CHAPTER 10

DESIGN AUTOMATION AND VERIFICATION

10.0 INTRODUCTION

Design automation and design verification are the keys to effective use of large-scale integrated circuit technology today. When circuits consisted of only a few transistors or gates, layout and checking of circuits by hand were reasonable. As circuit complexity increased to thousands and tens of thousands of transistors, manual tools were no longer sufficient for design, causing computer-based design aids to become prominent. With present integrated circuits containing hundreds of thousands of transistors, heavy dependence on design automation and design verification is necessary to design these circuits.

This chapter describes the nature and use of basic design automation and design verification tools as applied to the design of integrated circuits. *Design automation tools* are defined here as those computer-based tools that assist through automation of procedures that would otherwise be performed manually, if at all. Simulation of proposed design functionality and synthesis of integrated circuit logic and layout are just two examples. *Design verification tools*, on the other hand, are those computer-based tools used to verify that circuit design or layout meets certain prescribed objectives. A geometrical design rule checker for examining layout characteristics is an example, and a logic simulator with a specific set of input vectors and corresponding desired output vectors is another. Note that simulation can be classified in either category according to its purpose. Both design automation tools and design verification tools are included in the more general class known as CAD (computer-aided design) tools.

Both design automation and design verification tools require computer-readable descriptions of the underlying circuit function and structure to operate. These computer-based descriptions vary from simple geometrical specification languages such as CIF[1] (Caltech Intermediate Form) to high-level functional description languages such as VHDL[2] (a hardware design language). Initially the focus of this chapter is on a description of design tools related to or based on geometrical layout. A simplified geometrical specification language will be examined. Required functionality provided by tools that input and display integrated circuit layout will also be described. Then, tools that check layout geometries and extract circuit net list information will be detailed.

Design tools for higher-level design description and verification are described next. Circuit, switch, and logic simulation for digital circuits are introduced and compared. Timing analysis is examined as a way to verify the temporal operation of digital circuits. Hardware design languages such as VHDL and EDIF[3] (Electronic Design Interchange Format) are introduced with simple examples provided to clarify important concepts.

The descriptions of design verification and design automation tools provided here use MOS examples primarily. The concepts are directly applicable to bipolar designs, although some changes in specific tool capability may be required by different technologies. The chapter concludes with an introduction to automated methods of generating layout from high-level descriptions of digital circuits via silicon compilers.

10.1 INTEGRATED CIRCUIT LAYOUT

Historically, integrated circuit design and integrated circuit layout functions were performed by separate groups. The circuit design task resulted in mixed logic and transistor-level circuit diagrams describing the intended circuits. A circuit description like that of Fig. 10.1-1 was given to layout artists, who were experts at converting circuit diagrams to geometrical layouts such as the one shown in Fig. 10.1-2. For early commercial products, the layout drawings were transferred to rubylith masks by hand. Later, layouts were drawn on vellum—a tough, semitransparent drafting material—to withstand the many design modifications

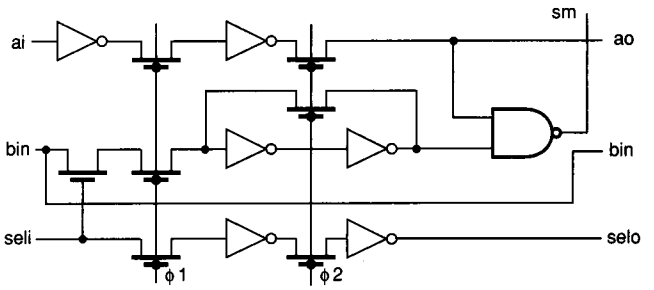


FIGURE 10.1-1
Partial circuit diagram for bit-serial multiplier of Fig. 10.1-2.

