock signal is pulled high. This could discharge the output of the secondary stage, preventing proper logical operation. A solution to this problem is to use cascaded stages with multiple clock signals so that the inputs to a stage are stable during its evaluation phase. Explanation of multiphase clock operation can be found in other sources.[14]

9.7.3 Domino CMOS

A variation on P-E logic, called *domino CMOS*, was popularized during the development of the BellMac microprocessor.[15] A domino logic gate consists of two elements: a P-E logic gate followed by a static inverter buffer at the output. The logic can be built in two forms: mostly n-channel, where the transistors comprising the logic are n-channel devices; and mostly p-channel, where the logic is performed by p-channel devices. The transistors used within the logic section can be minimum-size transistors. The static inverter at the output serves to buffer the logic part of the circuit from its output load, resulting in a more robust logic gate than standard P-E logic. The output inverter can be sized as desired, for example, to achieve symmetric output drive or to quickly drive a large capacitive load.

The behavior is explained based on the mostly n-channel form shown in Fig. 9.7-5. As with P-E logic, there is a precharge phase and an evaluation phase.

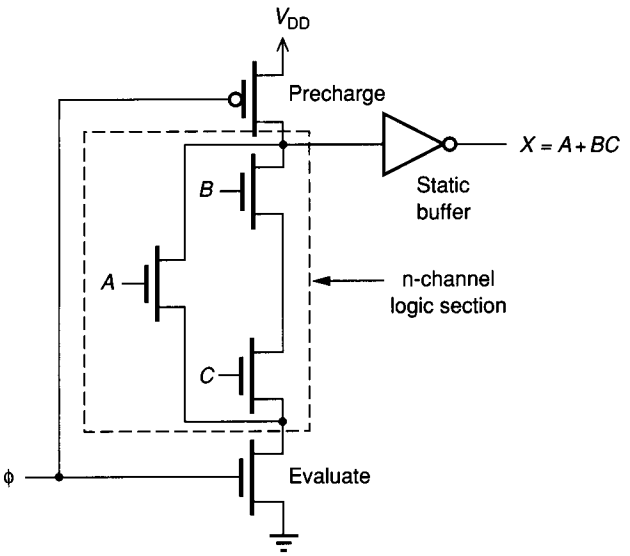


FIGURE 9.7-5
Domino CMOS logic gate.

During the precharge phase, the internal logic output is precharged to the high logic condition. This is inverted by the static buffer, providing a low logic output for the domino CMOS gate during the precharge phase. During the evaluation phase, the output of the logic part of the gate is selectively pulled low according to the logic input values. If the logic condition of the gate is satisfied, the internal output node is pulled low. This is subsequently inverted by the static buffer to provide a high logic output condition.

The domino CMOS gate has many of the same advantages and disadvantages described for P-E logic when compared to static logic. In addition, domino CMOS has advantages over the simpler P-E clocked logic form. For example, the static buffer provides output drive capability to either V_{DD} or ground. In P-E logic, the output can be driven only to ground in response to logical conditions, not to V_{DD}. When the logic condition of the P-E gate is not satisfied, dynamic charge storage at the output must maintain the high output value. The dynamic logic section of a domino CMOS gate always has a fan-out of one, thereby simplifying device sizing within the gate structure. As contrasted with P-E logic, domino CMOS stages can be cascaded successfully. The p- and n-channel transistors are easily grouped into a common n- or p-well, depending on the technology used. The fact that domino CMOS is a noninverting logic form provides at least one disadvantage over P-E logic. Lack of an inverting capability means that domino CMOS is not logically complete in the sense described in Sec. 7.2.

Examining the operation of the cascade of domino logic gates shown in Fig. 9.7-6 provides a basis to explain the choice of name for this clocked logic form. During the precharge phase with the clock signal near ground, the output of each domino stage is at the low logic condition. Thus, inputs to all subsequent domino stages are low. During evaluation, as the clock signal is pulled high, the outputs of some first-tier stages move to the high logic condition if their inputs are satisfied. The outputs of these gates may satisfy the logic for some second-

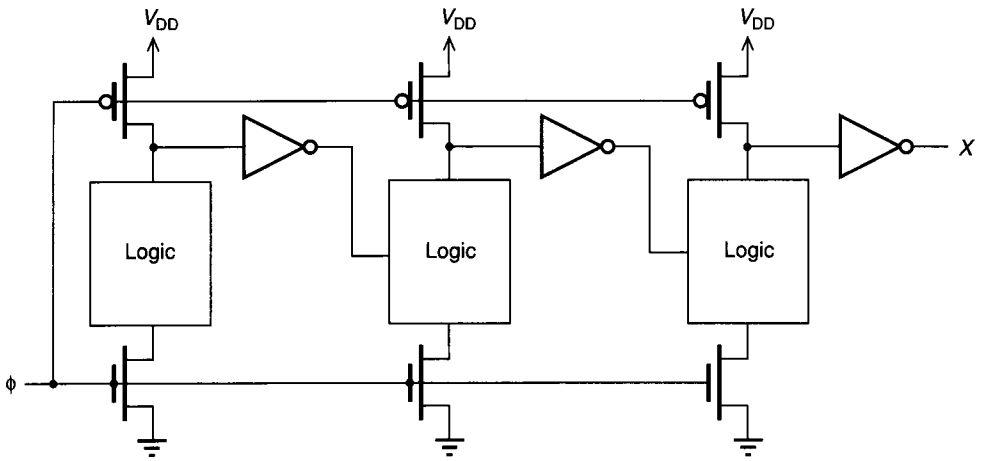


FIGURE 9.7-6
Cascade of domino logic stages.

tier stages, resulting in a high value at their outputs. In fact, during evaluation, logic decisions propagate through a cascade of stages like a falling domino chain.

As described, domino CMOS logic is a dynamic logic form. The precharged high at the input to the static inverter buffer is held by charge stored at the input to the static inverter and will not remain indefinitely. If this high could be maintained while the clock was stopped, a static logic form would result. The circuit of Fig. 9.7-7 shows a "keeper" transistor used to maintain a precharged high. This transistor can be formed as a weak static pullup device that contributes little to the pulldown current during evaluation or to static power dissipation. A weak static pullup is created by a large L:W ratio of 10 or 20:1 to increase the equivalent resistance of the transistor. This pullup transistor also improves noise immunity and allows a longer evaluation time.

Three styles of clocked logic were examined in this section. The first, C^2MOS, is useful primarily in shift registers. The latter two, P-E logic and domino CMOS logic are widely used for high-density, low-power implementations of logic equations. Both require only $N + k$ transistors, where N is the number of inputs and $k = 2$ for P-E logic and $k = 4$ for domino logic.

9.8 SEMICONDUCTOR MEMORIES

Integrated circuit memories provide the opportunity for semiconductor manufacturers to excel at their forte. That is, they create large, dense arrays of small cells with a process that is finely tuned for yield and performance characteristics. They strive to optimize the circuit design and the layout of individual memory cells to provide the maximum data storage capability for a memory part. Because the demand for dense semiconductor memory seems insatiable, a manufacturer has the opportunity to recover significant development costs with large sales volume

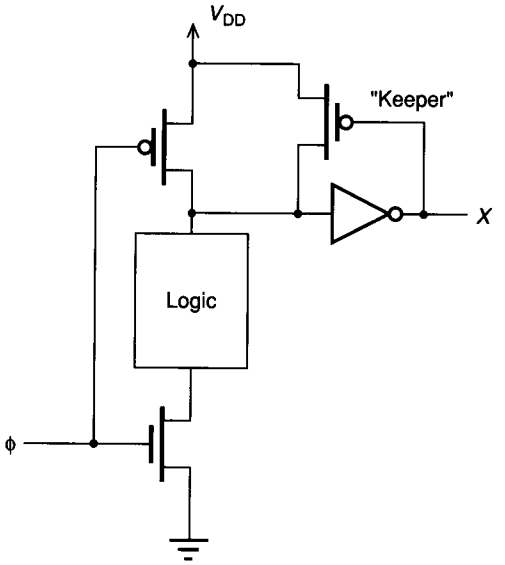


FIGURE 9.7-7
Use of "keeper" transistor to form static domino CMOS logic gate.

for a new memory design. With fierce competition for domination of the large market for memory devices, manufacturers constantly search for ways to create devices with smaller geometries and for clever circuit ideas that shrink the size of the basic memory cell or enhance its performance.

The excellent results of the research effort to provide small, dense semiconductor memories are rapidly incorporated into useful components of digital integrated system design. In many new microprocessors, ROM (read-only memory) partially replaces random logic in the control section. Designers of digital systems often find that dense memory is a good replacement for logic in other applications as well. Standard cell design libraries often provide forms of semiconductor memory as basic building blocks for systems. This trend toward increased use of memory requires digital system designers to be familiar with various types of semiconductor memory as tools for structured integrated circuit design.

9.8.1 Memory Organization

Semiconductor memories are universally organized as arrays of single-bit storage cells. These arrays are encircled by address decoding logic and interface circuitry to external signals. Figure 9.8-1 provides a block diagram of a typical memory chip organization. The memory array nominally uses a square organization ($m = n$ in Fig. 9.8-1) to minimize the external decoding circuitry necessary to select a particular memory cell. The reason for the square design can be seen by considering a memory part that contains 16k 1-bit storage cells. A memory array with 16k locations requires 14 address lines to allow selection of each bit ($2^{14} = 16,384$). If the array were organized as a single row of 16k bits, a 14-to-16,384-line decoder would be necessary to allow individual selection of the bit addressed by the 14 address lines. However, if this memory is organized as a 128-row by 128-column square, one 7-to-128-line decoder is required to select a row, and a second 7-to-128-line decoder is necessary to select a column. Note that a 128-row by 128-column matrix contains 16,384 crosspoints that allow access to individual memory bits. Thus, the square organization requires much less area for the address decoding circuitry than the linear organization.

Most memory chips operate such that the row address enables all cells along the selected row. The contents of these cells become available along the column lines. The column address is used to select the particular column containing the desired data bit. This data bit is ultimately routed to drive an output pin of the memory part. Some memory parts are organized so that n bits are accessed simultaneously. For these memories, the data from n columns are selected and gated to n data output pins simultaneously. Additional circuitry, including sense amplifiers, control logic, and tri-state input/output buffers, is normally required to create a functional memory part. However, the size of the memory storage cell and the resulting memory array are of primary importance in determining the size of the complete memory chip.

Several types of MOS semiconductor memory are in wide use today. These include ROM (read-only memory), EPROM (erasable programmable ROM), EEPROM (electrically erasable programmable ROM), SRAM (static random

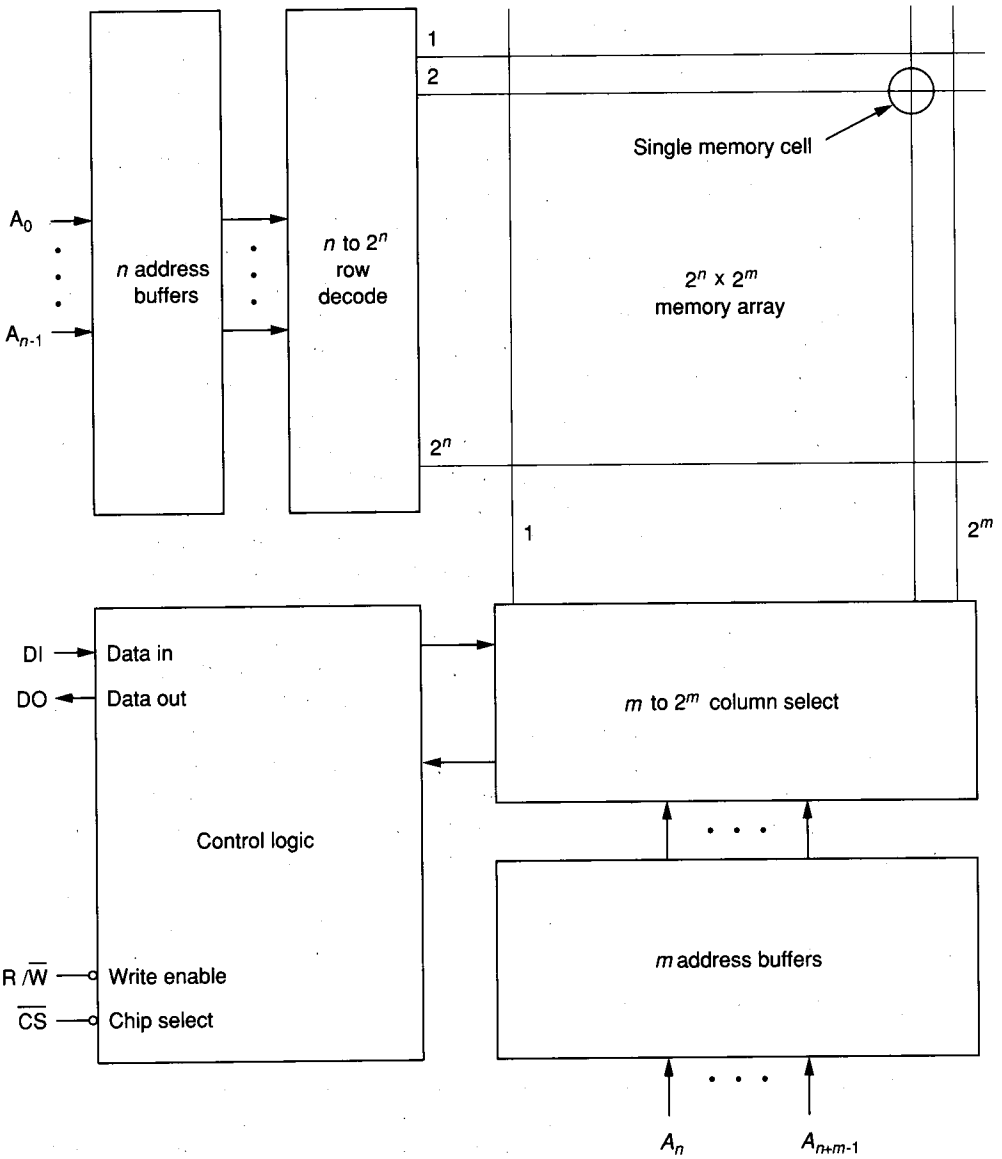


FIGURE 9.8-1
Typical memory chip architecture.

access memory), and DRAM (dynamic random access memory). These memory types derive their unique characteristics and advantages from the basic storage cell used in each, although the associated support circuitry will also vary. ROM-style devices will be discussed in the next section, and SRAM and DRAM memories will be addressed in two subsequent sections. Register array memories will be described in a fourth section.

9.9 READ-ONLY MEMORY

Read-only memory is the densest form of random-access semiconductor memory, using the presence or absence of a single transistor as the storage mechanism. Figure 9.9-1 shows the relationship between storage cells and the row and column lines of the memory matrix. Part of the memory address is decoded to select an individual row line and drive it to a positive voltage. Previously, each of the column lines had to be pulled to a high level. Each storage cell transistor has its gate connected to a row line, its drain connected to a column line, and its source grounded as shown by Fig. 9.9-1. Only if a transistor is placed where the selected row (corresponding to the row address) crosses a column will that column be pulled to a low level. Thus, each column line will be high or low to reflect the stored data along the selected row line. The remainder of the address lines are decoded to select the desired column or columns to provide the requested data.

The ROM memory contents are programmed by selectively placing transistors within the memory array. Therefore, the contents of a ROM memory are fixed as the part is manufactured and cannot be changed at a later time. In mass production, ROMs are the least expensive form of semiconductor memory; however, each unique ROM incurs relatively expensive start-up costs. As a result,

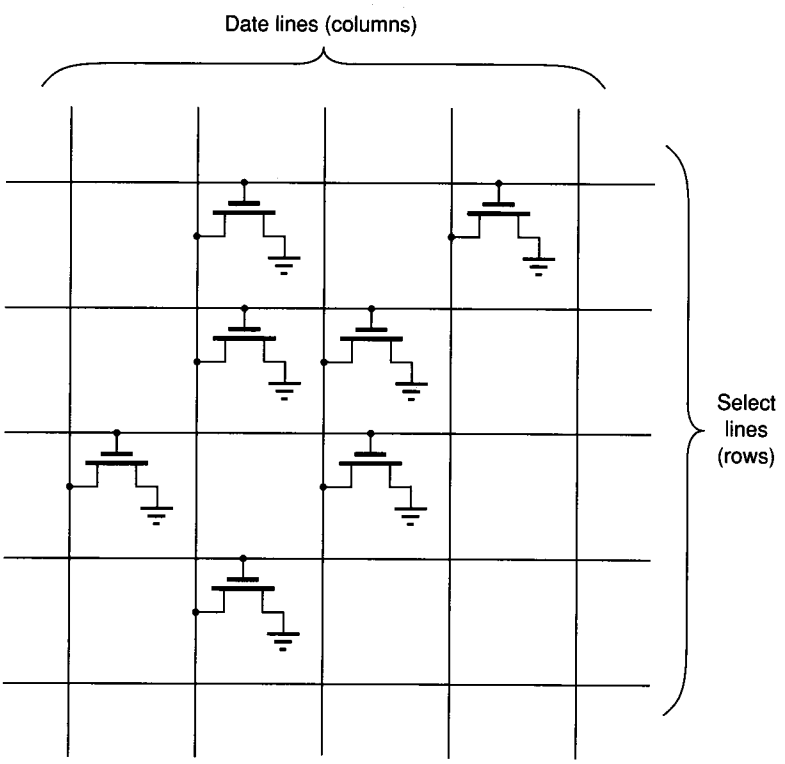


FIGURE 9.9-1
ROM memory array.

ROMs are normally feasible only for applications that require a minimum of several thousand identical memory parts with permanently stored data.

Each storage cell of a ROM may contain a single enhancement transistor. The size of the cell is set by the area required for the transistor and its associated row and column lines. An area of about 16 μ^2 per storage cell is required for today's ROMs. The memory array, requiring only enhancement transistors, can be fabricated in either NMOS or CMOS technology. The peripheral circuitry such as sense amplifiers, decoders, and control logic can also be fabricated in either technology. However, CMOS is usually chosen for newer ROMs to reduce static power dissipation of the peripheral circuitry.

9.9.1 Erasable Programmable Read-Only Memory

Many applications require semiconductor memory that is nonvolatile like ROM, yet can be reprogrammed to correct unintentional errors in the contents of the memory or to change program-based characteristics of a system. A *nonvolatile* memory is one that retains its stored data even while power is off. The ROM just described is nonvolatile and cannot be changed once it is manufactured. Other popular forms of semiconductor memory, such as SRAMs and DRAMs, have read/write capability but are *volatile*—that is, their contents are lost when power is lost. As a solution to this dilemma, the EPROM (erasable programmable ROM) was developed. An EPROM provides dense, nonvolatile storage yet can be reprogrammed as necessary. As a result, these memories are widely used in microprocessor systems and other circuits requiring nonvolatile storage where the cost of a unique ROM device cannot be justified.

