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

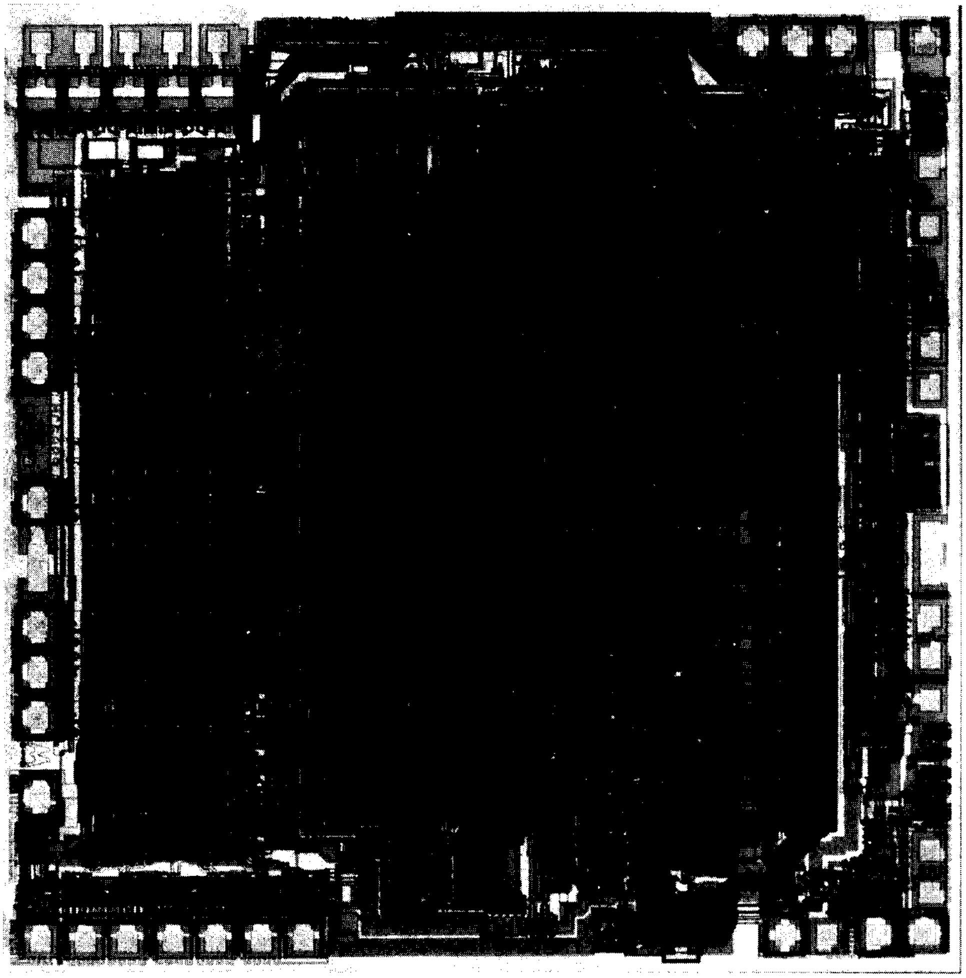


FIGURE 9.1-1
Die photo for Motorola 6809 microprocessor (Courtesy Motorola Inc.).

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.