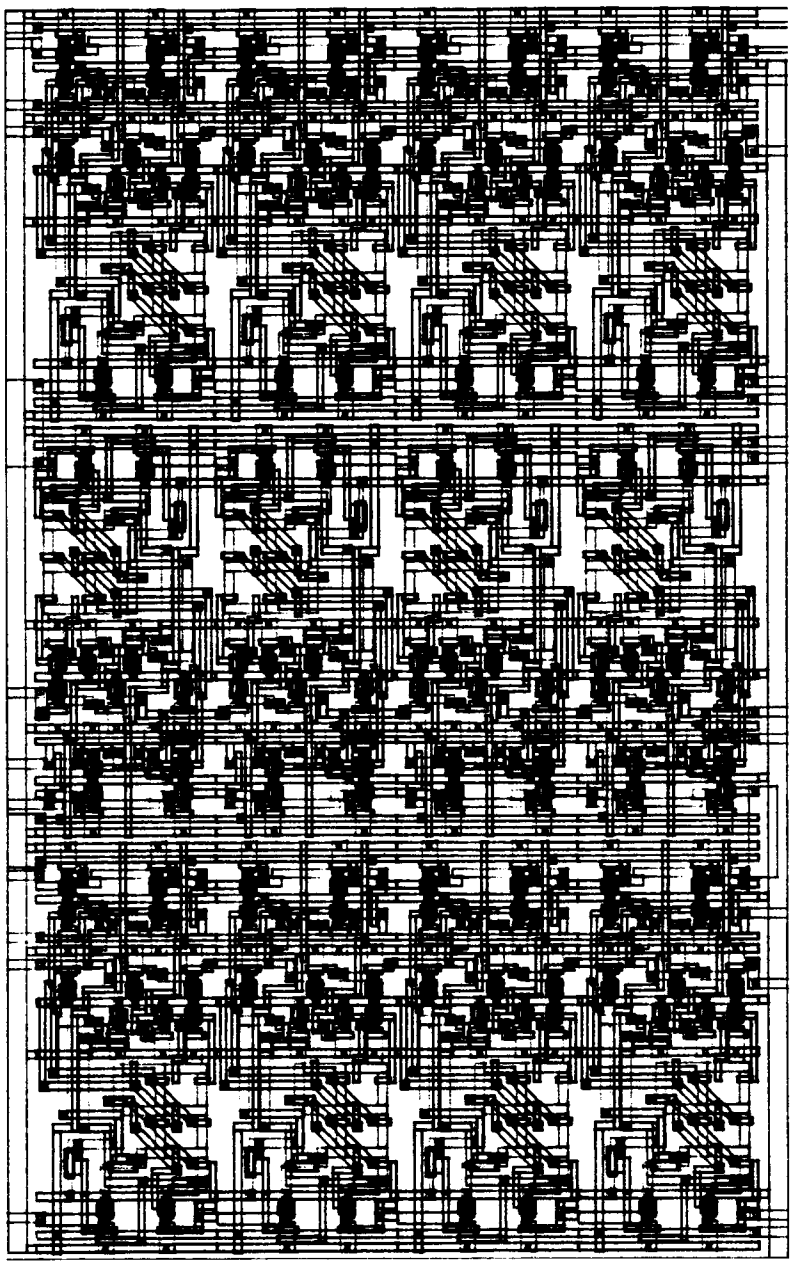


FIGURE 10.1-2
Layout for bit-serial multiplier based on circuit of Fig. 10.1-1.

that are inherent in the normal design process. The layouts from the large vellum plots were digitized to computer-readable form to allow automated checks and to provide input to the mask-making process. Although this method worked for many years, including the early days of microprocessors, the large number of devices required in modern integrated circuits causes fully manual layout to be too time-consuming and prone to error. However, even today, critical sections of the newest microprocessors are still handcrafted to pack the circuit into the smallest possible area.

Many modern methods of integrated circuit layout include both synthesis of control logic and handcrafting of critical building blocks that will be repeated. These layout pieces are entered into a computer at an early stage to allow mechanized help with replicating, checking, and plotting the complete integrated circuit layout. Design layouts may be entered via tools that help convert graphic layout information to computer-readable form. An early tool, shown in Fig. 10.1-3, is called a *digitizer* and was used to enter layout coordinates directly into a computer from a layout plot. Sometimes layout is converted directly to text input in the form of a geometrical specification language. Most often, geometrical layout information is entered through a color graphics workstation to specify the desired integrated circuit layout.