The EPROM memories acquire their useful characteristics through use of a unique storage cell. This cell was originated by Intel Corporation and was called the *FAMOS* technology (for floating-gate, avalanche-injection, metal oxide semiconductor). Figure 9.9-2 shows that the storage cell consists of a transistor

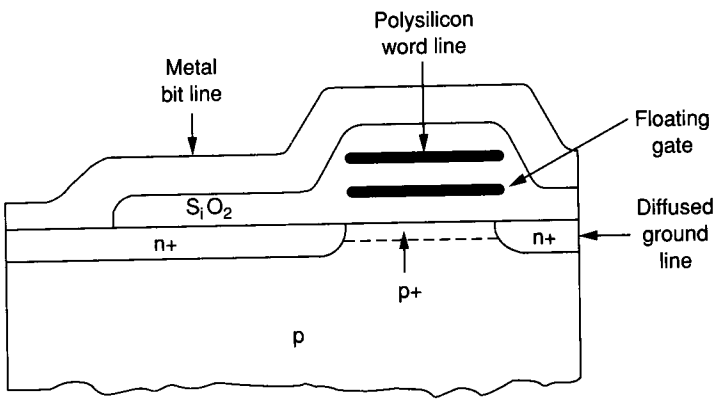


FIGURE 9.9-2
6×6 μ^2 EPROM storage cell.

with two gates, one of which is isolated from the circuit. If this floating gate somehow acquires charge to represent a stored data value, the charge remains there for a long period of time because the gate is insulated from its surroundings by silicon dioxide. The leakage paths for this circuit are of such high resistance that the time constant for the discharge path of the EPROM memory cell is tens of years. The presence or absence of stored charge is the mechanism by which the cell stores a data value, but how is the data value changed once it has been stored?

The mechanism for programming new data comes from the second part of the FAMOS name, *avalanche injection*. If a relatively high voltage (about 25 V—less for newer parts) is applied to the floating gate–substrate region, avalanche injection of electrons onto the gate takes place. This phenomenon serves to program memory by placing charge on the gate. Memory is erased by removing undesired charge from a gate as follows. If the floating gate is exposed to strong light of the proper wavelength (UV–2537 Å) for a period of time, enough energy is imparted to the stored charge to remove it from the floating gate. A quartz window is incorporated into the memory chip package; thus, all memory cells are exposed and erased simultaneously. Typical erase times for EPROMs are in the range of 20 to 30 minutes.

EPROMs are widely used in electronic systems where product volume is insufficient to justify the high initial cost of ROM parts. They are also used in prototyping microprocessor systems where reprogrammable but nonvolatile memory is required. A disadvantage of EPROM memory is the usual need to remove the memory part from the system if it becomes necessary to erase and reprogram the EPROM. The next section describes an improvement to the EPROM that was designed to overcome this undesirable characteristic.

9.9.2 Electrically Erasable Programmable Read-Only Memory

As useful as EPROM memories are, there are many applications requiring nonvolatile memory that can be reprogrammed quickly without removing the memory part from the system. For example, it is often desirable to program a standard CRT terminal with serial interface characteristics that will not be lost when power is disconnected. Yet, these characteristics must be changeable if the terminal is connected to a computer using a different serial data format. Applications such as this created a need for an EPROM whose contents could be changed electrically. Several manufacturers now offer memory parts called EAPROMs (electrically alterable) or EEPROMs (electrically erasable) that fulfill this need.

Interestingly, the EEPROMs solve the programming and erasure problems with two simple techniques. First, programming has been simplified by on-chip generation of the programming voltage. Instead of requiring an external connection to 25 V, a circuit called a *charge pump* is used to generate the necessary programming voltage from the standard 5 V supply. Second, instead of using ultraviolet light to erase the data, an internal connection is provided to reverse the electron injection phenomenon, allowing charge to be removed from the floating gate of the EEPROM.

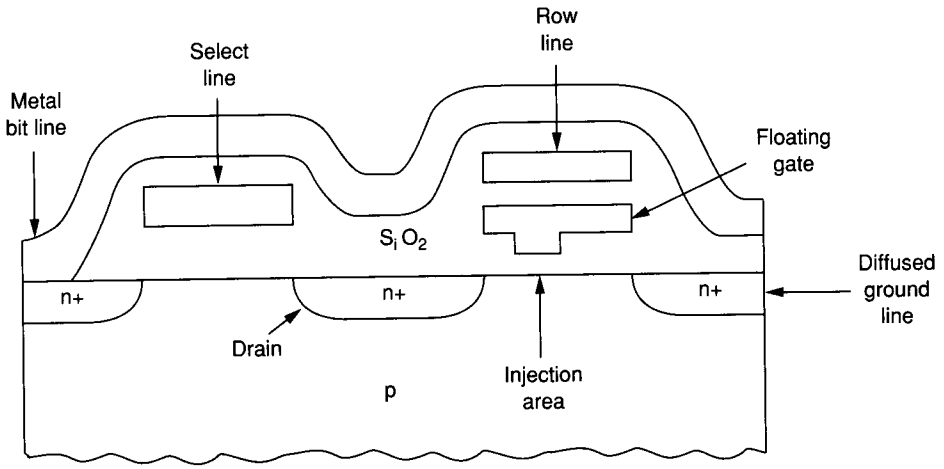


FIGURE 9.9-3
EEPROM storage cell.

The basic memory cell for an EEPROM consists of a memory transistor and a select transistor, as shown in Fig. 9.9-3. The memory transistor is composed of a dual-stacked polysilicon structure in which the bottom gate is floating. A small, thin oxide (<150 Å) isolates the floating gate from the drain and provides an injection area for electrons to and from the floating gate. Sending a short (several ms), high-voltage pulse to the row line while grounding the drain causes tunneling of electrons from the drain to the floating gate (erase). A similar pulse to the drain with the row line grounded causes tunneling of electrons from the floating gate to the drain (write). Integrating this device into a memory array requires an additional select transistor per bit as shown in Fig. 9.9-3 to avoid disturbing unwanted cells during erase or write. As a result, EEPROM memory is not as dense as EPROM memory.

The semiconductor memories described in this section each use the same basic array organization to achieve dense, nonvolatile storage. Ultimately, the storage capacity of these memory parts depends on the size of the basic memory cell used as the storage mechanism. The ROM is very dense, using only a single transistor as a storage cell; the EPROM is less dense, using a single transistor with a select gate and a floating gate as a storage cell; and the EEPROM is the least dense, requiring two transistors for each storage cell.

9.10 STATIC RAM MEMORIES

In this section the basic memory cell and organization used in static semiconductor memory circuits is examined as another example of a highly successful form of structured design. Static random-access memories, or SRAMs, are composed of static storage cells as the basic storage mechanism. Each static storage cell is formed by a pair of cross-coupled inverters. For SRAM memory, many of these cells are formed into a large memory array organized as explained in Sec. 9.8.

SRAMs differ from ROMs because they need continuous power to maintain the feedback required to hold a stored value. If power is lost, the active feedback path is eliminated, and the memory contents are destroyed. As power is restored, the memory cell will settle to a logic value that is independent of the previously stored data. Thus, the SRAM is classed as a volatile memory and depends on continuous power to maintain stored values.

For SRAMs to be useful as read/write memories, it must be possible to store desired data values in each cell. The basic memory organization must allow selection of each memory cell and accessing or storing binary values within that cell. A common structure consisting of select lines and data access lines for a SRAM memory cell is shown in Fig. 9.10-1. This figure shows a CMOS storage cell; a corresponding circuit structure is used for NMOS storage cells. A unique

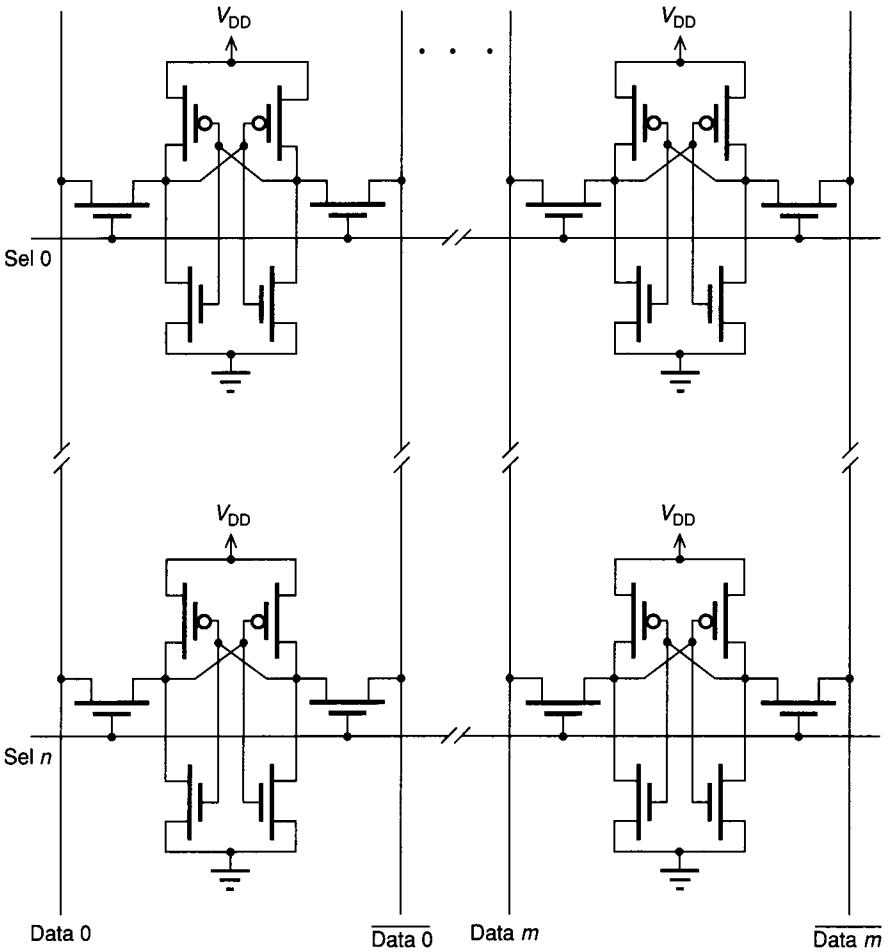


FIGURE 9.10-1
SRAM storage cell array.

select line is provided for each row of the memory array and, when high, selects all memory cells on its row. A pair of data lines are dedicated to each column to allow data from a selected cell on that column to be read or written. During a memory read, the two lines of each data line pair will be forced to opposite logic states by the contents of the selected memory cell. From a logical viewpoint, only one data line is needed to access the data within a memory cell. However, for a memory write, the two lines of the selected data line pair must be driven to opposite logic states to store a desired value within the memory cell. Because one cell along every column is selected by the row select, only data line pairs corresponding to the column containing the desired cell must be driven during a write. Other data line pairs along the row select will perform read operations.

As shown in the memory chip architecture of Fig. 9.8-1, n address lines are decoded to generate 2^n row select lines. The row select lines can be generated by the NOR decoder of Fig. 9.10-2, where $n = 3$. Here, the horizontal row lines are each pulled high by a pullup device. The pullup device could be a depletion transistor, a p-channel transistor with its gate grounded, or a clocked p-channel transistor, depending on the technology. The vertical address lines are converted

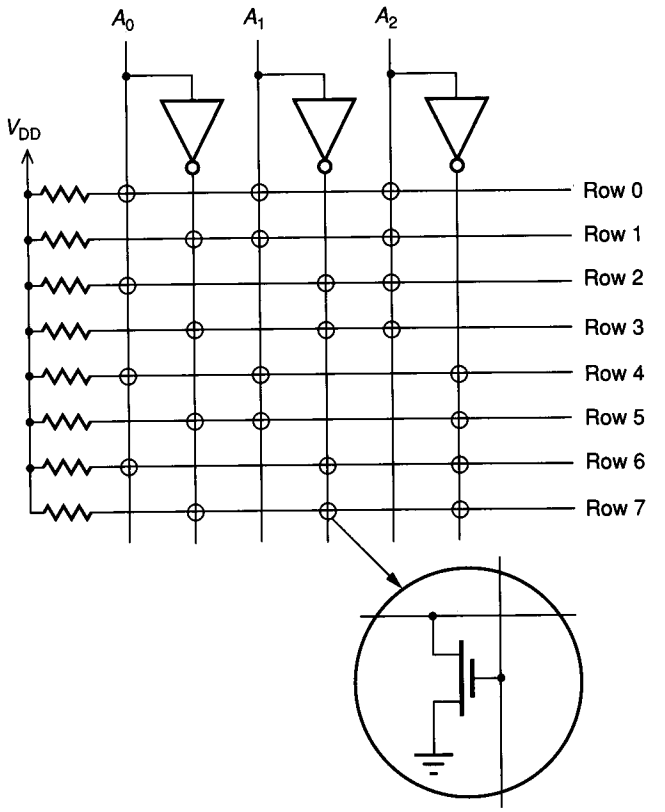


FIGURE 9.10-2
Address-to-row-select decoder.

to double-rail form and are used as the gates of pulldown transistors in the NOR structure. For example, row 0 of Fig. 9.10-2 is the logical NOR of A0, A1, and A2. If any of the three address lines is high, then the row 0 select line is pulled low. Only if A0, A1, and A2 are all low will the row 0 line be left high. Thus, the condition to select the row 0 memory cells is that A0, A1, and A2 must be low. To quickly charge the large capacitance inherent in the row select lines, a buffer stage (not shown) is usually inserted between the row decoder and the memory array.

Once a given row is selected, data from all memory cells on that row are available on the column data lines. The memory chip architecture of Fig. 9.8-1 shows that m address lines are used to select 1 of 2^m columns to access the desired data. Figure 9.10-3 shows an address-to-column selector circuit with $m = 3$ that gates one of eight columns to the data output. Address lines A3, A4, and A5 are used in a select array to choose the desired column. A3 selects the odd columns, while the complement of A3 selects the even columns. After the A3 stage, odd and even columns are paired because only one of the pair can be active. This reduces the number of available column paths by one-half. Subsequent stages of selection each divide the number of column paths by two. After m select stages, where 2^m is the number of memory columns, only a single column line is left. This line contains the desired data value. The data value must be buffered to drive the data output pin on the memory chip. To reduce the time between selection of a memory cell and availability of data at the chip output, sensitive

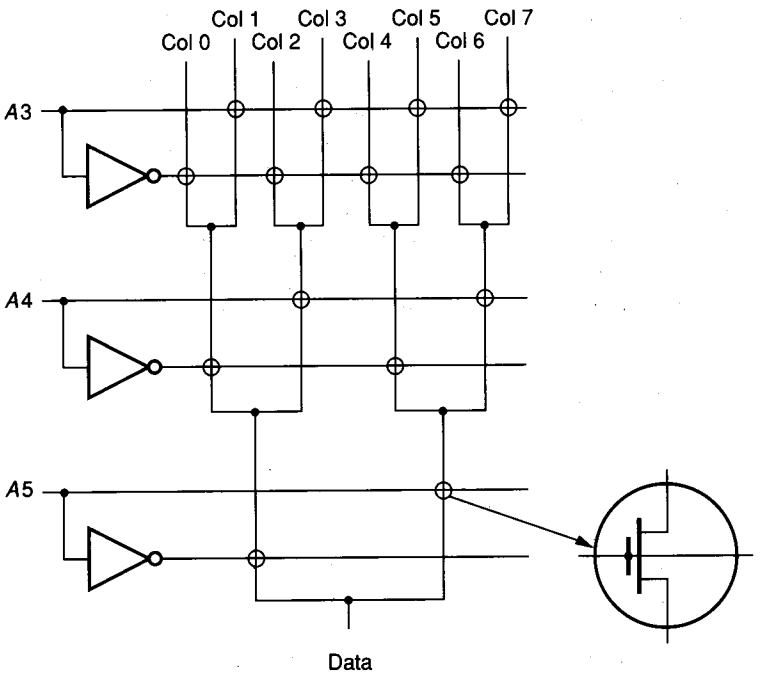


FIGURE 9.10-3
Address-to-column selector.

sense amplifiers (not shown) are included within the column select structure. The purpose of these amplifiers is early detection of the data value as the memory cell drives the large capacitance of its column line pair.

Electrical characteristics of the memory cells, data lines, and select lines are particularly important when a large array of memory storage cells must be operated at high speed. The select lines cross an entire array of memory cells where they drive the gate terminals for $2c$ transistors for a c-wide array. These lines are normally run in polysilicon instead of metal because they can directly gate the select transistors without a space-consuming contact at each select transistor. The metal level is reserved for the data lines. The length (and therefore area) of the polysilicon select line and the many gates to be driven provide a highly capacitive load for the select line drivers. As a result, careful consideration must be given to the delay caused by this capacitive load when the select line buffers are designed.

Example 9.10-1. Select line delay calculation Assume a 16k × 1-bit SRAM memory is organized as a square array of memory cells with 128 cells on a side. Further, assume that the actual memory array is 2 mm × 2 mm, the polysilicon select line is 2 μ wide, and the select transistor gates are $2\mu \times 2\mu$. The select line is driven from one end. The polysilicon select line resistance is 22 Ω per square. Capacitance to substrate for the polysilicon is 0.08 fF/μ^2 and gate capacitance is 1 fF/μ^2. Estimate the select line delay as the approximate 10% to 90% rise time at the far end assuming that the select line is driven by an ideal voltage step.