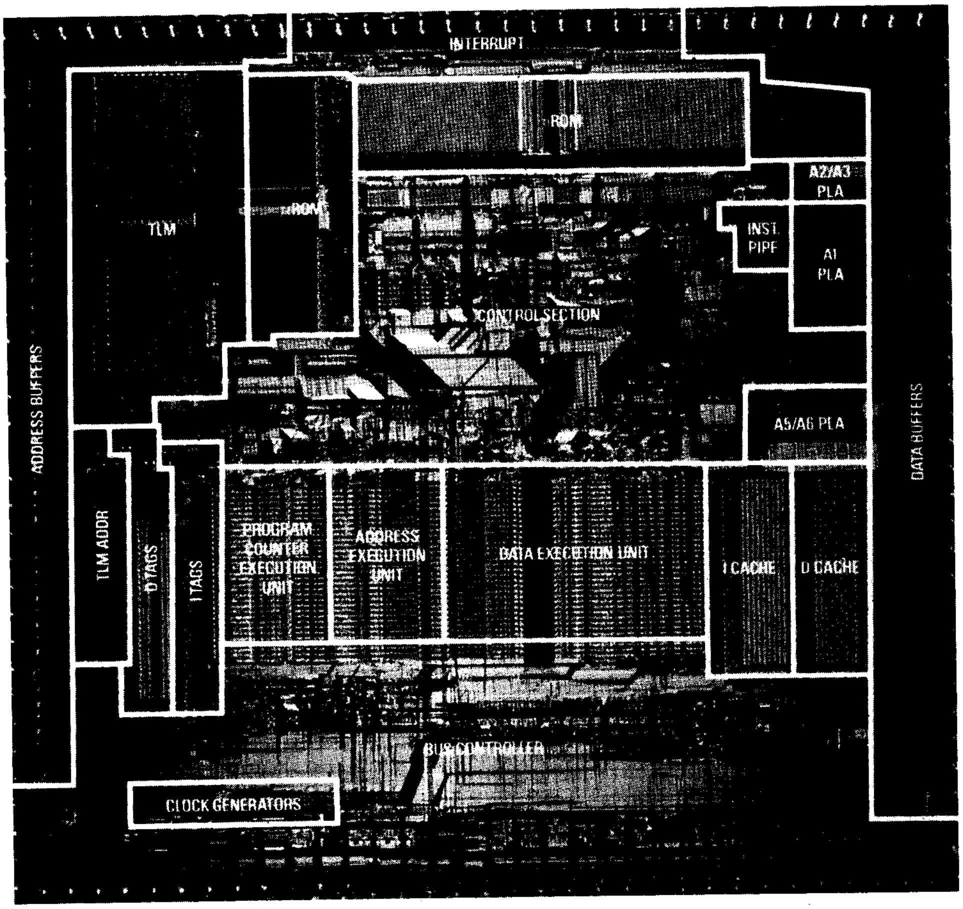


FIGURE 9.1-3
Die photo for Motorola 68030 microprocessor (Courtesy Motorola Inc.).

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.

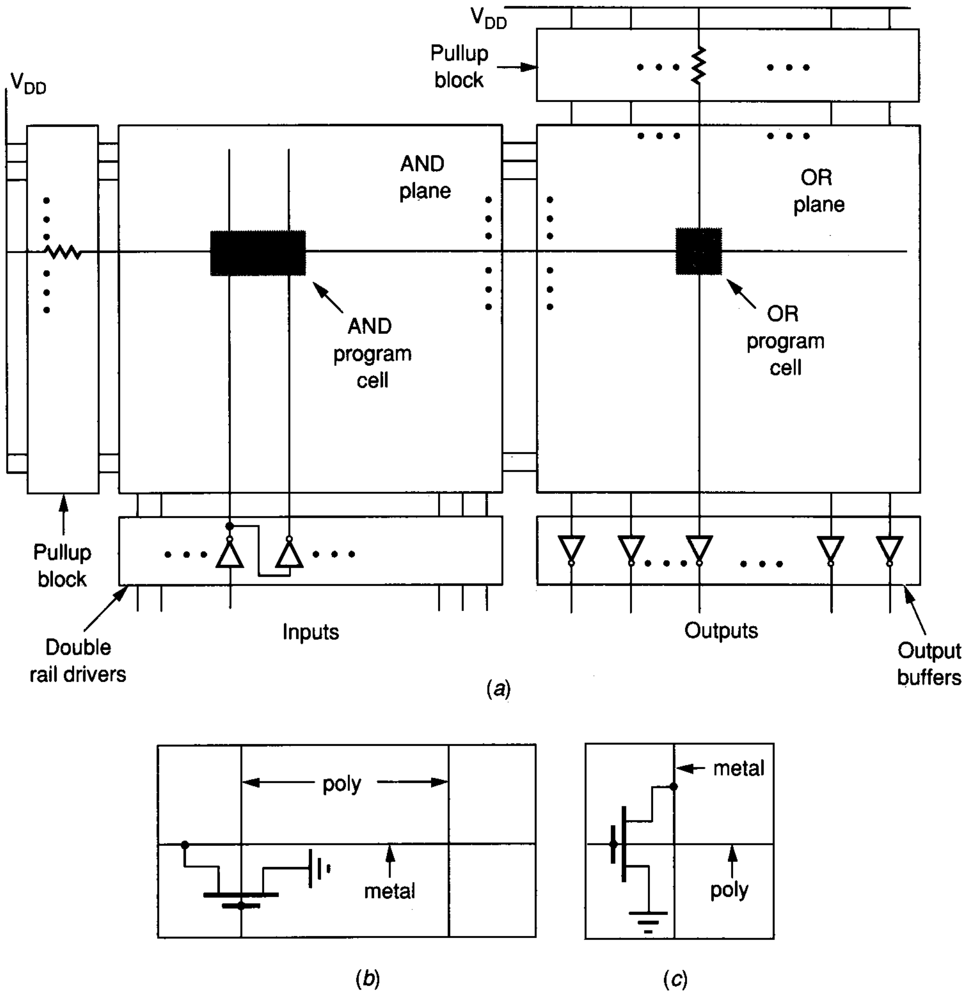


FIGURE 9.2-1
Simplified PLA architecture: *(a)* Major functional blocks, *(b)* AND program cell, *(c)* OR program cell.