10.1.1 Geometrical Specification Languages

Geometrical specification languages for integrated circuits allow computer-readable definition of the geometries for the mask layers required to fabricate an integrated circuit. These specification languages contain primitive structures such as wires and boxes to specify geometrical shapes and layout levels. Organizational constructs are also provided to allow placement and repetition of the geometrical structures. A geometrical specification language is much like a computer programming language, with the geometrical shape primitives corresponding to instructions and the organizational constructs corresponding to procedures with parameter values.

FIGURE 10.1-3
Digitizer board.

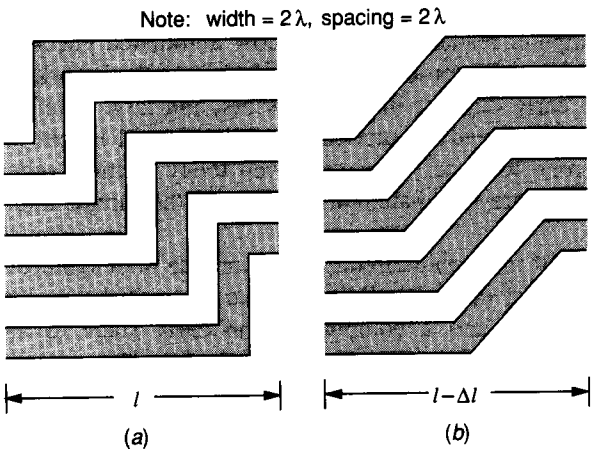


FIGURE 10.1-4
Layout Styles: (*a*) Manhattan, (*b*) Diagonal.

A simplified geometrical specification language for Manhattan style designs for a general MOS process is used here to illustrate relevant concepts. A *Manhattan design style* is one that supports only horizontal and vertical geometries. The name arises because Manhattan style layouts resemble an aerial view of the street layout of New York's Manhattan borough. This style precludes diagonal structures, such as interconnection jogs, that are sometimes used within circuit layouts to minimize area. Figure 10.1-4 shows layout styles with and without diagonal structures. The potential area savings with diagonal structures must be weighed against the increased complexity of programs used to verify the final design. Many commercial integrated circuit manufacturers allow diagonal layout structures but limit these to 45° angles from horizontal and vertical structures.

The simplified geometrical specification language defined here provides only two primitive statements. The two primitives are *boxes* and *levels*, while the organizational constructs include *macros* and *calls*. A macro is like a high-level language (HLL) procedure, and a call is like an HLL procedure call. Table 10.1-1 provides the syntax for these primitives and organizational constructs.

All parameter values are integers. Lengths are in terms of λ, a measure related to the characteristic resolution of the process and the layout design rule set. Macro numbers, layout levels, and orientations are limited to positive integers. A minimum set of layers for a typical NMOS n-well CMOS process is defined in Table 10.1-2. Appendices 2A and 2B define corresponding layers for a double polysilicon NMOS and a p-well CMOS process, respectively.

TABLE 10.1-1
Simplified geometrical specification language

B *x y dx dy*	Box structure with length *dx*, width *dy*, and lower left-hand corner placed at *x*, *y*
L *n*	Layout level for the box definitions that follow
M *n*	Start of macro number *n*
E	End of a macro
C *n x y m*	Call for macro number *n* with translation *x*, *y* and orientation *m*
Q	End of layout file

TABLE 10.1-2
MOS layer definitions

Layer	CMOS	NMOS
1	n-diffusion	n-diffusion
2	p-diffusion	Ion implant
3	Polysilicon	Polysilicon
4	Metal	Metal
5	Contact	Contact
8	n-well	—
9	Overglass	Overglass

The orientation represents possible rotations of the geometrical figure after translation. The relative order of translation and rotation is important (see Prob. 10.3). Here, rotation is performed first with translation following. The possible orientations are defined in Table 10.1-3 and demonstrated with the block letter P in Fig. 10.1-5.

This simple geometrical specification language will suffice to specify any MOS Manhattan integrated circuit layout if the necessary layout levels are defined. The description is based on alphanumeric characters and is easily displayed, edited, or transferred between computer systems.