Gross solution. First consider a simple solution with the total select line resistance and capacitance represented as a low-pass *RC* filter. The resistance can be found by calculating the number of squares from one end of the select line to the other and multiplying by the resistance per square for polysilicon. A 2 mm × 2 μ line is 1000 squares long. The capacitance can be calculated from the area of the polysilicon select line and the area of gates for the select transistors. The area of the select line less the gate area is (2 mm−256 × 2 μ) × 2 μ or 2976 μ^2. The total area of the gates will be 2 × 128 × 4 μ^2 or 1024 μ^2.

$$C_T = 2976\ \mu^2 \times 0.08\ \text{fF}/\mu^2 + 1024\ \mu^2 \times 1\ \text{fF}/\mu^2$$

$$C_T = 0.238\ \text{pF} + 1.024\ \text{pF} = 1.262\ \text{pF}$$

$$t_{10\%} = 0.105\tau = 0.105 \times 1.262\ \text{pF} \times 22{,}000\ \Omega = 2.92\ \text{ns}$$

$$t_{90\%} = 2.303\tau = 2.303 \times 1.262\ \text{pF} \times 22{,}000\ \Omega = 63.94\ \text{ns}$$

$$t_d = t_{90\%} - t_{10\%} = 63.94\ \text{ns} - 2.92\ \text{ns} = 61.02\ \text{ns}$$

Distributed lumped-parameter solution. For this solution, break the polysilicon line into 10 segments with the resistance and capacitance divided equally among the 10 segments. A SPICE simulation (refer to Chapter 4) can be used to find the 10% to 90% delay times. The results are given below.

$$t_{10\%\ \text{voltage}} = 4.39\ \text{ns}$$

$$t_{90\%\ \text{voltage}} = 33.77\ \text{ns}$$

$$t_d = t_{90\%} - t_{10\%} = 33.77\ \text{ns} - 4.39\ \text{ns} = 29.38\ \text{ns}$$

The lumped parameter solution with 10 equal segments is very close to the exact solution. The simple *RC* method presented first can be used to give a worst-case estimate of the delay at the end of the select line. (It more than doubles the actual delay in this example.) In fact, it has been shown that an estimate of half the simple *RC* delay is a good approximation for a distributed *RC* line.[16]

The delay characteristics of the data lines are also of concern because they traverse the entire memory array in the vertical direction, providing a large capacitive load. These lines are usually run in metal rather than diffusion because they must cross the polysilicon select lines. Unfortunately, during a read operation the memory cells themselves must drive the data line capacitance. Special line-driver circuits are not available to overcome this speed limitation because their size prohibits providing a line driver for each memory cell. The memory cells are designed for minimum size to increase the overall memory density, and they cannot provide good capacitive drive characteristics. As may be seen from Fig. 9.10-1 the selected SRAM storage cell must drive the complementary data lines of the cell column in opposite directions. Thus, the limiting delay condition is for the data line that must be driven high where the memory cell p-channel pullup device must charge the data line capacitance. For NMOS cells, if the pullup transistor provides a low resistance path to V_{DD}, the memory array will dissipate an undesirably large amount of power because one inverter of each memory cell always conducts. Providing a high resistance for the pullup transistors would cause an unacceptable delay in charging data line capacitance. A typical resolution of this conflict is presented next.

A common method for minimizing the pullup time for a highly capacitive line driven by ratio-type circuits is to pull the line to a voltage above the logic threshold voltage when it is not in use. This technique is called *precharge* because the line is precharged to a value at or near the high logic condition. For this scheme to work, it must be possible to isolate the line from any driving sources during the precharge time. Such sources can be isolated from the line with pass transistors as they are in a memory array. When the capacitive line is to reflect the logic condition of a driving circuit, the corresponding pass transistor is turned on. If the driving circuit has a high voltage output and the line has been precharged to a high level, the correct logic output is available immediately. If the driving circuit has a low logic output, the pulldown transistor must discharge the output line before the correct logic value is available. Because the resistance of the pulldown transistor has little effect on power dissipation, the driving circuit can have a low-resistance pulldown transistor, allowing the discharge time to be shortened. The use of precharged lines provides a means of bypassing the asymmetric drive characteristics of a ratio logic output stage for situations where a high-capacitance line must be driven. The precharge scheme is often used for buses that must be driven by many sources.

Example 9.10-2. Optimum precharge voltage for data lines Consider that an optimum precharge voltage level might exist for a capacitive line driven by ratio logic. That voltage level would equalize the charge and discharge times of the output line for a given driving circuit with a pullup/pulldown ratio R_u/R_d. Also, assume

high and low logic levels of V_h volts and V_l volts, respectively. What should the precharge voltage level V_p be for fastest symmetrical operation of a read?

Solution. To reach a high voltage level, the data line voltage as a function of time is

$$V(t) = V_{DD} - (V_{DD} - V_p)e^{-t/R_uC_L}$$

For the time to reach V_h, solve for t_h as

$$t_h = R_uC_L \ln \frac{V_{DD} - V_p}{V_{DD} - V_h}$$

To reach a low voltage level (ignore the small effect of R_u), the data line voltage as a function of time is

$$V(t) = V_p e^{-t/R_dC_L}$$

For the time to reach V_l, solve for t_l as

$$t_l = R_dC_L \ln \frac{V_p}{V_l}$$

Setting $t_h = t_l$ gives

$$R_u \ln \frac{V_{DD} - V_p}{V_{DD} - V_h} = R_d \ln \frac{V_p}{V_l}$$

Letting the pullup/pulldown ratio be $S = R_u/R_d$ and solving for S gives

$$S = \frac{\ln V_p/V_l}{\ln [(V_{DD} - V_p)/(V_{DD} - V_h)]}$$

as the relationship between S and V_p.

For a ratio-type memory cell, the pullup/pulldown ratio is set higher than the normal value for logic gates to minimize power dissipation in the memory cell. Assuming that $S = 10$, $V_{DD} = 5$ V, $V_h = 4$ V, and $V_l = 0.5$ V, then $V_p = 3.78$ V. In practice, S will be higher than 10 and V_p should be closer to V_h.

The basic cross-coupled SRAM storage cell has several variations. Figure 9.10-4*a* shows a depletion-load cell for an NMOS technology. Many newer static memory circuits use a polysilicon load resistor to form the circuit of Fig. 9.10-4*b*. This requires an additional mask step to provide a lightly doped polysilicon with a resistance of 100k to 1MΩ/□. If a high-resistance polysilicon pullup of minimum size is used, the cell size is reduced by elimination of the depletion transistor and its associated gate-to-source connection.

Another important structure for a static RAM uses CMOS inverters to implement a basic memory cell with extremely low quiescent power characteristics. Were it not for the size disadvantage of the CMOS cell, this cell would be the overwhelming choice for static RAM memories. However, because of the necessity to implement both p- and n-channel transistors and the corresponding n- or p-well spacing requirements, the CMOS memory cell is larger than its depletion- or resistive-load counterpart. Even with this disadvantage,

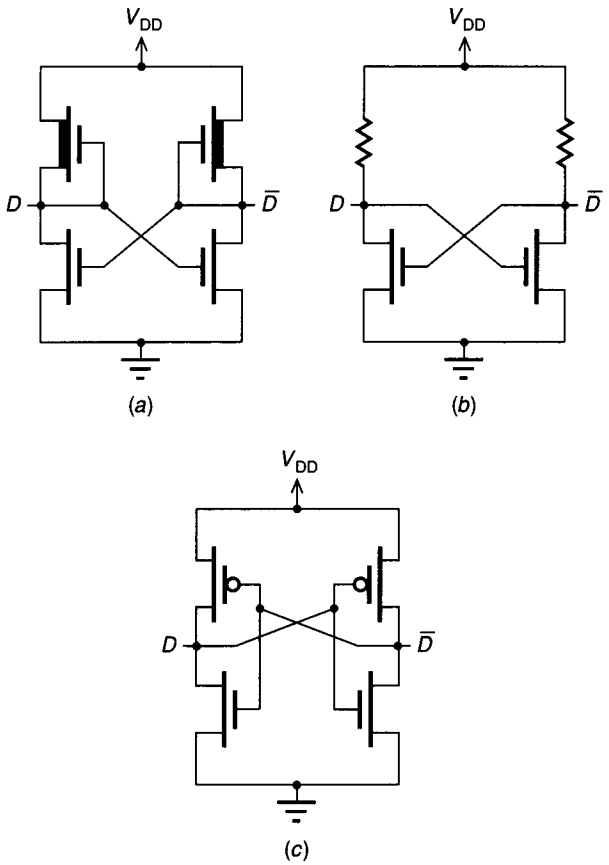


FIGURE 9.10-4
Static RAM cells: *(a)* Depletion load, *(b)* Polysilicon resistor load, *(c)* p-channel load.

many new static RAMs are built with CMOS to reduce power dissipation. The typical CMOS memory cell structure of Fig. 9.10-4*c* can be compared with the similar NMOS cell structure shown in Fig. 9.10-4*a*. The operation of the two cells is identical except that the CMOS cell dissipates negligible power because one of the series transistors from V_{DD} to ground is always turned off. The overall organization of a static memory is the same independent of the type of load device.

Unfortunately, SRAM memory is not as dense as the ROM memory types described earlier because a typical static RAM storage cell requires six transistors. The read-only memory cells of Sec. 9.9 required only one or two transistors per cell. Even when manufacturers replace the depletion pullup transistors with high-resistance polysilicon resistors, the SRAM memory cell still requires four transistors plus the polysilicon resistors. Another type of fast read/write memory, which requires only a single transistor and capacitor for a storage cell, is also available. This memory is described in the next section.

SRAMs are the fastest read/write semiconductor memory in wide use today. A speed advantage over DRAMs offsets the higher density and lower cost per

bit of DRAMs in many applications. For example, SRAMs are widely used in high-speed cache memories for modern computer systems. This section has shown the characteristic cross-coupled inverter structure used for SRAM cells. Additionally, examples showing operation of the highly capacitive row select and data bit lines were considered.

9.11 DYNAMIC RAM MEMORY

The dynamic RAM (DRAM) form of integrated circuit memory has surpassed all other random-access read/write memories in the number of cells or bits that can be placed on a memory chip. A DRAM memory circuit uses charge storage on a capacitor to represent binary data values. A few transistors (first three and now just one) are required to select the cell and access the stored data. Because SRAM memory requires more transistors per memory cell (either four or six, depending on how the pullup for the cross-coupled inverters is implemented), SRAM cannot be manufactured with the high memory density of DRAM. Historically, DRAM chips provide a ratio of about 4 to 1 in the number of memory cells provided relative to SRAM chips for the highest-density memory chips of each type. The DRAM memory array, requiring only enhancement transistors, can be fabricated in either CMOS or NMOS technologies. The peripheral circuitry such as decoders, selectors, sense amps, and output drivers can also be designed for either technology. Most new DRAMs are designed for CMOS processes to minimize power dissipation in the peripheral circuitry.

Dynamic RAM gets its name because the charge stored on the capacitor cell leaks off with time, causing the stored value to be dynamic. If a logic state is represented by a high voltage level on the capacitor cell, this voltage level decreases for a p-well or p-type substrate device because of various leakage paths until the value is indeterminate or changes to the complementary state. Conversely, for an n-well or n-type substrate, the cell voltage increases with leakage. The dynamic nature of this storage mechanism is described more fully in Sec. 9.6. To prevent loss of data, the voltage on the capacitor cell must be sampled and restored within a specified time period. This sample-and-restore operation is called a *memory refresh*; it takes additional external circuitry to ensure that all memory cells are refreshed periodically. A value of 2 ms is a typical specification for the maximum time period between refreshes for DRAM memories.

At one time, most DRAMs were manufactured using a three-transistor cell. This cell, shown in Fig. 9.11-1, is based on charge stored on a capacitor with one transistor acting as a buffer to drive the read data line, one transistor acting as a read-select switch, and a third transistor acting as a write-select switch. All transistors are minimum or near-minimum size to reduce layout area. The three-transistor cell requires four bus lines for operation. These bus lines include separate read and write selects and corresponding read and write data lines like those shown in Fig 9.8-1. Providing a buffer transistor to drive the data line during a read operation prevents degradation of the stored charge during a read operation. However, the charge on the capacitor must still be refreshed periodically because

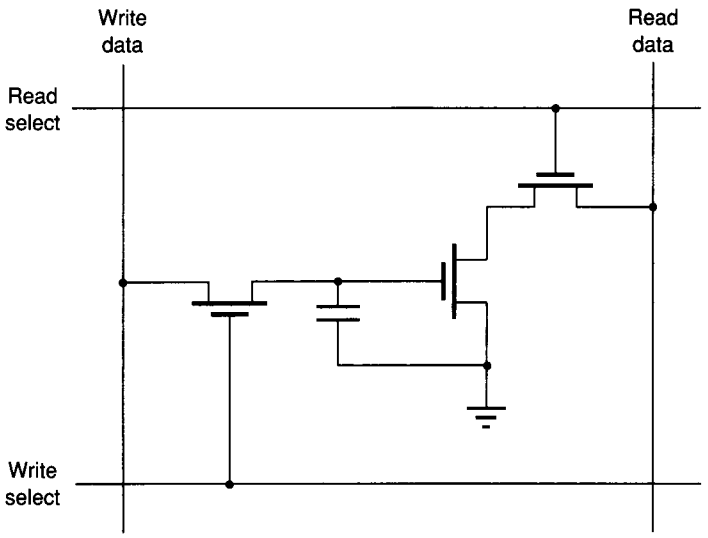


FIGURE 9.11-1
Three-transistor dynamic RAM cell.

of the leakage problems mentioned earlier. The memory refresh is performed by executing a read operation followed by a write operation. The three-transistor cell is robust with respect to the read operation because the stored charge is isolated by a buffer transistor.

Further search for increased memory density brought about the one-transistor DRAM cell. A typical cell with a single select transistor and capacitor for charge storage is shown in Fig. 9.11-2. The single transistor is a pass transistor that serves to connect the stored value to a data bus under control of a select line. The select line simultaneously selects all transistors along the same row, causing data to be placed on the column lines corresponding to each selected cell. Although valid data appear along every column, only one of these columns is further selected for connection to the output on typical DRAMs. These one-transistor cells are formed into a memory architecture as shown in Fig. 9.8-1.

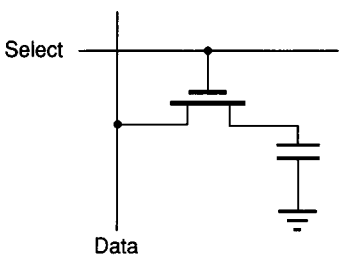


FIGURE 9.11-2
One-transistor dynamic RAM cell.

To keep the size of a dynamic memory cell small, both the select transistor and the storage capacitor must be small. The select transistor is a minimum or near-minimum size device. The small storage capacitor is required to charge the data line through the select transistor during a read operation. Because of the length of the data line, its capacitance is usually large compared with storage cell capacitance. When the select transistor connects the storage cell capacitance and the data line capacitance, the charge is shared to equalize the voltage across the two capacitors that now appear in parallel. Unfortunately, the charge on the larger data line capacitance will have more effect on the final data line voltage than the charge from the small memory cell capacitance. Thus, clever techniques and sensitive circuits are necessary to reliably sense the stored value of a DRAM cell.

One simple technique commonly used to sense the state of a dynamic memory cell involves splitting the data line into two equal halves, thereby splitting the capacitance. Both halves of the data line are precharged to a voltage approximately midway between the high and low logic levels. When a select line goes high, it connects a storage cell capacitor to one of the data line halves; the other half remains unselected. If a comparator circuit is connected with each data line half serving as an input, then even the small change in data line voltage caused by the selected capacitor cell can be detected. The inactive data line half serves as a reference point. This technique requires a comparator for each data line. A typical 256k $\times$ 1 DRAM has 512 data lines. The necessity of providing 512 comparators without using excessive area requires a simple comparator circuit.

Figure 9.11-3 shows a comparator circuit (also known as a sense amp) that has been used to sense the state of DRAM memory cells. This circuit is a flip-flop with special provision to break the cross-coupled links between the two inverters. Before a read operation, the column select, V_{FF}, and sense lines are set low. To execute a read operation, the data lines are precharged to equal voltages (V_{REF}); the desired data cell is gated by a row select to a column line, causing a slight voltage imbalance; the cross-coupled feedback lines of the flip-flop are connected (V_{FF}); and the flip-flop is enabled (Sense). The flip-flop was in a quasi-stable state before the sense line was asserted. The final state of the flip-flop is determined by the slight difference in voltage of the two data column halves caused by the selected memory cell. A later column select signal chooses one of the comparator outputs as the desired data.

Because of the regenerative action of the flip-flop, the data line half will be driven all the way to a high or low voltage, depending on the memory cell contents, and the selected memory cell on each column will be refreshed. That is, if the data cell voltage was higher than the precharged data line value, the flip-flop will switch to drive that half of the data line toward the supply voltage, thereby recharging the selected data cell. Conversely, a low data cell voltage will be discharged toward ground. All memory cells of a DRAM are refreshed by reading a cell on every row because all cells on a row are refreshed when any cell on that row is read. If a memory contains N storage cells and is organized as a square, the complete refresh operation requires a number of reads equal to the square root of N.

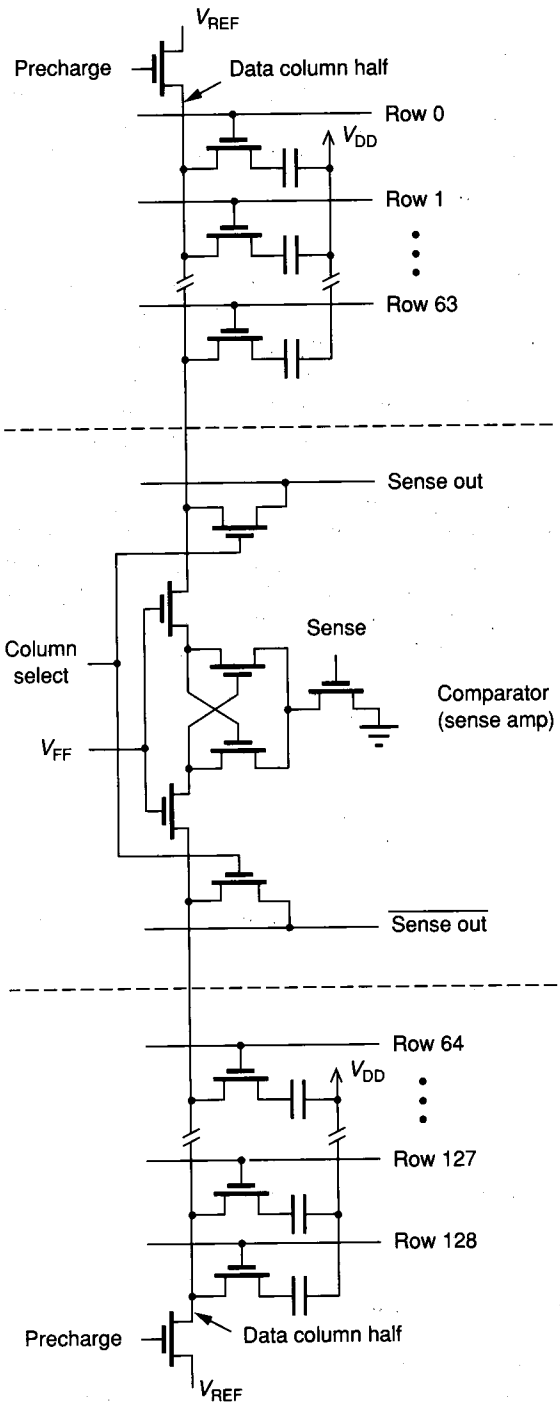


FIGURE 9.11-3
DRAM dynamic sense amplifier.