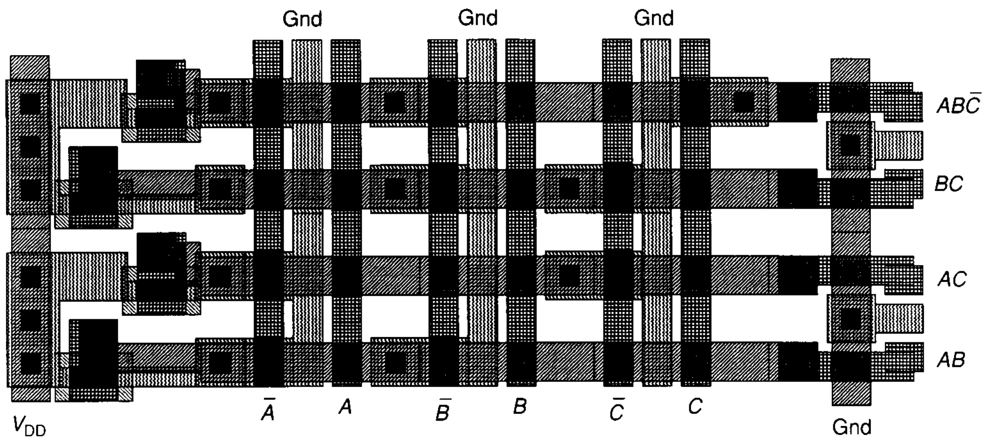
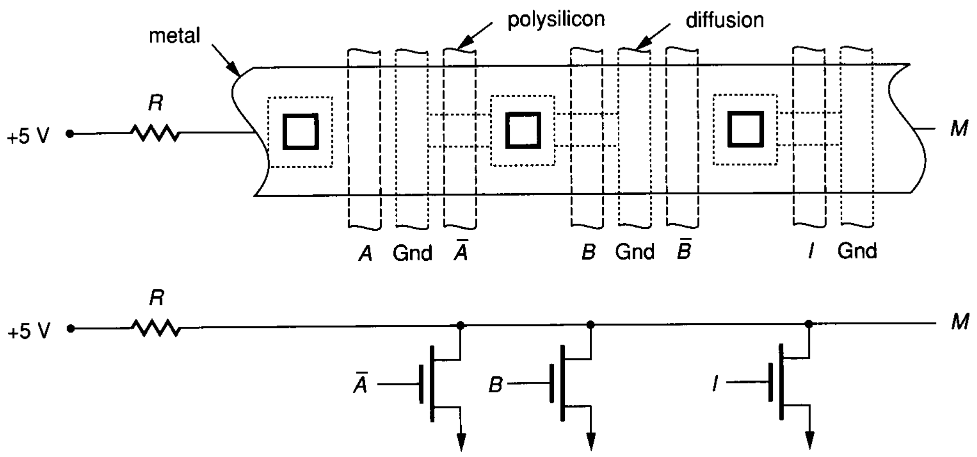


FIGURE 9.2-2
PLA AND-plane detail.

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



$$\left.\begin{aligned} M &= \overline{(\bar{A} + B + I)} \\ &= A\bar{B}\bar{I} \end{aligned}\right\} \text{by DeMorgan's theorem}$$

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.