TABLE 10.1-3
Rotations of geometries

Orientation	Description
1	No rotation
2	Rotate 90° CCW
3	Rotate 180° CCW
4	Rotate 270° CCW
5	Mirror about y-axis
6	Rotate 90° CCW and mirror about y-axis
7	Rotate 180° CCW and mirror about y-axis
8	Rotate 270° CCW and mirror about y-axis

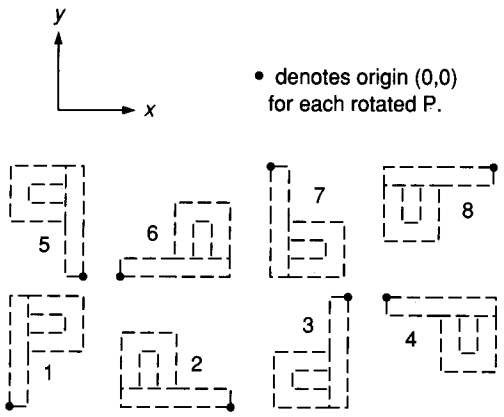
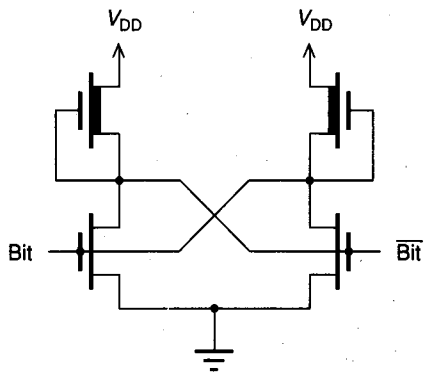


FIGURE 10.1-5
Cell orientations.

```
M 5
L 1
B 11 0 4 5
B 5 3 6 2
B 1 3 4 7
B 1 10 2 3
B 1 13 14 2
L 2
B 4 1 8 6
L 3
B 6 1 4 8
B 8 6 4 6
B 10 12 2 7
L 4
B 0 0 16 2
B 1 6 11 4
B 0 13 16 2
L 5
B 12 0 2 1
B 2 7 2 2
B 9 7 2 2
B 7 14 2 1
E
M 8
C 5 0 0 1
C 5 16 30 3
E
C 8 0 0 1
Q
```

(a)



(b)

FIGURE 10.1-6
Static memory cell definition: (*a*) Geometrical specification file, (*b*) Circuit diagram.

An example of the geometrical specification file for a static memory cell composed of two inverters tied back to back is shown in Fig. 10.1-6 along with the corresponding circuit diagram. A single inverter consisting of an enhancement pulldown transistor and a depletion pullup transistor is defined by macro 5. This inverter is placed twice, once in a rotated and translated position, to create the static memory cell defined as macro 8. Macro 8 is placed once to create the layout plot shown in Fig. 10.1-7.

10.1.2 Layout Styles

In spite of high labor costs, handcrafted layout is still used within the semiconductor industry because of the necessity to minimize the area required by high-volume integrated circuits. Even automated layout methods such as silicon compilation and standard cell synthesis use handcrafted layout to optimize the primitive cells that are combined through automated techniques. Frequently the basic form for the integrated circuit is sketched and optimized on paper prior to entry into

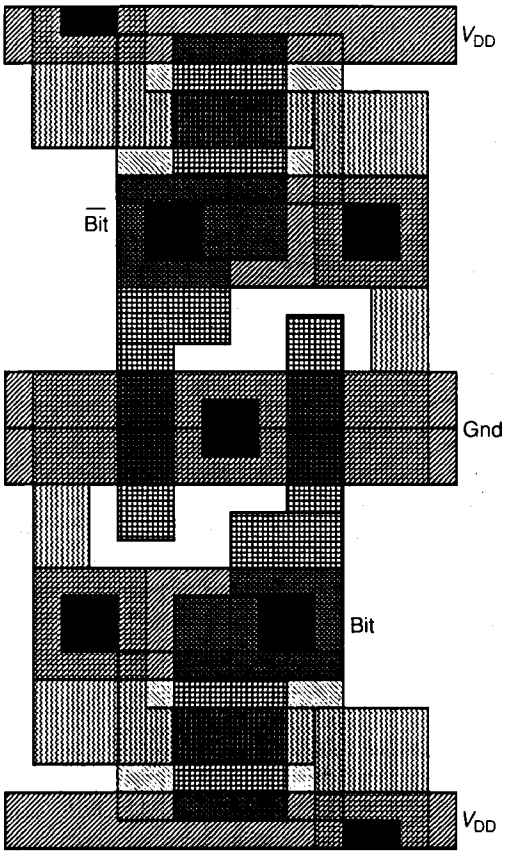


FIGURE 10.1-7
Static memory cell layout.

a computer. The resulting geometrical layouts are digitized, sometimes through use of a symbolic layout language but primarily with the help of an interactive CRT graphics editor.