As DRAMs have gotten larger and storage cells smaller, the ratio between data-line capacitance and memory cell capacitance has increased because of longer data lines and smaller memory cells. To demonstrate how this ratio affects the sensing voltage, consider the following example.

Example 9.11-1. Sensing voltage versus cell capacitance Determine the voltage change on a DRAM data line caused by connection to a memory cell in terms of data line capacitance and memory cell capacitance.

Solution. Let the subscript c refer to the memory cell, and the subscript d refer to the data line. Before the memory cell is selected,

$$V_d = \frac{Q_d}{C_d}$$

and

$$V_c = \frac{Q_c}{C_c}$$

After the cell is selected, the charge is redistributed so that both capacitors are at the same voltage, V_f. Then

$$V_f = \frac{Q_f}{C_f} = \frac{Q_d + Q_c}{C_d + C_c}$$

The change in data-line voltage will be

$$V_d - V_f = (V_d - V_c)\frac{C_c}{C_d + C_c}$$

This analysis shows that the change in data-line voltage that must be sensed is the initial difference between the data-line and memory cell voltages diminished by the ratio of the memory cell capacitance to the total capacitance. Typical capacitance values are C_c = 40 fF and C_d = 1 pF. Thus, an initial 2.5 V difference is divided by 25, resulting in only a 100 mV change in the data-line voltage. It is difficult to detect such a small change, but modern DRAMs are able to do so reliably.

DRAMs have the highest sales volume of memory chips fabricated today. The simple storage cell described in this section leads to high density and low cost per bit. The requirement for refresh of DRAM cell contents is an important consideration in DRAM applications. The small cell/data-line capacitance ratio hinders rapid sensing of memory cell state. Further decrease in the cell/data-line capacitance ratio is an important factor in the design of next generation DRAMs.

9.12 REGISTER STORAGE CIRCUITS

Previous sections described the organization and characteristics of the major types of semiconductor memories. These descriptions were for memories that are usually implemented as stand-alone chips packed with as many memory cells as the current technology allows. Other applications for memory cells are found within sequential machines where the machine state must be stored. Sometimes this temporary storage is accomplished with the shift register described earlier.

Other times, the data must be stored for longer than one clock period. For example, a general-purpose register within a microprocessor typically must hold data while other operations are performed. The following sections describe two types of storage cells that are used within sequential digital systems.

9.12.1 Quasi-Static Register Cells

Figure 9.12-1 shows a way to combine two inverters, two pass transistors, and a nonoverlapping two-phase clock to provide a *quasi-static register cell*. Although the *quasi-static register cell* uses exactly the same components as a two-stage dynamic shift register (see Sec. 9.6), these components are interconnected in a different way. The output of a first inverter is connected directly to the input of a second inverter. One pass transistor, called the *input pass transistor*, controls the input to the first inverter. The second pass transistor, called the *feedback transistor*, controls a feedback path from the output of the second inverter to the input of the first inverter.

The operation of the sample circuit of Fig. 9.12-1 is as follows. When a binary value is to be stored in the register cell, the input pass transister is turned on, and the feedback transistor is turned off. This is accomplished through use of a LOAD signal ANDed with clock phase ϕ_1 to control the gate of the pass transistor. ϕ_2 is low so that the feedback path is broken at this time. When the input pass transistor is turned on, any signal applied to the D input of the register cell is passed to the gate of the first inverter, resulting in the same logic value at the output R of the second inverter (after two successive inversions). When the input pass transistor is turned off, the value at the input node of the first inverter is stored dynamically on the parasitic capacitance of that node. The value at the output of the second inverter is actively driven and is logically equivalent to the stored value at the input of the first inverter. During the ϕ_2 clock phase the output of the second inverter is fed back to the input of the first inverter, thus reinforcing its logic value. As long as this feedback condition is applied often enough, the quasi-static register cell will maintain its stored value.

If the register cell of Fig. 9.12-1 can maintain its stored value indefinitely, why is this circuit connection called a *quasi-static* register cell rather than a *static* register cell? The answer can be found by examining operation of the circuit

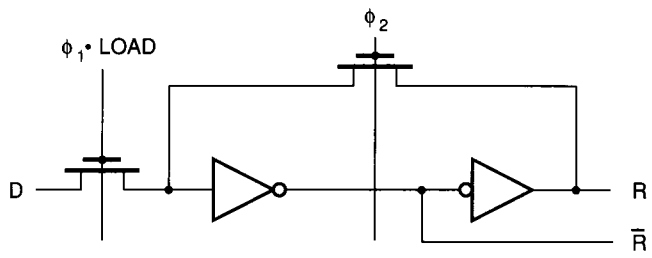


FIGURE 9.12-1
Quasi-static binary storage cell.

over a clock cycle in which a LOAD signal does not occur. Of course, while the ϕ_2 clock phase that controls the feedback transistor is high, the stored value is continuously reinforced. However, when the ϕ_1 clock phase that controls the input pass transistor is high but the LOAD signal is low, there is no active input to drive the gate of the first inverter. If this condition persists for too long a period, the logic value at this input gate may change because of charge leakage. Thus, there is a maximum time for the ϕ_2 clock connected to the feedback transistor to remain in the low state and still ensure the integrity of the data value stored in the cell. This condition places a lower bound on the clock frequency when quasi-static registers are used.

Quasi-static register cells were common in early microprocessors. For example, registers in the Motorola 6800 series of microprocessors were composed of an extension of the basic quasi-static cell that permitted dual-port read and write.[17] This cell, shown in Fig. 9.12-2, provides two gated load (write) signals on one clock phase, so the register can be loaded from either of two buses. A feedback path to refresh the stored logic value is provided on the alternate clock phase. The controller (not shown) that generates the write signals should logically AND them with ϕ_1 to avoid conflict with the feedback path that is controlled by ϕ_2 in Fig. 9.12-2. The register output, taken from the center of the register cell, drives a pulldown transistor. The output of this transistor is directed through pass transistors to one of two possible buses providing dual-port read. This cell requires four control signals (each externally gated by clock phase ϕ_1), an alternate clock signal ϕ_2 to control the feedback path, two bus lines (each bus line is common to one input and one output path), power, and ground. This cell, requiring a total of 10 transistors, will be compared with the static register cell described next.

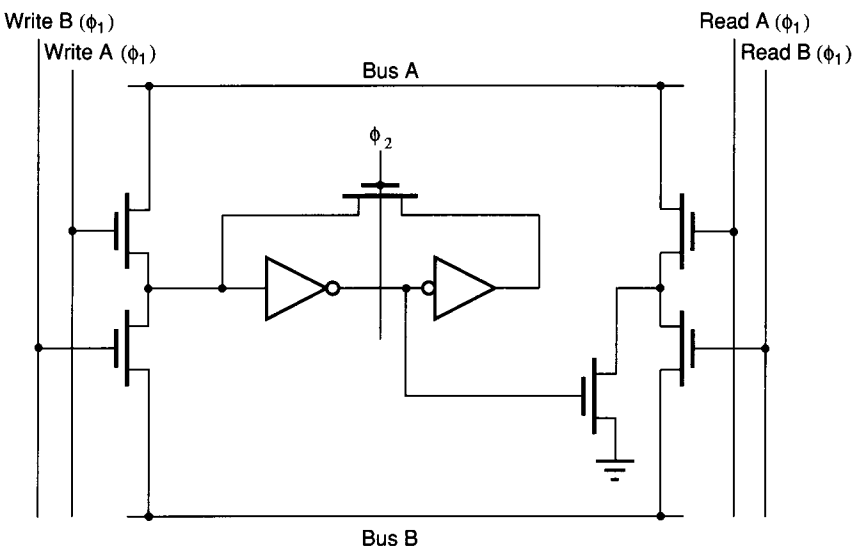


FIGURE 9.12-2
Motorola 6800 microprocessor register cell.

9.12.2 A Static Register Cell

Fully static register cells are frequently used within finite-state machines and within microprocessor register arrays. These static register cells are similar to the memory cell described previously for SRAMs, but often are designed with different constraints than those necessary for dedicated memory chips. One such static cell is based on the classical cross-coupled set-reset (SR) latch shown in Fig. 9.12-3*a*. This latch uses two cross-coupled NOR gates to achieve data storage. An equivalent NMOS transistor-level circuit for this latch is given in Fig. 9.12-3*b*. That this is a static register cell is obvious because the storage does not depend on clock signals, but only on a directly coupled feedback path.

To explain static register cell operation, the SR latch circuit of Fig. 9.12-3*b* will be transformed into a static register cell in two steps. Figure 9.12-4*a* shows the previous circuit split into a cross-coupled inverter pair with the set and reset pulldown transistors physically separated from the storage element by bus lines. These buses hold signals representing the register cell's logic state and its complement. Figure 9.12-4*b* completes the transformation by including pass transistors between the outputs of the cross-coupled inverter pair and the buses to the set and reset pulldown transistors. The pass transistors provide a way to isolate the register cell from the buses. Note that if both pass transistors are on, the circuit is equivalent to that of Fig. 9.12-3*b*, except for additional resistance in the set and reset pulldown paths because of the pass transistors. This basic static register cell consists of six transistors, four for the cross-coupled inverters and two for the connections to the buses. A CMOS version of this cell is created by replacing the NMOS inverters of Fig. 9.12-4 with CMOS inverters.

Because the basic register cell of Fig. 9.12-4*b* can be isolated from the buses, additional six-transistor register cells can be attached between the same

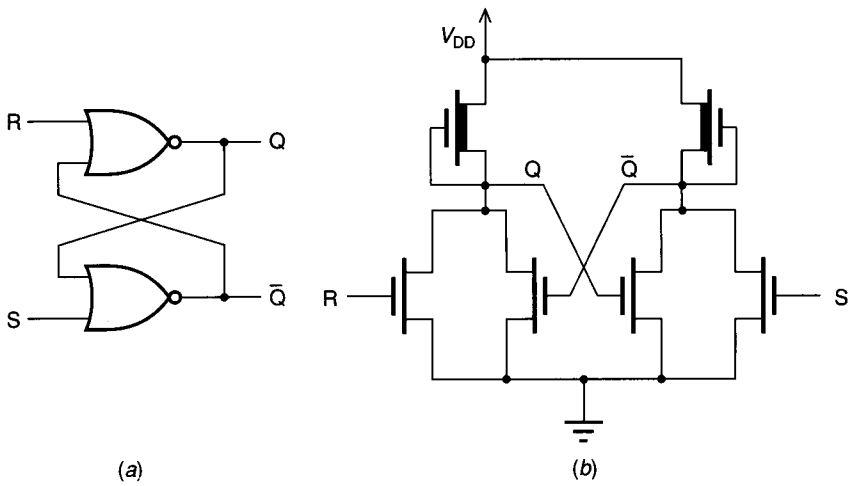


FIGURE 9.12-3
Cross-coupled NOR latch: *(a)* logic, *(b)* circuit.

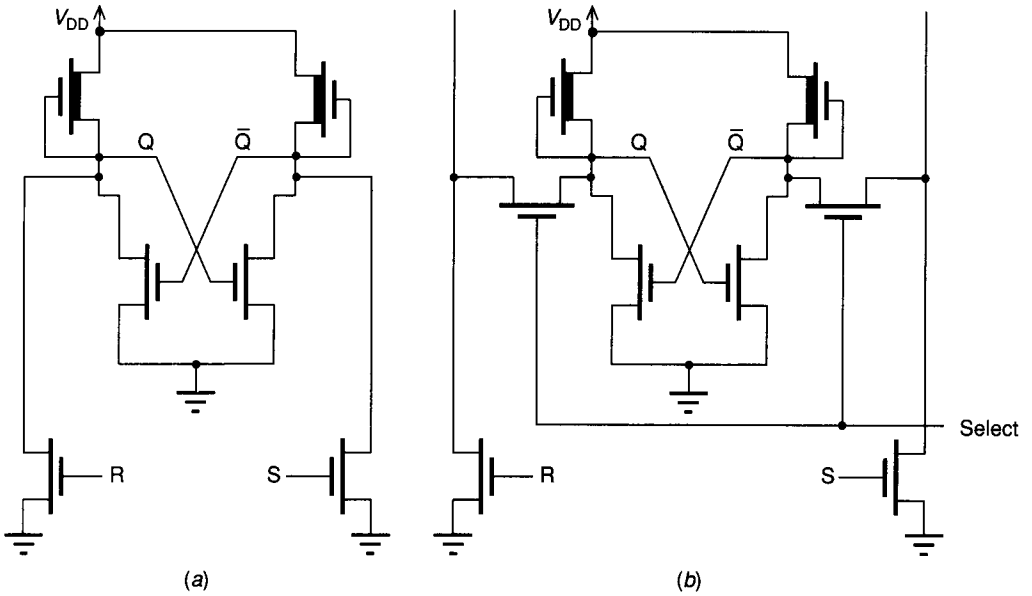


FIGURE 9.12-4
NMOS static storage cell.

two buses. Then a particular register cell is selected for read or write by selecting (turning on) both pass transistors associated with that cell. Figure 9.12-5 shows four CMOS static register cells that use the same two buses for read or write of cell data.

This static register cell is similar to those used in SRAMs. Many applications, however, do not require the considerable address decoding circuitry and sensitive read sense amplifiers necessary for large SRAM memory chips. There are two reasons for this. The first is size. A typical microprocessor application might require a register array with 1024 bits compared to commercial SRAMs with 256k bits. The smaller size reduces capacitive loading and diminishes noise sources, allowing simplified supporting circuitry. The second factor is organization. As explained in Sec. 9.8, a square organization requiring both row decoding and column selection to access a single bit is preferred for SRAMs. A typical 1024-bit microprocessor application might have 32 registers, each containing 32 bits. Each 32-bit register has its contents accessed as a unit. Thus, only a 5-to-32 address decoder is required to select a 32-bit register. Based on these concepts, then, a data value can be stored simply by selecting a cell and asserting a set or reset line. A stored value can be read by asserting the desired select line and accepting the logic value on the data bus. When this circuit is used within a microprocessor register array, a dual-port read is possible by controlling the two select transistors of a cell individually. Thus, one cell can have its stored value gated to the data bus, while a second cell has its stored value gated to the complement data bus. Many microprocessor instructions require two input operands, making the dual-port structure highly desirable for a register array.

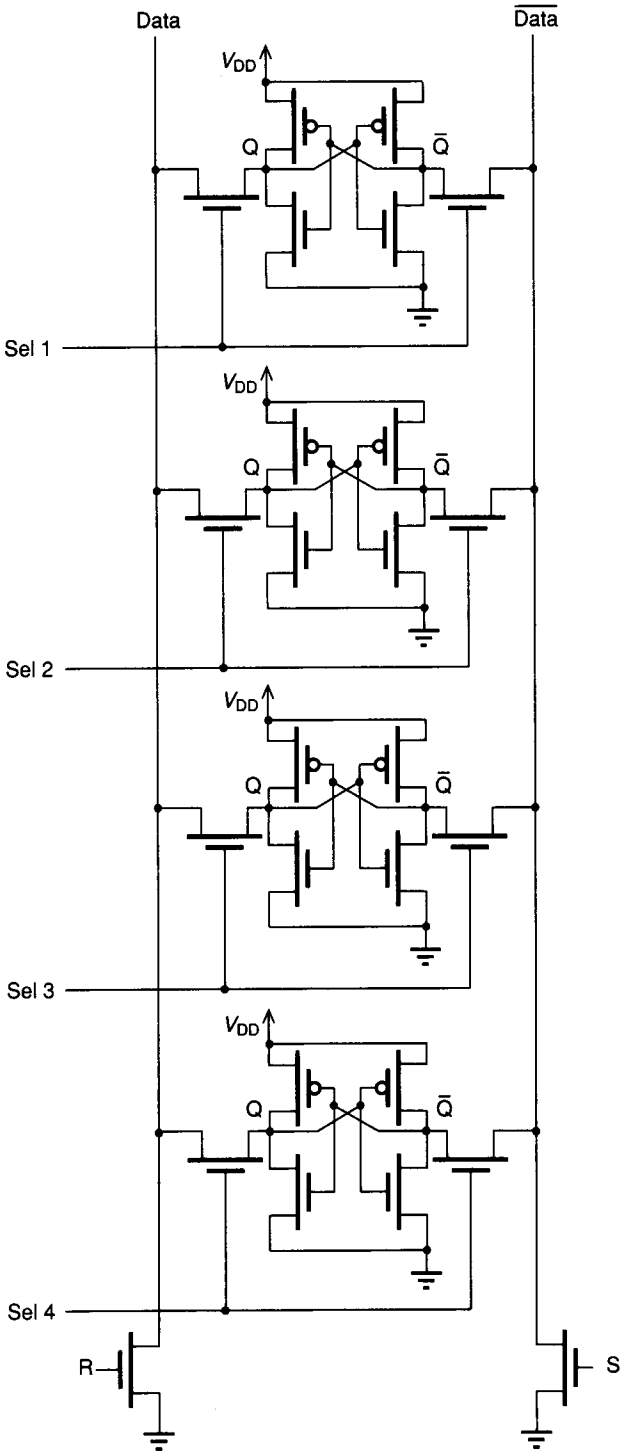


FIGURE 9.12-5
CMOS static register cell array.

Two simple storage cells were described in this section. These are important in digital design for applications that require static storage capability, for example FSMs and microprocessors. Both quasi-static and fully static storage cells were described. Individual storage cells are easily configured into n-bit wide registers where n is set by the width of the data word. The simple design and operation of these circuits make them ideal for many applications.

9.13 PLA-BASED FINITE-STATE MACHINES

Most digital systems are composed of combinational logic and memory combined in a form called a *finite-state machine (FSM)* or, equivalently, a *sequential machine*. A sequential machine is normally implemented as a forward path containing combinational logic and a feedback path that includes memory. In classical digital systems the memory is provided by flip-flops or latches. Within MOS integrated circuits a particularly simple form of sequential machine is possible. This simple finite-state machine consists of a PLA that realizes the combinational logic and a clocked shift register in the feedback path to serve as memory. A dynamic shift register such as the one described in Sec. 9.6 is often used.

Figure 9.13-1 shows the classical form for one type of FSM, called the *Moore machine*.[18] This FSM is characterized by outputs that are isolated from momentary input changes by memory. This type of FSM is of particular interest here because an excellent integrated circuit implementation based on a PLA is available. A block diagram of a PLA-based FSM is shown in Fig. 9.13-2, where a PLA is augmented with pass transistors to gate its inputs and outputs. These pass transistors in combination with the output buffers and double-rail drivers form a clocked shift register so that the next state presented by the PLA OR plane is available as the present state at the inputs to the PLA double-rail drivers after a ϕ_1, ϕ_2 clock sequence.