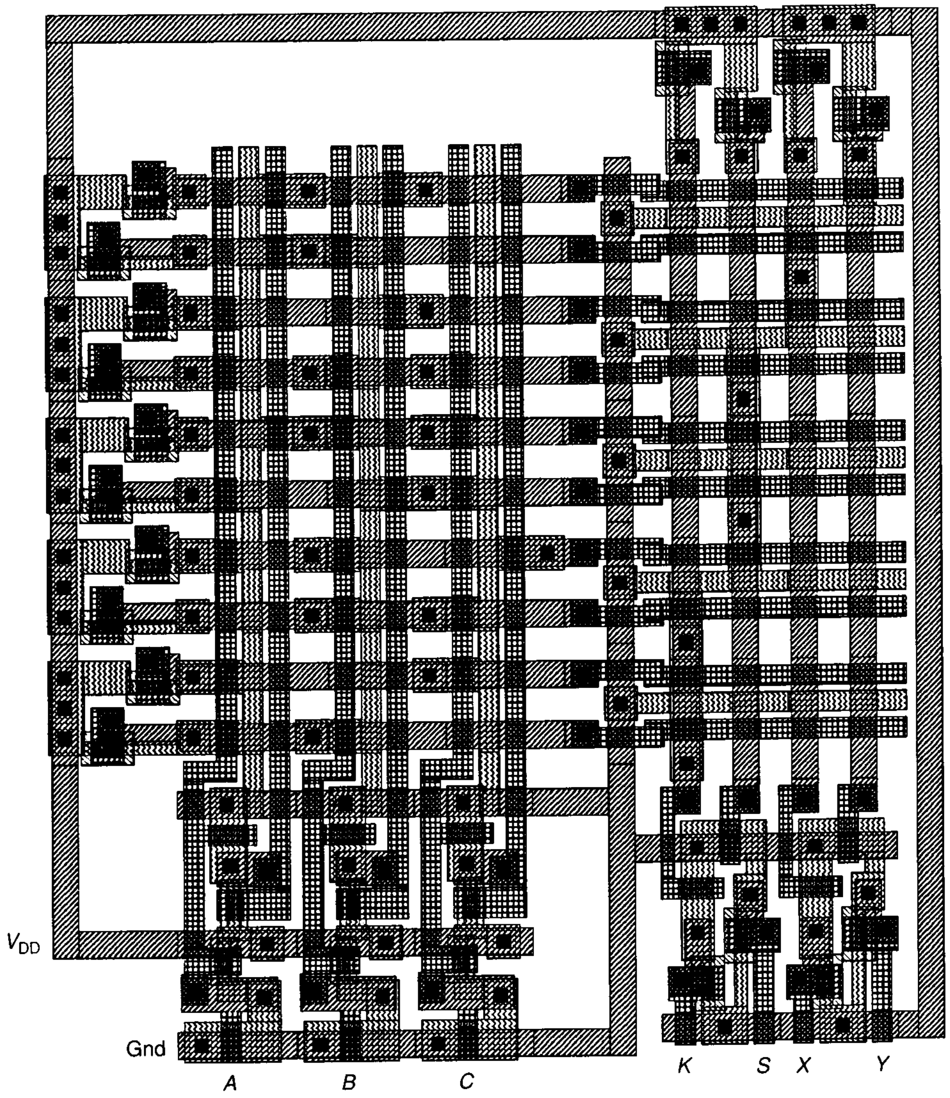


FIGURE 9.2-4
PLA layout for Eqs. 9.2-1 through 9.2-4.

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 $(2i + o)p$.

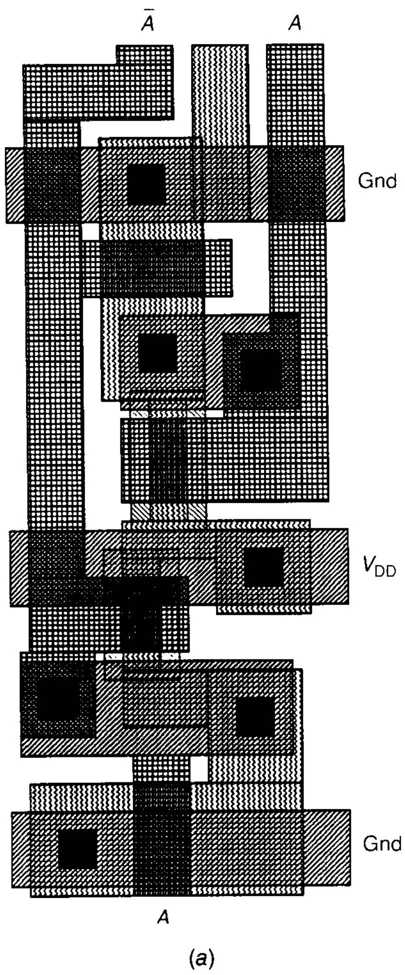


FIGURE 9.2-5
Basic PLA cells: *(a)* Double-rail driver, *(b)* Pull-up pair, *(c)* AND-plane section, *(d)* AND-OR plane connection, *(e)* Inverting buffer pair, *(f)* Programming plug.