Handcrafted layouts can be entered directly into a computer in geometrical form through use of an interactive CRT graphics editor. A mouse or joystick is used in conjunction with a cursor to size and position geometrical objects such as boxes on a high-resolution CRT display. A corresponding data file is kept in computer memory to describe the displayed geometries. With an operator's command, this data file may be converted to a geometrical specification language description or can be saved for further use. Several advantages of this graphical editor accrue from bypassing the need to input numerical data to a computer with a text editor or digitizer and from the ease with which geometries can be changed or duplicated.

The graphics editor called Magic,[4] currently popular with universities, uses the painting idiom to create geometrical objects on a color CRT display. The user chooses a color (layout level) from a palette on the screen and paints areas on the screen by specifying two opposite corners of a rectangular field.

The chosen color fills the area. The final result is the same as for other layout methods: a geometrical specification file is created in computer memory, saved, and ultimately transferred to the mask shop.

Graphical editors in both industrial and university environments typically maintain their own unique memory and disk representations of layout geometries. For reasons of efficiency (fast editing response and minimum memory requirements), these representations are often highly optimized binary data structures. In the university environment, the geometrical specification language CIF was defined as a common interchange format among universities and between universities and the MOSIS fabrication service. In industry, EDIF was defined as the interchange format. Most industrial CAD tools provide conversions between their internal format and EDIF. In addition, most industrial CAD tools convert their internal format to a special binary format for submittal to the mask shop. The Berkeley Oct tool set[5] provides conversion from its internal format (Oct) to and from both CIF and EDIF.

In summary, both specification of layout geometries and designer entry of layout geometries are described in this section. Many different geometrical specification languages have been defined and used. A very simple one was defined here for demonstration purposes only. In the university environment, CIF is the predominant interchange format, and EDIF is the interchange standard in industry.

10.2 SYMBOLIC CIRCUIT REPRESENTATION

Descriptions of integrated circuit layouts can take many forms. The geometrical specification language of the previous section provides a primitive textual description of a circuit. Other, more symbolic, forms of representation are often used by designers to specify layouts. A hierarchy of these, including a parameterized layout representation, parameterized module generation, a graphical symbolic representation, and logic equations, is described here.

10.2.1 Parameterized Layout Representation

A symbolic layout language[1] (SLL) allows a textual description of circuit layout in a form that is more easily generated and understood by humans than the geometrical specification language of the previous section. In the past, an SLL was used to represent design layouts that were drawn by hand on graph paper and then digitized. Two main characteristics differentiate an SLL from the geometrical specification language described previously. First, the SLL uses descriptive identifiers for the parameters necessary to specify a geometrical layout. Examples are BOX, POLY, and DX for the geometrical shape, the layer, and the width in the x direction, respectively. This provides a readable description of geometries that specify a layout. Thus, the SLL description is easily entered into a computer using the designer's favorite text editor. Second, symbolic entries are allowed in addition to the numerical data of the geometrical specification language. For example, the x and y position of a geometry might be specified by the variables XPOINT and YPOINT. This allows the final placement of the geometry to depend on the

placement of other cells. At some point in the design process, the SLL must be converted to a geometrical specification language for use by other CAD tools and for transmittal to the mask shop. XPOINT and YPOINT must be assigned numerical values to specify the location of the geometry before this conversion takes place.

In addition to the use of symbolic parameters in an SLL, programming constructs such as loops and conditionals can provide additional capability in the specification of a cell's layout. The use of an SLL to describe layout is much like the use of assembly language to describe the machine language (binary) program for a computer. An assembly language program uses mnemonics for the instructions and symbols for variables to simplify and expedite the process of programming a digital computer. Both forms describe the same end object; the binary representation provides the most concise description, while the assembly language is a preferable working medium for programmers.