As discussed in Sec. 9.2, automatic PLA generation programs are available. Based on logic equations in the sum-of-products form, a complete PLA layout can be created and programmed to realize correctly the specified logic functions. It is a simple task to augment a PLA generation program to include clocked input drivers and clocked output buffers in the form shown in Fig. 9.13-2. Such a PLA generator can be used in one of two modes: it can generate a standard

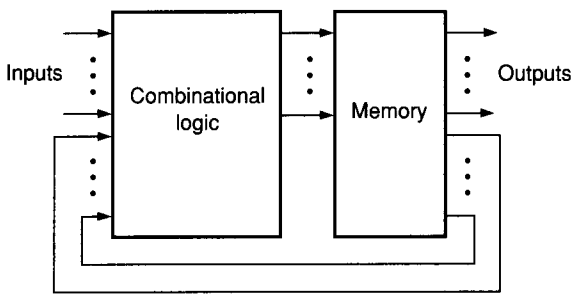


FIGURE 9.13-1
Classical finite-state machine.

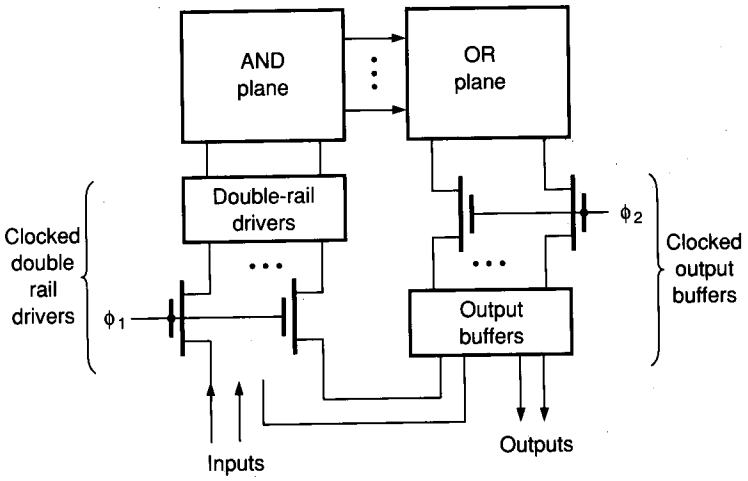


FIGURE 9.13-2
FSM based on PLA with clocked stages.

PLA consisting of combinational logic only, or it can generate a FSM formed from a standard PLA plus clocked shift register feedback created by connecting the output of a clocked output buffer to the input of a clocked double-rail driver.

To demonstrate the design of a FSM based on a PLA with clocked shift register feedback, consider the following example.

Example 9.13-1. Finite-state machine Assume that 16 magnetic switches are to be monitored remotely for a home security application. The 16 status bits corresponding to the state of the switches are available from the remote location through an asynchronous serial data link as alternating bytes of data. The serial data is received in 8-bit groups by a UART (universal asynchronous receiver/transmitter) whose parallel output must be stored in two eight-bit registers that drive LED indicators. Each register is composed of 8 D-type flip-flops activated by a rising clock signal. The first byte of data received is displayed in one set of LED indicators, and the alternate byte is displayed in a second set of LED indicators. Thus, there are 16 LED indicators, one for each magnetic switch. This task can be accomplished with a sequencer (FSM) that alternately selects one of two display registers, A or B, to store the received data bytes. To simplify the design, assume that the system is always synchronized with the first, third, and other odd bytes going to display register A and the even bytes to display register B. The FSM must also generate a data strobe signal (S) required by the UART to acknowledge that a byte is accepted. Of course, the UART generates a data ready signal (R) whenever a new status byte is available at its output. The logic components that compose the receiving system are shown in Fig. 9.13-3. Show the logical design for the PLA FSM block of this figure.

Solution. A PLA FSM that satisfies these requirements is described by the state diagram of Fig. 9.13-4 and the state transition table of Table 9.13-1. The FSM waits in state *a* until the UART indicates that a data byte is ready by asserting the

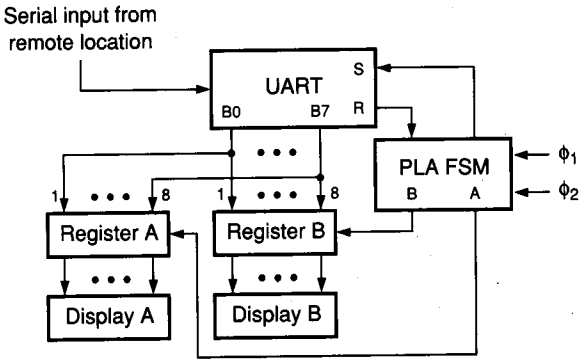


FIGURE 9.13-3
Block diagram for system display.

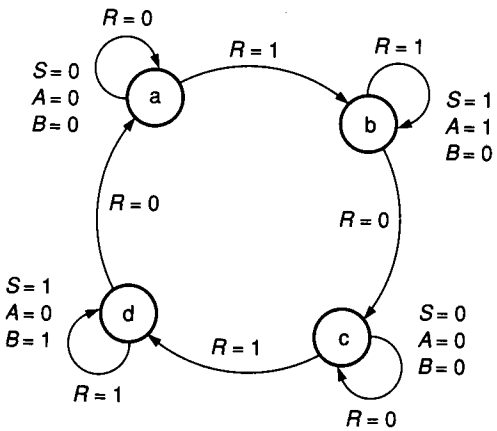


FIGURE 9.13-4
State diagram for status sequencer.

TABLE 9.13-1
State transition table for security monitoring system

Input	Present state	Next state	Outputs		
R	*XY*	*xy*	*S*	*B*	*A*
0	00	00	0	0	0
1	00	01	0	0	0
0	01	11	1	0	1
1	01	01	1	0	1
0	10	00	1	1	0
1	10	10	1	1	0
0	11	11	0	0	0
1	11	10	0	0	0

data ready signal (R). Data ready causes a transfer to state b at the next clock, causing a load signal (A) to display register A and a data strobe signal (S) to the UART. It is important that the display register is loaded by the rising edge of load signal A. Then the FSM waits in state b until the UART removes the data ready signal (R), causing a transfer to state c. At this point, one byte of data has been received and the value in register A updated. When a second byte from the UART is ready, the resulting data ready signal (R) causes a transfer to state d at the next clock, where the appropriate load signal (B) to display register B and data strobe signal (S) are generated. Later, after data ready is removed by the UART, the FSM returns to the first state and waits for new status data.

The state transition table (Table 9.13-1) provides the information necessary to specify the logic operations to be performed by the PLA. Two state variables are required to specify four different states. Call the present state variables X and Y and the corresponding next state variables x and y. The states are encoded with a Gray code (state $a = 00$, state $b = 01$, state $c = 11$, state $d = 10$) so that only one state variable changes for each state transition. The equations for the next state variables and the outputs are obtained from the state transition table and are given here.

$$x = \overline{R}Y + RX$$

$$y = \overline{R}Y + R\overline{X}$$

$$S = \overline{X}Y + X\overline{Y}$$

$$A = \overline{X}Y$$

$$B = X\overline{Y}$$

From these equations, it is easily determined that a (3,5,5) PLA is required—that is, three inputs by five product terms by five outputs. Two outputs form the next state variables, while three other outputs generate the data strobe (S), load register A (A), and load register B (B) signals. Figure 9.13-5 shows a complete PLA-based FSM that implements the controller described here.

The ability to create a FSM automatically from a set of Boolean logic equations is an extremely powerful tool for digital system design. Small PLA-based FSMs are frequently used as building blocks to construct larger digital systems such as microprocessors and communications processors. Large PLA-based FSMs suffer from two important limitations. A large PLA may be sparsely populated with programming sites, resulting in excessive area to realize a function. Also, large PLAs tend to be slower than alternative solutions when a large number of terms must be processed. One alternative is to use several small PLA FSMs rather than one large PLA FSM to implement required control logic; a second alternative is described in the next section.

9.14 MICROCODED CONTROLLERS

The clocked PLA structure for FSMs explained in the previous section is an excellent means to implement small digital controllers. The layout structure is regular and can be generated automatically and compactly from the logic

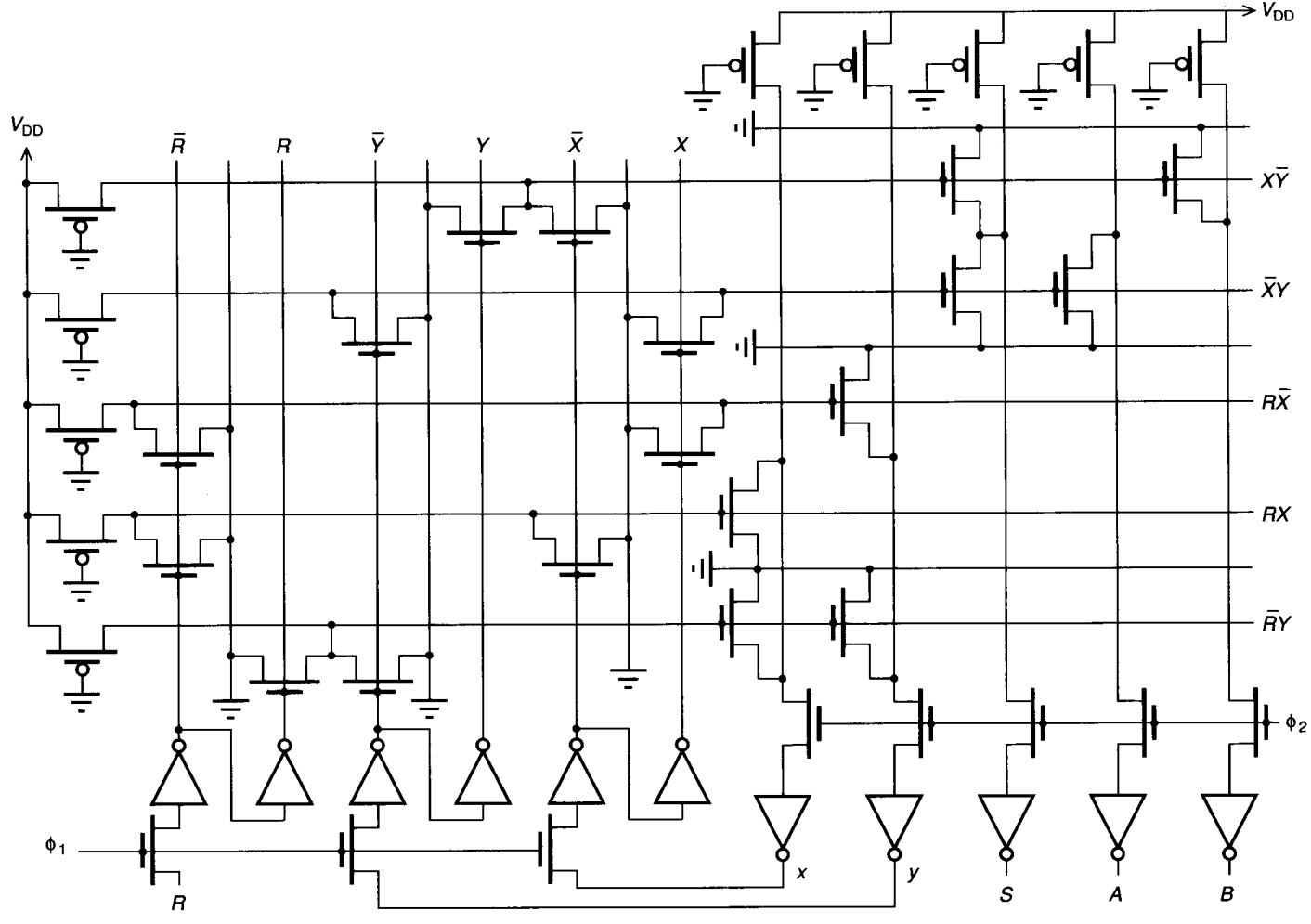


FIGURE 9.13-5
Clocked PLA FSM for sequencer.

equations for a system. For larger digital systems, the logic design to implement a clocked PLA FSM becomes unnecessarily complex and the resulting large PLA, if generated, would be slow. These larger systems require a method that overcomes the disadvantages of a large clocked PLA FSM yet emphasizes regularity in design and layout. A common method to implement complex digital systems in a regular way is to use a memory-based structure known as a *microcoded controller.*

A microcoded controller (shown in Fig. 9.14-1) comprises a memory whose contents are called *microinstructions* and a *next-address sequencer* that directs the execution sequence of microinstructions. A microinstruction is a set of (usually) encoded control bits that direct the operation of the logic during a clock cycle. In essence, a microcoded controller is a special form of computer. The execution hardware is fixed, and the functions performed are a result of instructions placed in the microinstruction memory. This memory is frequently read-only memory and is thus called *microROM.* As discussed earlier, memories are designed with a dense, regular structure. Because the microcoded controller consists primarily of memory, a microcoded controller can also be regular and dense. However, because microcoded controllers require the overhead of a next-address sequencer that requires design time and integrated circuit area, this technique is used primarily for larger machines.

In its simplest form, the microcoded controller of Fig. 9.14-1 does not require status inputs. The next-address sequencer simply generates the next instruction addresses in a fixed pattern, for example, by incrementing a counter. A microcoded controller configured in this way functions as an *open-loop controller* with a fixed execution sequence. If status inputs are provided, the next-address sequencer can modify the address of the next instruction, depending on conditions presented by the status inputs. This provides a *conditional branching capability.* In either case, the function of the microcoded controller is determined primarily by a program placed in its microROM. Conceptually, programming a microcoded controller is similar to programming a microprocessor in machine language. In practice, however, the programming task is extremely tedious because of the multiplicity of individual control bits whose state must be determined for each instruction.

Figure 9.14-2 shows a typical memory organization for a microcoded controller consisting of a microROM and a memory address register (MAR). While most semiconductor memory chips are organized with a wide address bus and a

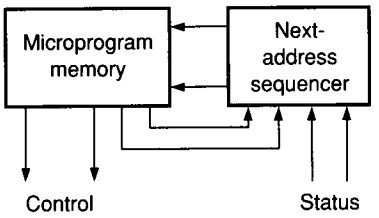


FIGURE 9.14-1
Simple microcoded controller.

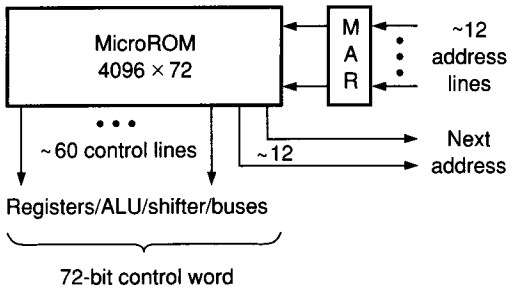


FIGURE 9.14-2
MicroROM architecture.

narrow data bus (nine multiplexed address lines and one data line for most 256k DRAMs), the memory (microROM) for a microcoded controller usually has a wide data bus relative to its address bus (perhaps 72 or more data lines compared with 12 or fewer address lines). Most of these data lines are dedicated to driving control points within the system. A few data lines are used to provide next-address information to the next-address sequencer. The next-address sequencer uses this address information along with status inputs from the controlled process to calculate the address of the next microinstruction.

A microROM organized as in Fig. 9.14-2 would contain almost 300k bits ($2^{12} \times 72 = 294,912$) of control information and would consume a correspondingly large silicon area. An alternative form for the microROM is shown in Fig. 9.14-3. This two-level microprogram memory consists of a relatively small microROM driving a secondary memory called a *nanoROM*. This organization is based on two reasonable assumptions. First, only a few of the 2^{72} possible control word combinations of Fig. 9.14-2 are necessary in a given system. Second, many of the control words that are necessary will be required repeatedly. If fewer than 256 unique control words are necessary, for example, and the microprogram memory is organized as shown in Fig. 9.14-3, only about 50k bits ($2^{12} \times 8 + 2^8 \times 72 = 51,200$) of control memory are required. This reduction in memory size is not free; the two-level microROM is slower than a single-level memory because a memory access must traverse two memory units to produce data. Overlapped instruction fetch and execution and careful design of control circuitry to minimize additional delay can partially offset the slower control memory.

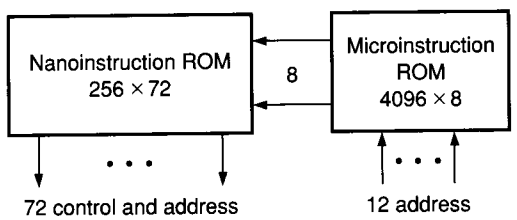


FIGURE 9.14-3
Two-level microprogram memory.

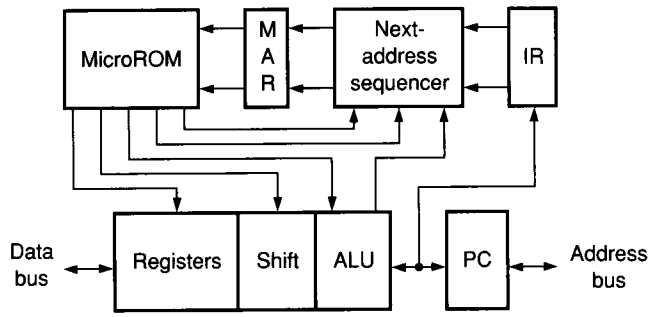


FIGURE 9.14-4
Microprogram-controlled microprocessor architecture.

A simplified block diagram for a microprogram-controlled microprocessor is given in Fig. 9.14-4. The microprogrammed controller drives a data path with registers, a shifter, and an arithmetic logic unit (ALU). A memory address register (MAR), an instruction register (IR), and a program counter (PC) are also shown. The operation of this design will be examined with the following example.

Example 9.14-1. Simple microprogrammed instruction execution Explain how the microprogrammed controller of Fig. 9.14-4 could be used to add the contents of two registers (register 4 and register 7) and return the sum to register 4. In register transfer form, the required operation is

$$\text{Register 4} \leftarrow \text{Register 4} + \text{Register 7}$$

Solution. To start execution, a program memory address is placed in the PC. The PC contents are placed on the address bus and a computer instruction is fetched from program memory (not shown) over the data bus and placed in the IR. The next-address sequencer uses the instruction in the IR to specify a particular starting address in the microROM. The corresponding microprogram control word is output from the microROM. This control word selects register 4 and subsequently gates register 4's contents to the ALU. If the register file is organized for dual-port read, the same control word from the microROM simultaneously selects register 7 and gates its contents to the ALU along a second bus. A next microROM control word address is generated by the next-address sequencer. This address selects another microROM control word, which causes the ALU to add its two input operands. Still another address is provided by the next-address sequencer, and a third microROM control word is selected. This last control word stores the result of the ALU operation in register 4 and prepares the microprocessor to fetch the next instruction for the IR from program memory by updating the PC contents.