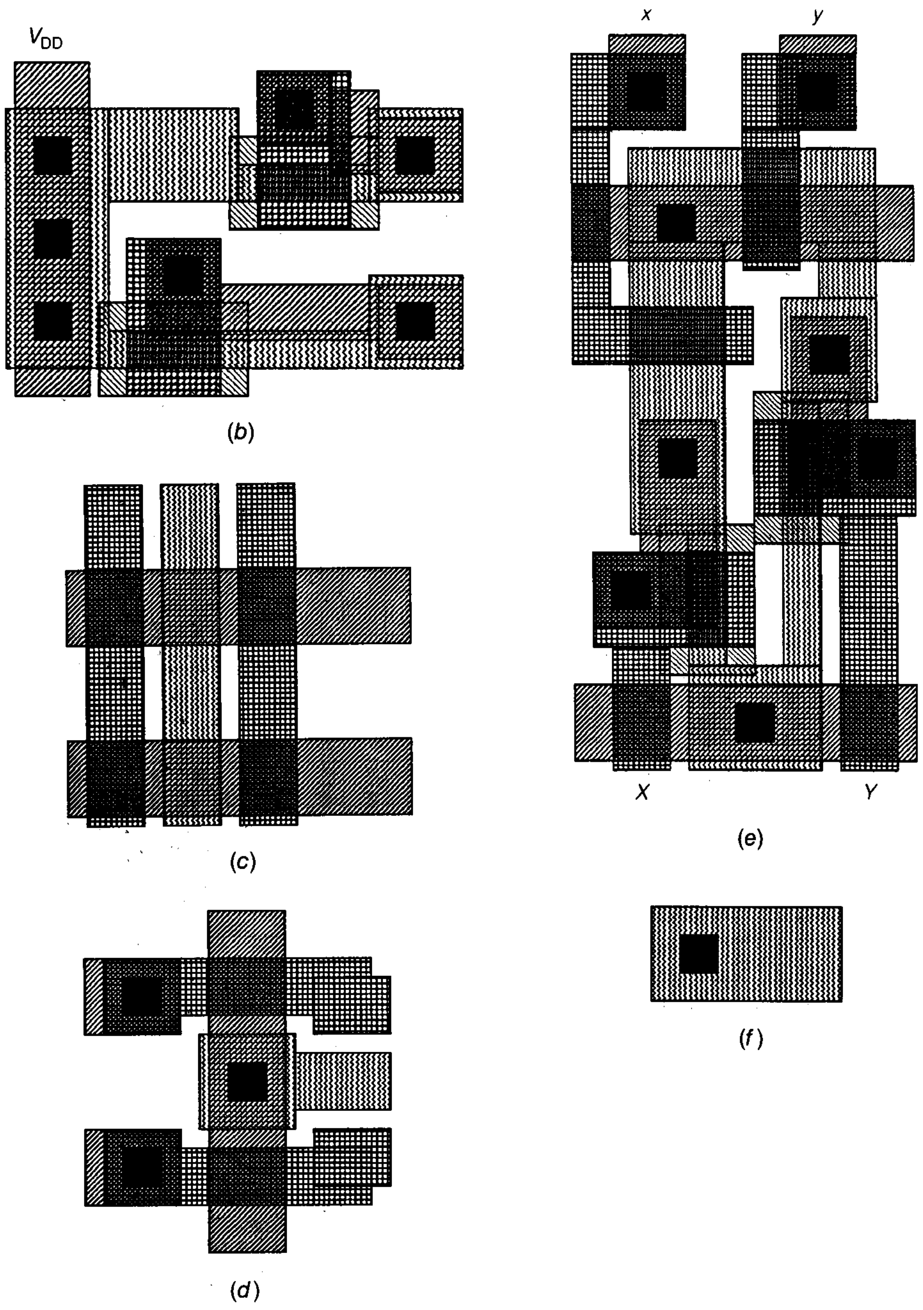


FIGURE 9.2-5
(*Continued*)

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.

AND plane												OR plane						
.	.	.	.	.	.	.	x	.	x	.	x	.	.	.	.	x	.	$\bar{D}\bar{E}\bar{F}$
.	.	.	.	.	.	.	.	x	.	x	.	x	.	.	.	x	x	DEF
.	.	.	.	.	.	.	.	x	x	.	.	.	.	.	x	.	x	$D\bar{E}$
.	.	.	.	.	.	x	.	.	x	.	.	.	.	.	x	.	.	$\bar{D}E$
.	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	$\bar{C}$	D	$\bar{D}$	E	$\bar{E}$	F	$\bar{F}$	S	K	R	W	X	Y	

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

AND plane						OR plane						
D	$\bar{D}$	E	$\bar{E}$	F	$\bar{F}$							
y	.	y	.	y	.	.	.	.	.	x	.	$\bar{D}\bar{E}\bar{F}$
.	y	.	y	.	y	.	.	.	.	x	x	DEF
.	y	y	.	.	.	.	.	.	x	.	x	$D\bar{E}$
y	.	.	y	.	.	.	.	.	x	.	.	$\bar{D}E$
.	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	W	X	Y	

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.