An SLL description for the layout of the CMOS inverter of Fig. 7.5-5 is given in Fig. 10.2-1. Note the verbose nature of this description compared to the geometrical specification file of Fig. 10.1-6. The description of Fig. 10.2-2 demonstrates the use of variables to allow the inverter cell of Fig. 7.5-5 to be stretched in either the VERT (vertical) or HORZ (horizontal) directions. Also, a REPEAT statement is included to allow the cell to be repeated NR times. RX and RY are the repeat distances along the x and y axes, respectively. If the variables VERT and HORZ are each set to a value of 0 and NR is set to 4, the inverter cascade of Fig. 10.2-3 is produced. The two variables VERT and HORZ can be used to stretch the inverter cell to match the pitch of adjacent cells by

```
CELLNAME CMOSINV;
BOX NDIF X=3 Y=0 DX=4 DY=4;
BOX NDIF X=3 Y=4 DX=2 DY=4;
BOX NDIF X=3 Y=8 DX=4 DY=4;
BOX PDIF X=3 Y=20 DX=4 DY=4;
BOX PDIF X=3 Y=24 DX=5 DY=4;
BOX PDIF X=3 Y=28 DX=4 DY=4;
BOX POLY X=0 Y=5 DX=7 DY=2;
BOX POLY X=0 Y=7 DX=2 DY=18;
BOX POLY X=0 Y=25 DX=10 DY=2;
BOX POLY X=4 Y=14 DX=8 DY=4;
BOX MET1 X=0 Y=0 DX=12 DY=4;
BOX MET1 X=0 Y=28 DX=12 DY=4;
BOX MET1 X=3 Y=8 DX=4 DY=16;
BOX MET1 X=7 Y=14 DX=1 DY=4;
BOX CONT X=4 Y=1 DX=2 DY=2;
BOX CONT X=4 Y=9 DX=2 DY=2;
BOX CONT X=5 Y=15 DX=2 DY=2;
BOX CONT X=4 Y=29 DX=2 DY=2;
BOX CONT X=4 Y=21 DX=2 DY=2;
BOX NWEL X=0 Y=18 DX=12 DY=16;
END CMOSINV;
```

Figure 10.2-1
Symbolic layout language description of CMOS inverter of Fig. 7.5-5

```
CELLNAME CMOSINV;
BOX NDIF X=3 Y=0 DX=4 DY=4;
BOX NDIF X=3 Y=4 DX=2 DY=4;
BOX NDIF X=3 Y=8 DX=4 DY=4;
BOX PDIF X=3 Y=20 DX=4 DY=4;
BOX PDIF X=3 Y=24 DX=5 DY=4;
BOX PDIF X=3 Y=28 DX=4 DY=4+VERT;
BOX POLY X=0 Y=5 DX=7 DY=2;
BOX POLY X=0 Y=7 DX=2 DY=18;
BOX POLY X=0 Y=25 DX=10 DY=2;
BOX POLY X=4 Y=14 DX=8+HORZ DY=4;
BOX MET1 X=0 Y=0 DX=12+HORZ DY=4;
BOX MET1 X=0 Y=28+VERT DX=12+HORZ DY=4;
BOX MET1 X=3 Y=8 DX=4 DY=16;
BOX MET1 X=7 Y=14 DX=1 DY=4;
BOX CONT X=4 Y=1 DX=2 DY=2;
BOX CONT X=4 Y=9 DX=2 DY=2;
BOX CONT X=5 Y=15 DX=2 DY=2;
BOX CONT X=4 Y=29+VERT DX=2 DY=2;
BOX CONT X=4 Y=21 DX=2 DY=2;
BOX NWEL X=0 Y=18 DX=12 DY=16+VERT;
END CMOSINV;
CELLNAME FOURINV;
REPEAT CMOSINV NR=4 RX=12+HORZ RY=0;
END FOURINV;
```

FIGURE 10.2-2
Parameterized symbolic layout language description for inverter cascade of Fig. 10.2-3