This rather simplistic description of an ADD instruction demonstrates basic operation of a microprogram-controlled data path. In many instruction sequences, the next microROM address depends on results of an ALU operation. This allows conditional branching to be implemented. The preceding description omits many important considerations, including timing, pipelined operation, program counter update, and control signal generation.

It is appropriate at this point to compare the PLA and microprogrammed forms of FSMs. In general, a microprogrammed control unit is more complex than the corresponding PLA FSM because of the next-address generation circuitry. In fact, a PLA FSM can be compiled automatically once the state equations for the system are determined; this is much more difficult for a microprogrammed FSM. The peripheral circuitry for a microprogrammed FSM usually depends on the application and is thus not automatically generated. For these reasons, the PLA FSM is normally best for small, simple systems where minimum design time and circuit area are required. However, larger systems are sometimes created through use of several small PLA FSMs to offset the difficulties with large PLAs. The microprogram machine is usually more desirable for larger systems, where the additional design and area penalties can be offset by its advantages. A significant advantage is the ability to substantially change the details of operation by modifying the contents of the microROM prior to manufacture without changing the underlying circuit design and layout. This may be necessary for correction of design errors or to create new capabilities for a working design.

General agreement on the form of FSM that is most appropriate for commercial microprocessor design does not exist. Recent 32-bit microprocessors have been designed using each of these FSM forms. For example, the Bellmac 32 uses several small PLAs for its control circuitry, and the HP 9000 uses a large microprogrammed memory to control its operation.[19,20]

9.15 MICROPROCESSOR DESIGN

The focus of this chapter is structured forms of digital integrated circuits. The evolution of microprocessors provides an interesting study in the development of structured logic forms. The earliest microprocessors, the Intel 4004 and 8008, were born to counteract the high development costs for custom large-scale integrated (LSI) circuits.[21] Because custom large-scale circuits had to be designed for specific tasks, it was often difficult to reach the sales volume required to justify the development of a custom part. This literally forced the development of a logic form (the microprocessor) that could be tailored to many different applications by the addition of control logic (programs) contained in separate integrated circuit devices (memory chips). Only through the large application market that could be served could the development costs for a custom LSI circuit like the microprocessor be recovered.

As the complexity of microprocessors has increased, the design time and costs have also expanded. Development of structured designs using regular logic forms, such as those discussed in this chapter along with new computer-aided design tools, has been required to allow the evolution of microprocessor architecture. A comparison of the Intel 4004 die photograph (Fig. 9.15-1) with the Intel 80386 die photograph (Fig. 9.15-2) provides a vivid illustration of the relative percentages of silicon area used for regular structures and the relative complexity of these two microprocessors.

Today's basic microprocessor consists of a *control unit* and a *data path*. This is shown by Fig. 9.14-4 of the previous section, where the data path consists

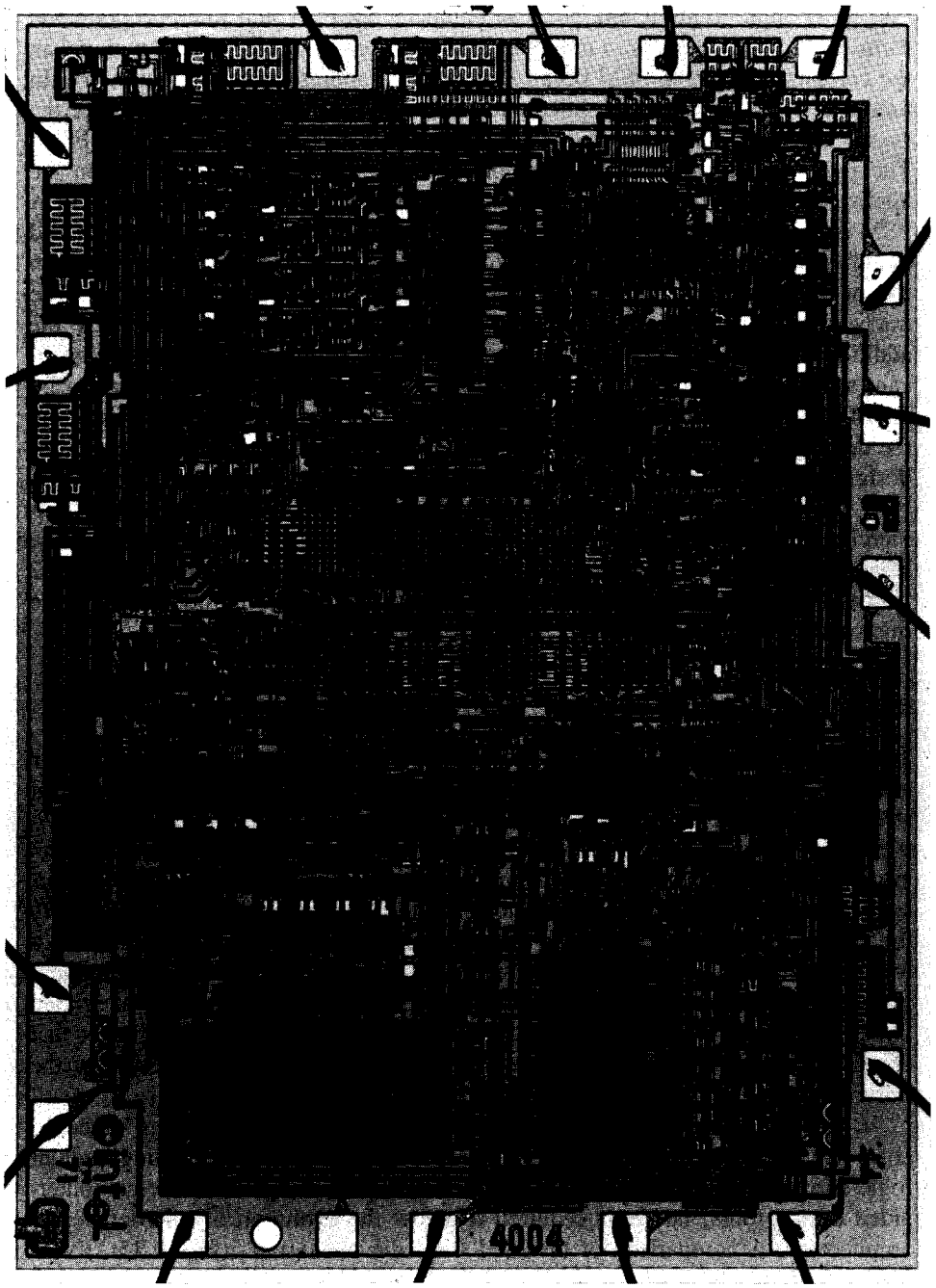


FIGURE 9.15-1
Intel 4004 die photograph (Courtesy Intel Corp.).

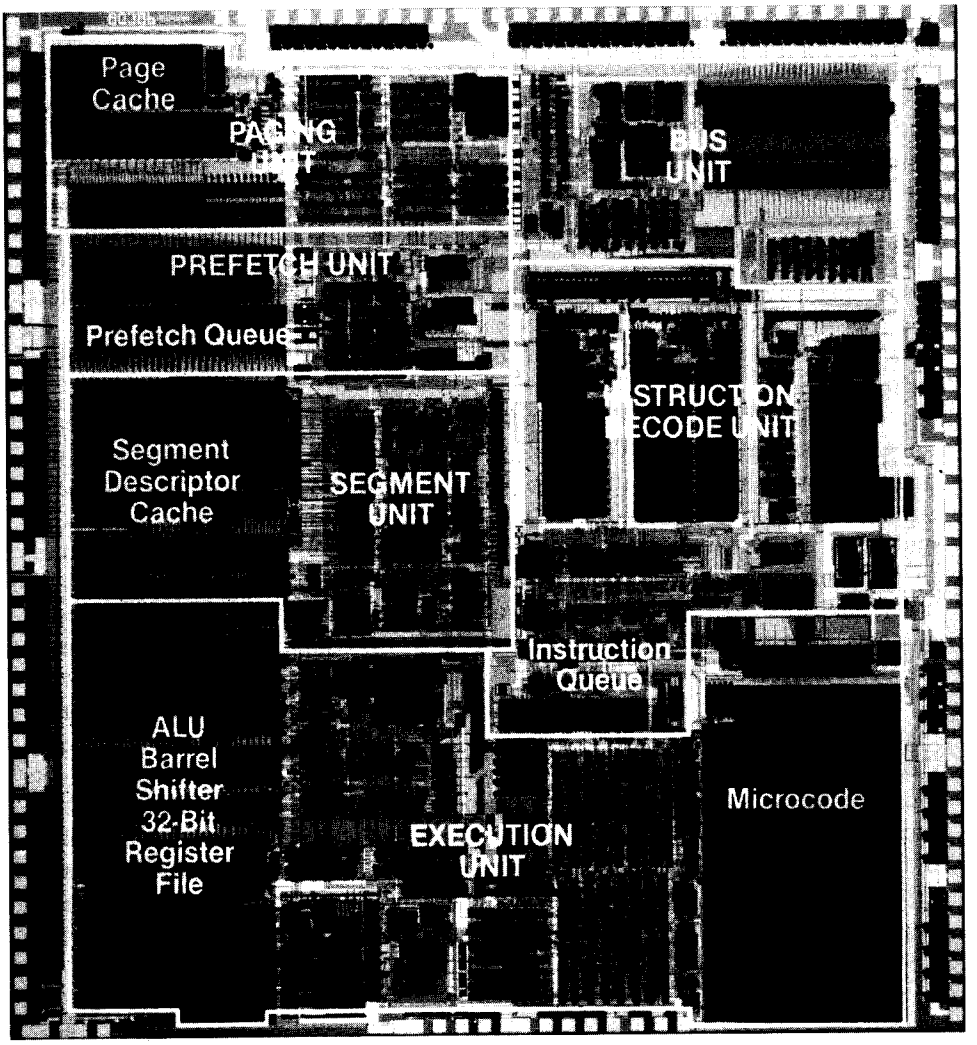


FIGURE 9.15-2
Intel 80386 die photograph (Courtesy Intel Corp.).

of registers, shifter, and ALU, and the control unit contains the microROM, MAR, next-address sequencer, IR, and PC. Although these two subsystems (control unit and data path) may be augmented with bus interface controllers, memory management units, cache memory, and other functions by different manufacturers, the present discussion will focus on the control unit and the data path as essential components of a microprocessor. The data path for a microprocessor is usually formed with 8, 16, or 32 identical bit paths. As a result of these identical bit paths, there is an inherent regularity within the data path for microprocessors. In contrast, the control units have varied structures, with most present manufacturers choosing microcoded or PLA style controllers.

9.15.1 Data Path Description

The data path, sometimes called the *execution unit*, is the place where the microprocessor executes operations such as addition, subtraction, shifts, rotates, and Boolean logical functions on data. Figure 9.15-3 shows a typical n-bit data path structure consisting of a dual-port register array, a barrel shifter, an ALU, interconnection buses, and support circuitry. Data flows along n parallel paths in the horizontal direction, while control of the data flow and ALU operations is provided vertically from the top of the data path. Execution of a typical data path operation (see Example 9.14-1) requires selection of operands from two registers, execution of an operation on the two selected operands, and placement of the result in a register. The elements of the data path must be designed to facilitate such operations.

The use of a dual-port register array is convenient for the fast execution of microprocessor programs. This local storage is usually provided within the data path as a small array of static memory cells. These are organized as an $n \times m$ structure where n is the width in bits of the microprocessor data bus and m is the number of registers provided. Because an ALU operation often requires access to the contents of two registers before execution can commence, most register arrays are organized with dual-port read access. With a dual-port register array, contents from two separate registers can be fetched simultaneously to minimize the delay before execution of an operation can begin.

The memory cell structure of a dual-port register array is quite similar to that of an SRAM cell. There is a need, however, for two data buses and a mechanism to allow the contents of each register to be switched to either data bus. The memory cell used for the dual-port register array of the Berkeley RISC processor[22] is shown in Fig. 9.15-4. In this circuit, the designers took advantage of the provision of double-rail data access to allow the contents from two registers to be obtained simultaneously. Remember that double-rail access is normally required to allow storage of data in a simple cross-coupled inverter storage cell.

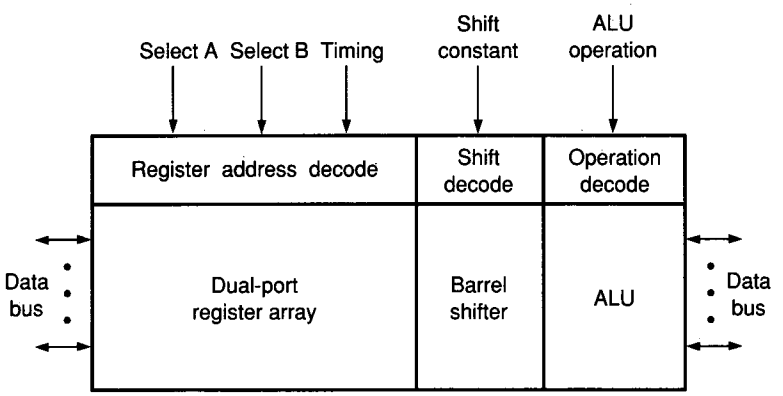


FIGURE 9.15-3
Microprocessor data path.

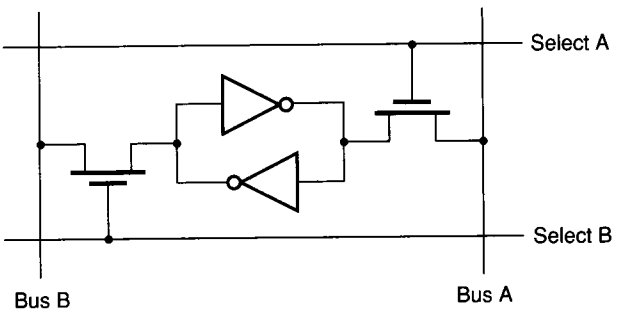


FIGURE 9.15-4
Dual-port register cell.

For a write operation, both data lines must be gated to the storage cell with complementary values. However, the contents of the storage cell may be read by gating the cell to either data line. If provision is made to drive the two select lines, A and B, separately for a read operation, then it is possible to obtain the data from a first register along one rail of the data bus (bus A), while the data from a second register is obtained along the other rail of the data bus (bus B). Of course, the data from the second bus will be the complement of the cell data and must be inverted.

9.15.2 Barrel Shifter

A second component that is included in the data path for many microprocessors is a structure that allows the contents of the data path to be shifted or rotated. A variable-length shift of a bit on the data path requires the possibility of connecting the selected bit to any one of several other bit paths. A 1-to-n multiplexer circuit for each bit will accomplish the desired connection. An ideal means of implementing multiplexer circuits is provided by the pass transistor available within MOS integrated circuits.

A particularly useful circuit structure to implement a shift or rotate is known as a *barrel shifter*. This circuit structure can be explained by first considering Fig. 9.15-5, which shows the circuit diagram of a general-purpose bus multiplexer for a 4-bit data path. This multiplexer circuit requires 16 pass transistors to allow connection of any bit line to any other bit line. If each pass transistor could be selected individually, 16 control lines would be required. Because most requirements are for parallel shifts with all bits moved the same number of bit positions, only four shift possibilities are really necessary. Figure 9.15-6 shows a better circuit with the pass transistors connected in groups of four, reducing control line requirements from 16 to 4 separate control lines, 50-53. A particular control line might be selected by encoding a 2-bit control field to drive a 2-to-4 decoder circuit. The individual decoder output would enable the proper shift control line. For a 32-bit data path, 1024 pass transistors are necessary to allow the desired shift operations. Assuming only parallel shifts, 32 control lines selected by a 5-bit encoded control field are sufficient.

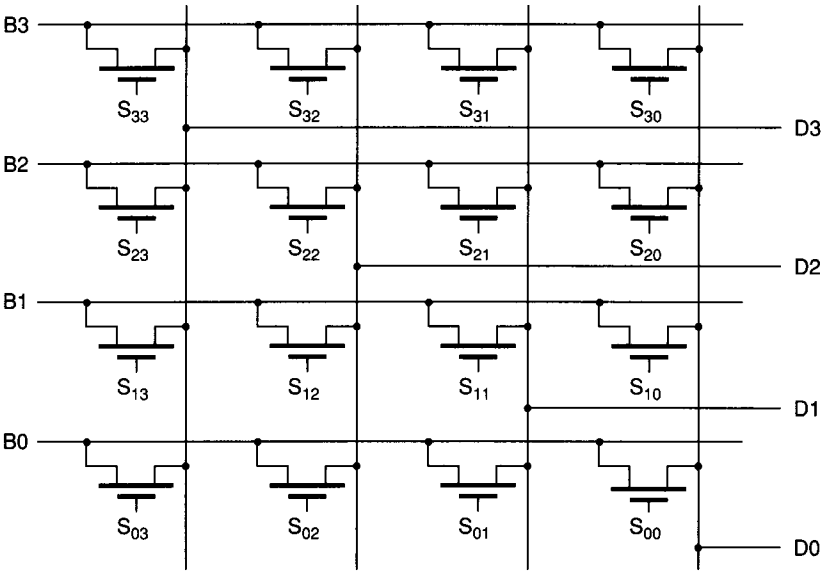


FIGURE 9.15-5
General bus multiplexer (16 control lines, S_{00} . . . S_{33}).

Studying the barrel shifter structure of Fig. 9.15-6 allows some interesting observations. First, note that if the data path runs horizontally, it is necessary to provide a vertical path for control to implement the shift function. This vertical connection is sometimes used to insert or extract external data to or from the data path along D4–D6 or D0–D3. Remembering that control signals S0–S3 for the data path are provided vertically, the vertical path of the barrel shifter can be used to insert data that is part of the data path control instruction. Data that is part of the data path instruction is usually called *literal* or *immediate* data. Second, recognition of the simplicity of the barrel shifter structure suggests that the vertical pitch of the data path layout will probably be limited by the vertical pitch of the register array or the ALU rather than by the barrel shifter. This observation allows minimization of the horizontal dimension of a barrel shifter while the vertical dimension is stretched to match the rest of the data path to obtain area efficiency of the layout.