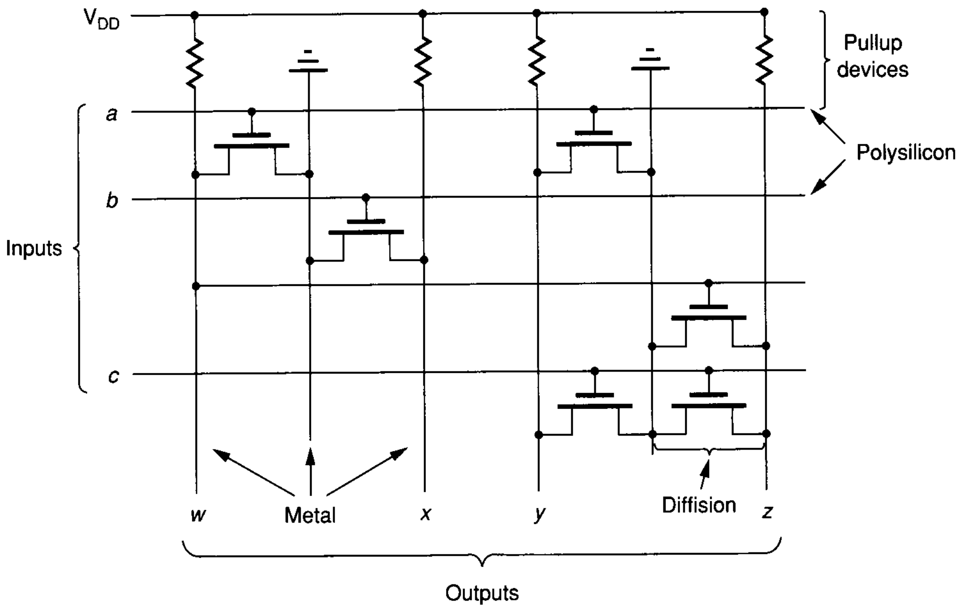


FIGURE 9.3-1
Weinberger NOR array organization with $w = \bar{a}$, $x = \bar{b}$, $y = \overline{a + c}$, $z = \overline{w + c}$.

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

$$\overline{X} = \overline{A}\,\overline{B} + AB \qquad (9.3\text{-}1)$$

and converting to the NOR-NOR form as

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

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.

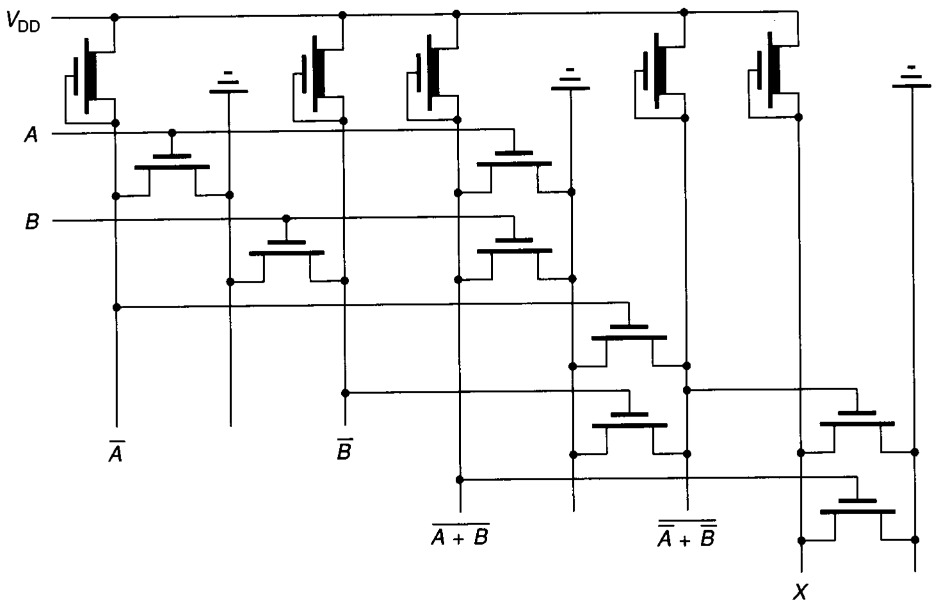


FIGURE 9.3-2
NMOS Weinberger array implementation of exclusive-OR function.

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-

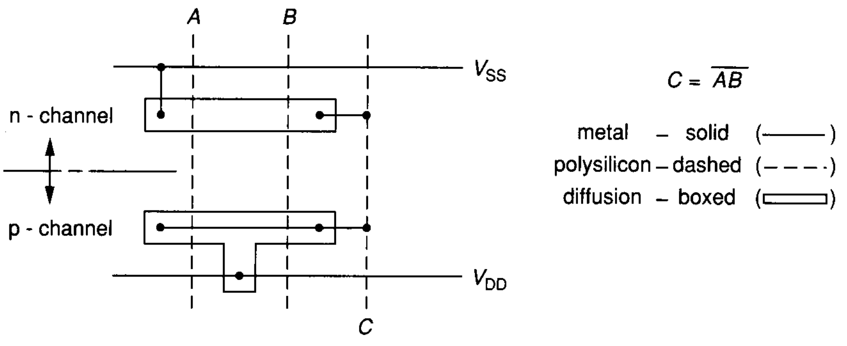


FIGURE 9.3-3
Gate matrix layout for NAND gate.

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

```
            A     B
VSS  -  +  +  -  +  +
        :  ¦     ¦  ¦
        *  N     N  *
           ¦     ¦  ¦
        *  P  +  P  *
           ¦  !  ¦  ¦
VDD  -  -  +  *  +  +
                    C
```

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

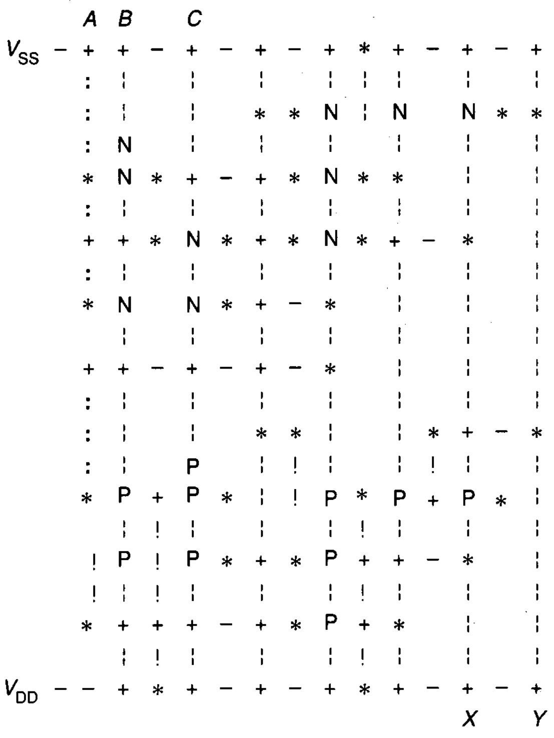


FIGURE 9.3-5
Symbolic gate matrix layout description for 3-input, 2-output logic function (see Prob. 9.8).

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.