9.15.3 Arithmetic Logic Unit

The arithmetic logic unit (ALU) is the last important part of the data path to be discussed. As its name suggests, the ALU must provide arithmetic and logic operations on data furnished from the data path. The ALU accepts two operands, performs a specified operation, and outputs the result. A block diagram of a 32-bit wide ALU showing the inputs, control, and outputs is given in Fig. 9.15-7. The *A* and *B* inputs of the ALU along with the dual-port register array discussed earlier suggest that two parallel 32-bit buses should be provided in the data path

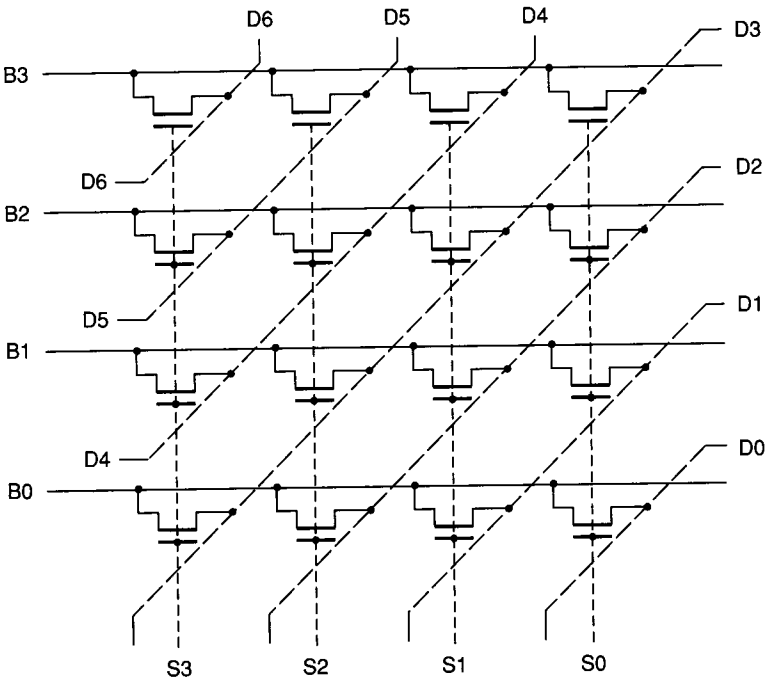


FIGURE 9.15-6
Barrel shifter (4 control lines, S0-S3).

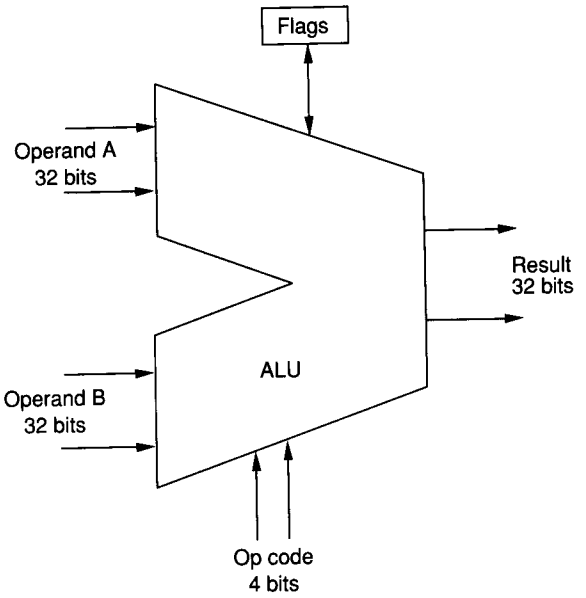


FIGURE 9.15-7
Arithmetic logic unit.

TABLE 9.15-1
Common ALU operations

Mnemonic	Operation
ADD	Add
ADC	Add with carry
SUB	Subtract
SUBC	Subtract with borrow
NEG	Negate (2's complement)
AND	Logical AND
OR	Logical OR
EOR	Logical exclusive-OR
COM	Complement (1's complement)
CMP	Compare
ASL	Arithmetic shift left

between the register array and the ALU. This allows both inputs to receive data simultaneously. One of the two parallel buses is used to return the result to the register array after the ALU operation is complete.

The ALU for a microprocessor is normally expected to accomplish the operations of Table 9.15-1 as a minimum. These 11 functions require a 4-bit encoded operation code (often abbreviated as *op code*) for their selection, although a 3-bit op code could be used if certain of the operations were combined. For example, the operations ADD and ADC could be implemented by specifying the ALU ADC operation with a separate control line to choose 0 or the previous carry as the carry input to implement ADD or ADC, respectively. The function COM could be implemented by using the EOR operation with the second ALU operand set to all 1s. And the ASL function could be implemented by providing the same operand to both ALU inputs and executing the ADD operation. Although a 3-bit ALU operation code would suffice, other control lines are required to select the carry input or set the second ALU operand to all 1s.

The ALU execution time may limit the maximum clock frequency of the microprocessor unless special care is taken for arithmetic operations. These operations are slowed by carry or borrow propagation delays across the width of the ALU. This problem has worsened as data bus widths have moved from 8 through 16 to 32 bits. Most microprocessors use a precharged carry line with each bit position of the ALU required to generate a carry propagate or a carry generate signal. In addition, newer microprocessors include one or more levels of carry skip circuits to speed carry propagation across groups of adjacent stages.

9.15.4 Microcoded Controller

Most present microprocessors use a form of microcoded controller (described earlier in this chapter) to generate required control signals for operation. Both the Motorola 68030 shown in Fig. 9.1-3 and the Intel 80386 shown in Fig. 9.15-2 are examples of microcoded microprocessors. Many times, bit fields of the microprocessor instruction word can be used directly to simplify the control field of

the microROM output. For example, encoded register specification fields can be gated from the instruction register to the data path to save microcode bits. Also, the operation code field of the instruction can directly specify the address in the microROM where execution of an instruction should start. Specific details of controller implementation vary considerably with different microprocessors and different manufacturers.

9.16 SYSTOLIC ARRAYS

The ability to place hundreds of thousands of transistors on a single integrated circuit and then replicate that circuit inexpensively has led many researchers to propose the use of connected sets of identical integrated circuits to solve problems in parallel. In particular, one class of parallel processors has been given the name *systolic arrays* because the data flow through the array is analogous to the rhythmic flow of blood through human arteries after each heartbeat.[23] In essence, the concept of systolic processing combines a highly parallel array of identical processors with local interconnections and rhythmic data flow. The array of processors may span several integrated circuit chips. The connections are formed so that data is accepted and processed at each stage, with the result ready for output to the next stage as new data arrives at the current stage. The objective is to keep most processors busy doing useful work to reduce the time to achieve a result. Three examples of parallel computational systems using multiple, identical circuits are described in this section.

9.16.1 Systolic Matrix Multiplication

Figure 9.16-1 shows a systolic interconnection of processors arranged to compute the matrix multiplication product

$$C = A \times B$$

Consider any single hexagonal processing stage of this array. An element of the A matrix arrives from the upper left; an element of the B matrix arrives from the upper right; and a partially computed element of the C matrix arrives from the bottom. The calculation $c_{ij} = c_{ij} + a_{ik}b_{kj}$ is performed, and the new value for c_{ij} is passed up to the next processing stage. Two successive steps of this calculation are shown in Fig. 9.16-1*a* and *b*. With this organization, all data are moved over local interconnections, and one-third of the processors are busy at each step. Eventually, the newly computed C matrix will surface at the top of the array. The total computation can be performed faster with a systolic array such as this than with a sequential computer because many of the required calculations are performed in parallel.

9.16.2 General Linear System Solver

A second look at the systolic array of Fig. 9.16-1 reveals that local feedback paths are not present. Although this is not a concern for matrix multiplication,

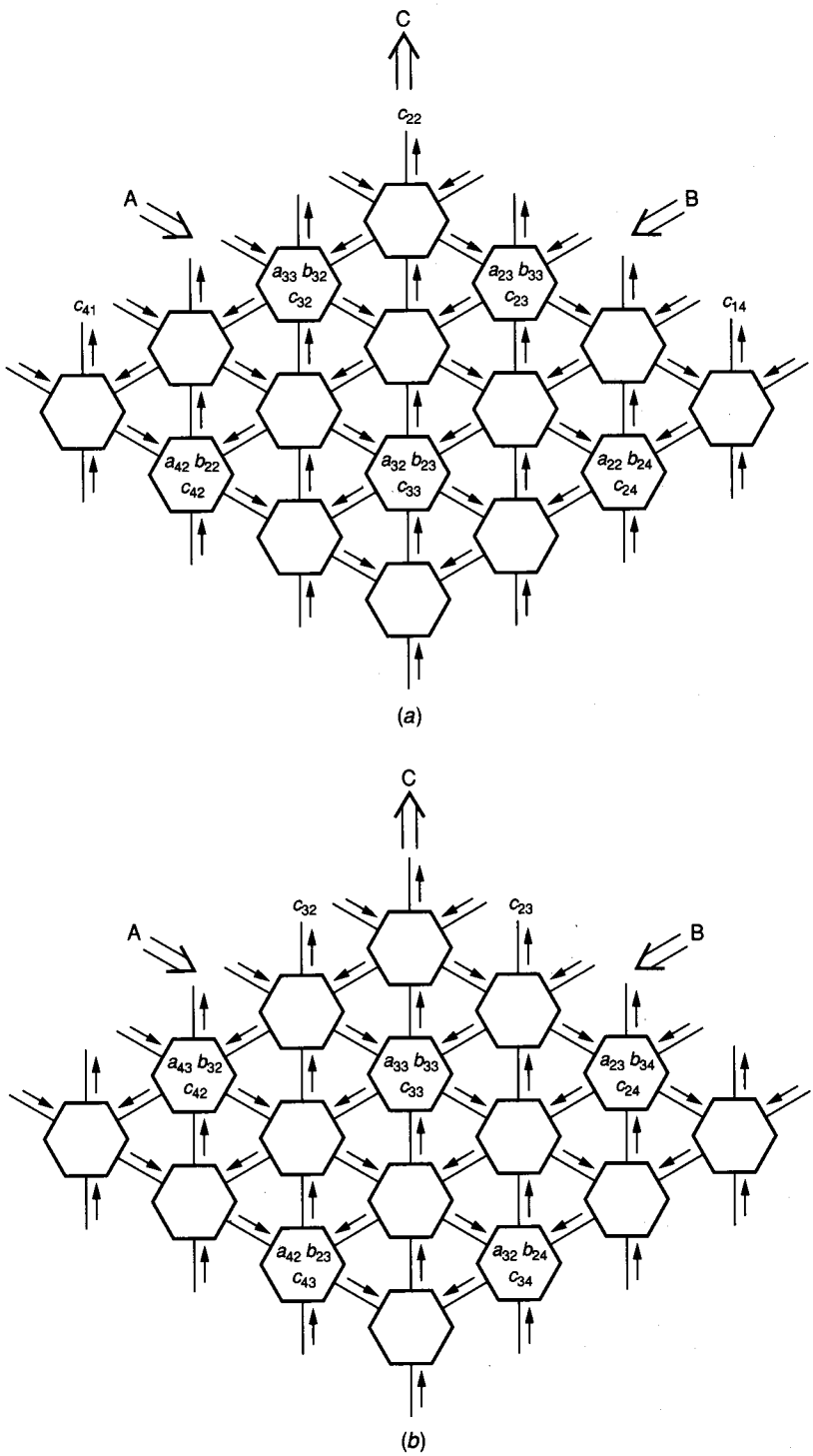


FIGURE 9.16-1
Systolic array multiplication: (*a*) Step n, (*b*) Step $n + 1$.

many engineering problems require a system of equations with feedback for their solution. Consider the general linear system representation as

$$X(k + 1) = AX(k) + Bu(k) \tag{9.16-1}$$

and

$$Y(k + 1) = CX(k) + Du(k) \tag{9.16-2}$$

The $(k + 1)$th value of the state variable vector X depends on the kth value of the state vector. This implies a connection between the output of the state vector calculation and the inputs to the next calculation. This would be difficult with the array structure shown in Fig. 9.16-1. A systolic solution to this problem based on simple processing stages is possible. The solution, shown in Fig. 9.16-2, requires that all processors access a single common bus.

The operation of the state variable solver of Fig. 9.16-2 can be explained as follows. Rows of coefficient values from the state variable matrix are stored in circular shift registers within each processing stage. Only the output value from this circular shift register is shown for each processing element of Fig. 9.16-2. At the start of a new state vector calculation, the accumulator in each processing stage is cleared as shown in step 1. As the calculation starts, $x_1(k)$ is placed on the common bus. Each processing stage simultaneously forms the product $a_{i1}x_1(k)$, where i is the process stage identifier. The resulting product is added to the contents of the accumulator. Next, at the second step, $x_2(k)$ is placed on the common bus; the product $a_{i2}x_2(k)$ is formed; and the result is added to the accumulator. Finally, for an nth order system, at the $(n + 1)$th step $u(k)$ is placed on the bus, the product $b_iu(k)$ is formed, and the result is accumulated. With the completion of this step, the $(k + 1)$th state vector is computed.

If $n + 1$ processing stages are provided, then the output $y(k + 1)$ can be computed simultaneously with the next-state vector. Note that a total of $n + 1$ processors are used to solve for the next-state vector and output value in $n + 1$ iterations. Direct approaches to this problem require n^2 iterations. The single common bus required here is less desirable than completely local interconnections, but a solution for feedback problems with only local communication is not apparent. For the structure of Fig. 9.16-2, a single, large bus driver can be placed at the left end to minimize delays associated with driving the long bus.

9.16.3 Bit-serial Processing Elements

To conclude this section, a look at a simple integrated circuit processing element is appropriate. Even as integrated circuit technology scales to smaller dimensions, the prospect for placing n parallel processing stages on a single silicon die is dim for large n. The size of the typical processing stages and the interconnection buses require too much silicon area. A partial solution to this problem uses bit-serial processing stages. These stages are smaller than their m-bit parallel counterparts by at least a factor of m, allowing m times as many stages to be placed on a silicon die. If the processing stages are designed properly, individual stages will interconnect directly as they are placed, thereby eliminating interconnection buses. This technique of "interconnection by default" is generally useful within integrated circuit design, saving both layout time and silicon area.

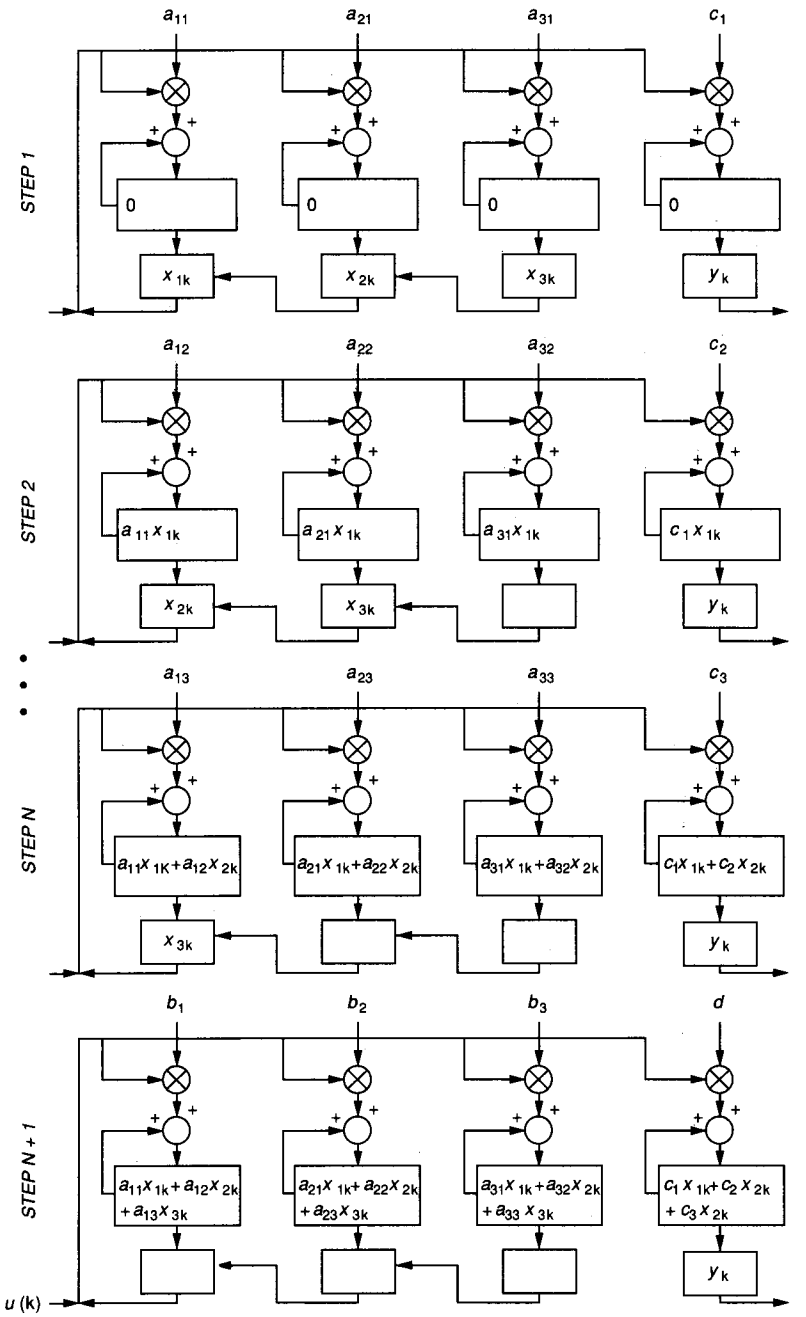


FIGURE 9.16-2
Block diagram for linear equation solver.

Initially, it might appear that bit-serial processing stages are a factor of $1/m$ as fast as parallel processing stages. This could negate the gain achieved by placing m times as many processors on a silicon die. However, bit-serial multipliers and adders can be designed to eliminate carry propagation delay. This allows a net processing speed advantage when m bit-serial processing stages replace a single m-bit parallel processing stage.

An example of a simple bit-serial multiplier is given in Fig. 9.16-3. The algorithm for this particular bit-serial multiplier requires the multiplicand b in parallel and the multiplier a in bit-sequential form. During the first iteration, the first bit of the multiplier a_0 is input and ANDed with the parallel multiplicand b, producing a set of variables called *summands*. The summands are added by the full adders to compute a set of partial product bits and the first product bit P_0. Carries and partial product bits are saved and shifted through unit delay registers so they are available for the next step. As the second multiplier bit a_1 is shifted in, it is ANDed with the multiplicand and added to previous carries and partial product bits. This produces a second bit of the product P_1. At each iteration the weight of each stage doubles, allowing the carry to be fed back within the same stage. This operation continues until the multiplier a is exhausted and the entire product has been shifted out of the multiplier.

The simple multiplier just presented requires one operand in parallel. It is often desirable to accept both inputs in bit-sequential form. This is easily accomplished by using pipeline techniques or through other, slightly more complex bit-serial multipliers.[24]

The capability to map algorithms into hardware with large-scale integrated circuits is revolutionizing the way signal processing is accomplished. The ability to create special-purpose processors to provide parallel solution of time-consuming problems may be the next major step in increasing computational speeds. The concept of systolic processing, with many processors working in lock step fashion, is an important means to achieve this goal.