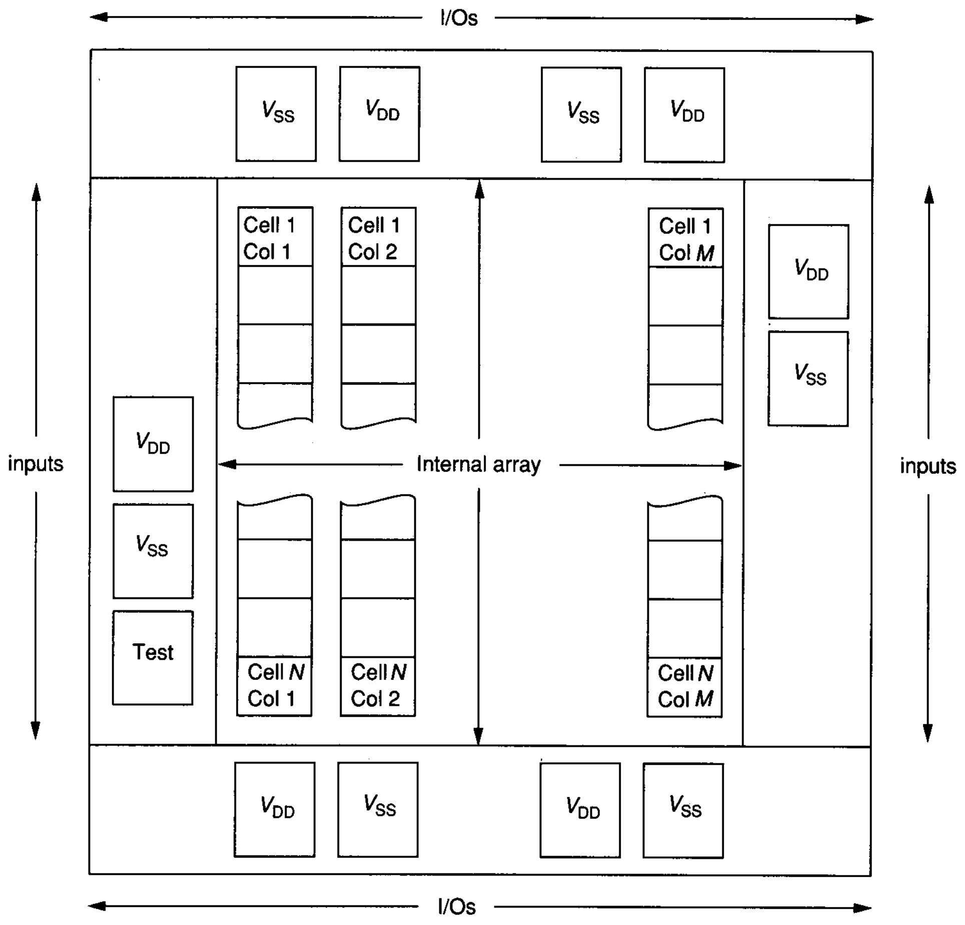


FIGURE 9.4-1
Topology for typical gate array.

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

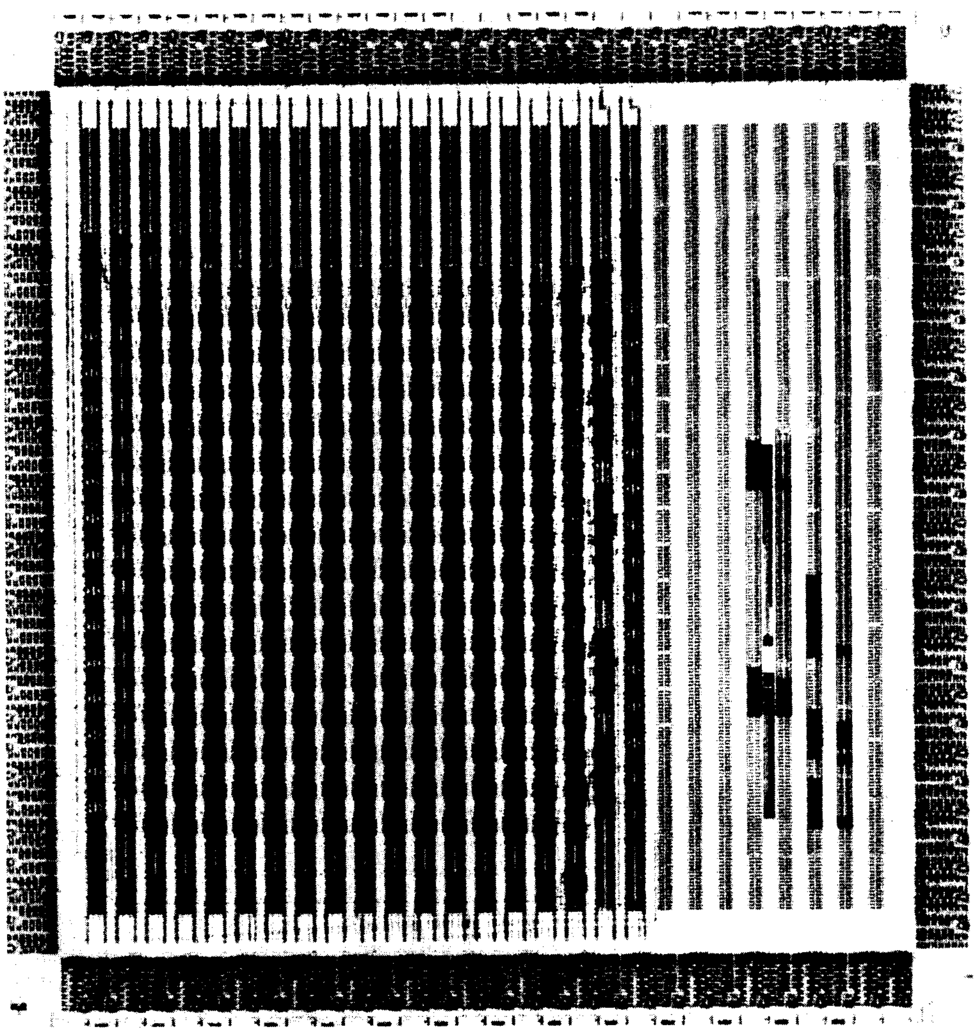


FIGURE 9.4-2
CMOS gate array chip (©IEEE 1982, Kobayashi et. al., ISSCC Proc., p. 316).

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.