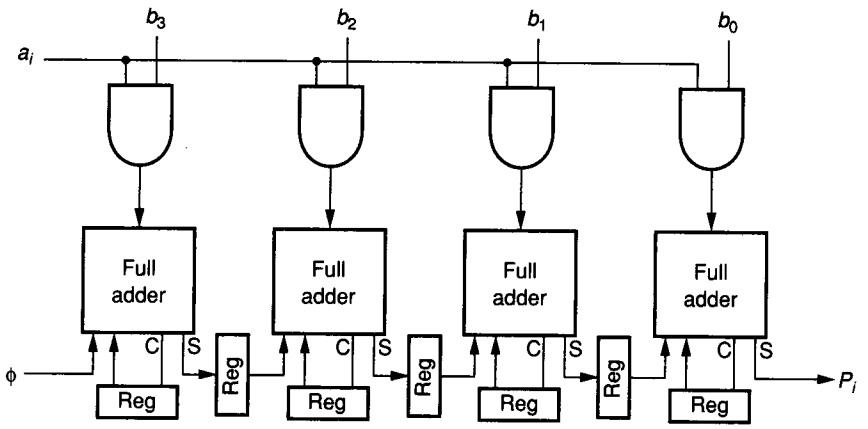


FIGURE 9.16-3
Serial-parallel multiplier.

9.17 SUMMARY

The scope of a chapter describing structured digital circuits and systems is, of necessity, quite broad. The topics presented include structured logic forms such as PLAs, Weinberger arrays, gate matrix layout, and gate arrays (also sea-of-gates). Clocking schemes for digital circuits were introduced as a prerequisite to the treatment of dynamic storage, clocked logic, and sequential machines. Three styles of clocked logic, including C^2MOS, precharge evaluate logic, and domino CMOS logic, were explained. A prototypical semiconductor memory architecture was presented, followed by an investigation of the prominent types of memory storage cells that are used. Salient limitations of both the architecture and the individual cells were provided, usually by example. The memory sections were followed by brief introductions to two widely used forms of sequential machine: PLA FSMs and microcoded FSMs. With these concepts available, an overview of microprocessor architecture was presented to show how the digital subsystems just described can be formed into a complex digital system. And finally, systolic arrays were introduced along with three examples: an array multiplier, a linear system solver, and bit-serial multiplication.

The goal of this chapter was to provide an awareness and basic understanding of structured digital circuits and examples of how these structures are used to form complex digital systems. The information presented on digital integrated circuit structures, along with the prerequisite background in digital system design, allows the design of large digital systems within the integrated circuit medium.

REFERENCES

1. B. Lattin: "VLSI Design Methodology: The Problem of the 80's for Microprocessor Design," *Proc. Caltech Conf. on VLSI*, pp. 247–252, January 1979.
2. D. A. Patterson and C. H. Sequin: "A VLSI RISC," *Computer*, pp. 8–21, September 1982.
3. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli: *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.
4. G. De Micheli and A. Sangiovanni-Vincentelli: "Pleasure: A Computer Program for Simple/-Multiple Constrained Unconstrained Folding of Programmable Logic Arrays," UCB/Electronics Research Laboratory Memorandum M82/57, University of California, Berkeley, August 9, 1982.
5. G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli: "Techniques for Programmable Logic Array Folding," *Proc. 19th Design Automation Conf.*, pp. 147–155, June 1982.
6. A. Weinberger: "Large Scale Integration of MOS Complex Logic: A Layout Method," *IEEE J. Solid-State Electronics*, vol. SC-2, no. 4, pp. 182–190, December 1967.
7. J. M. Siskind, J. R. Southard, and K. W. Crouch: "Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions," *Proc. MIT Conference on Advanced Research in VLSI*, pp. 28–39, January 1982.
8. S. M. Kang, R. H. Krambeck, H. F. S. Law, and A. D. Lopez: "Gate Matrix Layout of Random Control Logic in a 32-Bit CMOS CPU Chip Adaptable to Evolving Logic Design," *Proc. 19th Design Automation Conf.*, pp. 170-174, 1982.
9. A. D. Lopez and H-F. S. Law: "A Dense Gate Matrix Layout Method for MOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-15, no. 4, pp. 736–740, August 1980.
10. G. Dupenloup: "A Wire Routing Scheme for Double-Layer Cell Arrays," *Proc. 21st Design Automation Conf.*, pp. 32–35, June 1984.
11. C. Sechen and A. L. Sangiovanni-Vincentelli: "The TimberWolf Placement and Routing Package," *IEEE J. Solid-State Circuits*," vol. SC-20, no. 2, pp. 510–522, April 1985.

12. J. Y. Chen: "CMOS—The Emerging VLSI Technology," *Proc. IEEE Int. Conf. on Comp. Design*, pp. 130–141, October 1985.
13. W. N. Carr and J. P. Mize: *MOS/LSI Design and Application*, McGraw-Hill, New York, 1972.
14. N. H. E. Weste and K. Eshraghian: *Principles of CMOS VLSI Design*, Addison-Wesley, Reading, Mass., 1985.
15. R. H. Krambeck, C. M. Lee, and H-F. S. Law: "High-Speed Compact Circuits with CMOS," *IEEE J. Solid-State Circuits*," vol. SC-17, no. 3, pp. 614–619, November 6, 1981.
16. J. A. Marques and A. Cunha: "Clocking of VLSI Circuits," *VLSI Architecture*, B. Randell and P. C. Treleaven, eds., Prentice-Hall, Englewood Cliffs, N. J., pp. 165–178, 1983.
17. T. H. Bennett: "Split Low Order Internal Address Bus for Microprocessor," U.S. Patent No. 3,962,682, Motorola, Austin, Texas, June 8, 1976.
18. W. I. Fletcher: *An Engineering Approach to Digital Design*, Prentice-Hall, Englewood Cliffs, N. J., 1980.
19. B. T. Murphy, R. Edwards, L. C. Thomas, and J. J. Molinelli: "A CMOS 32 Bit Single Chip Microprocessor," *ISSCC Digest of Technical Papers*, p. 230, 1981.
20. J. W. Beyers, L. J. Dohse, J. P. Fucetola, R. L. Kochis, C. G. Lob, G. L. Taylor, and E. R. Zeller: "A 32-Bit VLSI CPU Chip," *IEEE J. Solid-State Circuits*, vol. SC-16, pp. 537–542, October 1981.
21. G. Moore: "VLSI: Some Fundamental Challenges" *IEEE Spectrum*, pp. 30–37, April 1979.
22. R. W. Sherburne, Jr., M. G. H. Katevenis, D. A. Patterson, and C. H. Sequin: "Datapath Design for RISC" *Proc. MIT Conf. on Advanced Research in VLSI*, pp. 52–62, January 1982.
23. H. T. Kung: "Let's Design Algorithms for VLSI Systems," *Proc. Caltech Conf. on VLSI*, pp. 65–90, January 1979.
24. N. R. Strader and V. T. Rhyne: "A Canonical Bit-sequential Multiplier," *IEEE Trans. Comput.*, vol. C-31, no. 8, pp. 791–795, August 1982.

PROBLEMS

Section 9.1

9.1. If a manufacturer's layout rate is eight transistors/day and a chip regularization factor of 20 is achieved, how many man-years are required to lay out a 300,000 transistor chip?

9.2. Estimate the layout time for the Intel 8086 and the Motorola 68000 if a layout rate of 10 transistors/day is assumed for each (use Table 9.1-1).

Section 9.2

9.3. Determine the sizing triplet (i, p, o) for a PLA that implements the following logic equations.

$$X = PB + D\overline{B} + \overline{P}\,\overline{D}A$$

$$Y = P\overline{A} + DA + \overline{P}\,\overline{D}A$$

$$Z = PB + DA$$

9.4. In terms of λ, estimate the minimum pitch for repeating the AND plane section of Fig. 9.2-5*c* using the design rules of Table 2B.2 of Appendix 2B. Consider the spacings required to connect the rotated AND section into the OR plane. *Pitch* is defined as the repetition spacing for repeated placements of layout segments.

9.5. Find a folded PLA realization for the following equations so that the PLA sizing triplet is reduced from a straightforward implementation. Can you determine if this is the minimum realization? Why or why not?

$$X = ABC + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

$$Y = ABC + \overline{B}\,\overline{C}$$

$$Z = ABC$$

$$R = \overline{A}BC + AB\overline{C}$$

$$S = \overline{A}\,\overline{B} + AB\overline{C} + \overline{B}C$$

Section 9.3

9.6. Show an NMOS Weinberger NOR array implementation for the logic equations in Prob. 9.3.

9.7. If no rows are shared between input terms and intermediate terms in a Weinberger NOR array, compute the area required to implement the logic equations in Prob. 9.3 as the product of the rows and columns. Compare with a PLA implementation for the same logic equations using the same metric.

9.8. What logic functions are defined by the symbolic gate matrix layout of Fig. 9.3-5?

9.9. Provide a symbolic gate matrix layout for the logic equations of Prob. 9.3.

Section 9.4

9.10. The logic equations of Prob. 9.3 are to be implemented in a CMOS gate array using logic function blocks from the library of Table 9.4-1. Provide a list of the blocks required and the specific logic equations implemented by each block for this design. Use five or fewer types of standard logic function blocks.

9.11. Compare the number of transistors required for a straightforward CMOS implementation of the logic equations of Prob. 9.3 and the number of transistors required with a gate array implementation using blocks from Table 9.4-1. Note that each standard logic function block in Table 9.4-1 is composed of a multiple of 12 transistors (6 p-channel and 6 n-channel).

Section 9.5

9.12. For the circuit of Fig. 9.5-4*a*, let NOR2 be the gate connected directly to ϕ_2, NOR1 be the gate connected to ϕ_1, and INV represent the inverter. Identify the gate or gates that cause each of the four delays shown in Fig. 9.5-4*b*. If the delays are 3, 4, and 5 ns for the INV, NOR2, and NOR1 gates, respectively, find the nonoverlap time following (*a*) ϕ_2 and (*b*) ϕ_1.

9.13. If a pair of inverters is added between each NOR gate output and its feedback connection in Fig. 9.5-4*a*, show the resulting clock waveforms as in Fig. 9.5-4*b*. Assume unit delays for all gates and inverters. (Note that such inverter pairs could be sized geometrically to increase clock drive capability.)

Section 9.6

9.14. For an NMOS dynamic storage circuit with a pass transistor drain diffusion area of 20 μ^2, a metal interconnect area of 20 μ^2, a non-gate polysilicon area of 10 μ^2 and an inverter gate area of 6 μ^2, calculate an equivalent dynamic storage time constant if $C_{\text{diff}} = 0.04$ fF/μ^2, $C_{\text{met}} = 0.03$ fF/μ^2, $C_{\text{poly}} = 0.03$ fF/μ^2, $C_{\text{ox}} = 0.4$ fF/μ^2, and the leakage current $I_r = 0.5$ fA/μ^2.

9.15. In what ways does the use of a CMOS transmission gate rather than an NMOS pass transistor affect the storage time of a dynamic storage circuit? Explain.

9.16. The level restorer transistor of Fig. 9.6-1*c* must be sized above some minimum value and below some maximum value. What limits the minimum and maximum allowable resistances for this transistor?

9.17. Consider the linear shift register of Fig. 9.6-2. If the input to stage A is brought high during ϕ_1 of a first clock cycle and then left low, prepare a timing diagram showing the clock signals ϕ_1 and ϕ_2 and the outputs of each of the four shift stages for five consecutive clock cycles.

9.18. Prepare a symbolic layout diagram (see Fig. 9.6-4) that shows a clocked shift register that provides a left shift, no shift, or a right shift for four parallel bits.

Section 9.7

9.19. Generate geometrical layouts for the shift register circuits of Figs. 9.7-1 and 9.7-2. Compare the complexity of the two layouts, particularly noting the number of contacts required.

9.20. Provide circuits to implement the following logic equations using complex P-E gate structures as in Fig. 9.7-4.

$$R = AB + BC + AC$$

$$S = ABC + \overline{A}\,\overline{B}\,\overline{C}$$

9.21. Assume the drain and source of each transistor in Fig. 9.7-4 contribute an equal incremental capacitance at each node. If the node between the transistors gated by signals E and F is low, the signals D, B, and A are each low, the gate output is precharged high, the clock is high, and the input signal F is then driven high, calculate the effect of charge sharing on the output voltage if load capacitance caused by subsequent connections is ignored.

9.22 P-E logic stages cannot be directly cascaded. Consider alternating P-E stages of n-channel logic clocked by clk1 with P-E stages of p-channel logic clocked by clk2. Can clk1 and clk2 be complementary signals? What conditions must clk1 and clk2 satisfy for this cascade configuration to work correctly?

9.23. Is it possible to realize the exclusive-OR function using only domino logic circuits? Why or why not?

9.24. Create a table to compare static NMOS, static CMOS, P-E CMOS, and domino CMOS logic gates in terms of total transistor count, static power dissipation (yes or no), and output availability (duty cycle).

Section 9.8

9.25. *IEEE Spectrum* publishes a technology update issue in January of each year. Prepare a 1–2-page summary of the state of the art in semiconductor memories based on the most recent update issue.

9.26. Quantify the row and column decode circuitry required for a 1M × 1-bit memory (*a*) organized as a square array and (*b*) organized with four more address lines feeding the row decoder than the number of address lines feeding the column decoder.

Section 9.9

9.27. Assume the charge on the gate of an EPROM cell is 10×10^{-15} C. Assume the gate loses charge exponentially with a time constant of 10 years. Calculate the equivalent leakage resistance from the gate to ground if the gate capacitance is 4 fF.

9.28. Provide the layout for a macro defining the repeatable layout for a ROM memory cell in the technology of Appendix 2B. Now estimate the area required for a 256k-bit memory based on your cell layout. Estimate the size of a repeatable memory cell in a commercial 256k-bit memory area if the array size is 10 mm^2.

Section 9.10

9.29. Consider a 256k SRAM with row lines fabricated as 2 μ polysilicon runs. Assume the row select transistors are 2 μ × 4 μ and the select lines are 5 mm long. Assuming a square memory array organization, (*a*) provide a reasonable estimate of the delay across the select line, and (*b*) provide a reasonable estimate of the delay across the select line if the row decoder divides the memory array into two equal parts to halve the select line length. Use parameters from Appendix 2B.

9.30. For a 256k SRAM that dissipates 0.4 W in the memory array, estimate the resistance of the polysilicon load resistors for the memory cells.

Section 9.11

9.31. If a 1M DRAM is built from cells with 40 fF storage capacitance and the minimum differential voltage that will reliably trigger the sense amplifiers is 80 mV, determine the maximum allowable capacitance for the word lines.

9.32. Assume that a DRAM storage cell is built using a 50 fF capacitor and that a voltage of at least 3.8 V can be reliably recognized as a logic high. If a memory refresh period of 2 s is sufficient under average conditions, estimate the total leakage resistance from the storage cell to ground. Assume a memory refresh charges the capacitor to 5 V.

9.33. In the design of a DRAM memory array, it is found that the delay along the polysilicon row select line is too great, and the suggestion is made to widen the row select line to reduce its equivalent resistance. Will this solution help reduce the select line delay? Why or why not? Will this have other, detrimental effects on the memory array? Explain.

Section 9.12

9.34. Show how to modify Fig. 9.12-5 for dual-port read capability.

9.35. The data lines of a static register array are precharged to the supply voltage and have much larger capacitances than the register cell output. If the select line couples a logic low output of the register cell to the precharged data line during a read operation, what may happen to the register cell contents? Explain.

9.36. For a static register cell, determine a constraint on the relative size of the inverter pulldown transistors and the select transistors to prevent disturbing the register cell state through charge sharing during a read operation with precharged data buses.

9.37. Using the cross-coupled NOR structure for the static register cell array, is it reasonable to provide a dual-port write capability? Explain.

Section 9.13

9.38. A Gray Code counter has the distinguishing feature that successive counts never differ in more than one bit. Thus, a 2-bit Gray Code counter counts as 00, 01, 11, 10, 00, 01, etc. An application requires a Gray Code counter that can count up when an UP control line is asserted or down when a DOWN control line is asserted. If both controls are low, the counter remains in its present state. External

circuitry prevents the condition of both control inputs being high simultaneously. Provide a state diagram, a state transition table, logic equations, and a logic/circuit diagram of a PLA FSM to implement this counter.

Section 9.14

9.39. For a typical microROM such as that of Fig. 9.14-2, consider the number of unique 72-bit control words available. With an execution rate of 10 MHz, how long would it take to execute each control word exactly once?

9.40. Considering the typical ROM organization described in previous sections, compare the maximum number of rows and columns that must be traversed during a bit access using the single-level microROM of Fig. 9.14-2 and the two-level microprogram memory of Fig. 9.14-3. Explain how this would affect access time for the two memories.

9.41. Using the description of Example 9.14-1, list the control words necessary to perform the same add operation if the register array is single-port read rather than dual-port read.

Section 9.15

9.42. Estimate and compare the percentages of total area consumed by regular structures on the die photographs of Fig. 9.15-1 and Fig. 9.15-2.

9.43. Assume the data path of a microprocessor has a 32-bit by 16 register array, a 0 to 32-bit shift capability for the barrel shifter, and the ALU operations of Table 9.15-1. Ignoring timing and temporary results storage, specify the control lines required to operate this data path.

9.44. Using the barrel shifter of Fig. 9.15-6, indicate the states of the control lines and the source and destination buses necessary to perform (*a*) a shift left by 2 bits (quadruple the magnitude of the input), and (*b*) a shift right by 1 bit (halve the magnitude of the input).

9.45. Four flag bits are common for most microprocessor ALUs. These include the C, N, Z, and V bits (carry, negative, zero, and overflow, respectively).
(*a*) The Z bit is normally generated using a distributed NOR gate structure at the ALU output. Show how this might be accomplished.
(*b*) Explain how the N bit could be generated.
(*c*) Explain how the C bit could be generated.
(*d*) The V bit can be generated as the exclusive-OR of the final carry bit and the penultimate (next to the highest) carry bit. Demonstrate that this is logically correct for two's complement arithmetic.

Section 9.16

9.46. Using the systolic array of Fig. 9.16-1 as a guide, show the systolic array structure required to multiply two 4×4 square arrays to produce a 4×4 result. How many processors are required?

9.47. For the serial-parallel multiplier of Fig. 9.16-3, how many steps are required to multiply two 4-bit numbers? Show the value of the sum bits and the carry bits for each step when multiplying 6 by 5.

9.48. Symbolically show the hand multiplication of two 4-bit numbers a and b. Demonstrate that the ith product bit is a function of only the ith and lower-order multiplicand and multiplier bits.