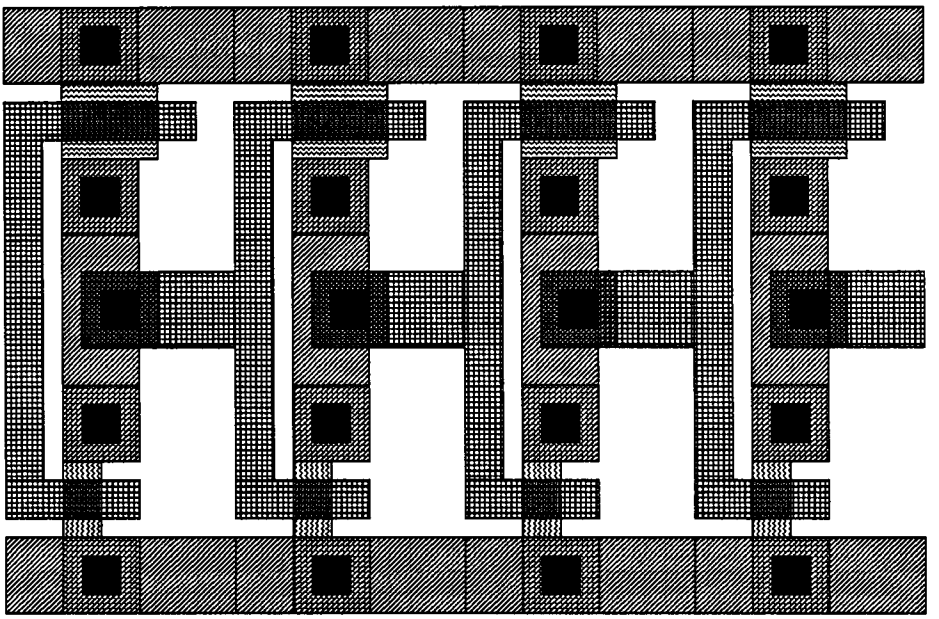


FIGURE 10.2-3
Inverter cascade created from parameterized symbolic layout language description.

specifying positive values for one or both of VERT and HORZ. The use of a programmatic description of layout greatly expands the capabilities of a layout designer in specifying the geometrical structure of a circuit.

10.2.2 Parameterized Module Generation

A recent advance in the area of symbolic layout descriptions is the use of parameterized module generators. A parameterized module generator is a software procedure that can generate many different cell layouts depending on values that are specified when the generator program is executed. Parameterized module generators have been written for RAMs, ROMs, PLAs, multipliers, adders, and data paths, for example. Many of these generators use input parameters to specify the width or number of bits in the generated layout.

As an example, three separate designs might require an 8-bit adder for the first design, a 16-bit adder for the second design, and a 32-bit adder for the third design. Typical design style would use an interactive graphics editor to create each of these adders separately. If a parameterized generator for the adder module could be defined, however, a single module generator could be used to produce an N-bit adder where N is a parameter that can be set to 8, 16, 32, or some other integer value. Then each of the three adders could be created from the same parameterized description. A parameterized module generator is particularly well suited to modern integrated circuit design styles, which commonly utilize regular structures such as rows of cells and arrays of cells.

Parameterized module generators use many of the powerful constructs of high-level programming languages to describe layout structure, position subcells, and fit the overall layout of a larger cell together. Parameterized variables are used with their values bound to a specific value when a module is generated. Conditional statements allow creation of specialized edge cells and programming of memory and PLA contents. For example, a parameterized module generator for an array of cells might include conditional statements such that if both the x and y indices were equal to 0, then an upper-left corner cell would be generated. If the x and y indices were each between the smallest and largest values, a center cell would be generated, and so forth.

The use of high-level programming language techniques also provides a disadvantage for many parameterized module generators. That is, the layout cannot be visualized until the generation program has been compiled and linked to instantiate the layout for a module. These potentially time-consuming steps may hinder the use of interactive layout in designing a module generator for a new cell. To circumvent this problem, there is ongoing research on ways to provide interactive graphical feedback as the geometrical structure of a cell is defined.[6]

With such a tool, a silicon layout specialist can create the parameterized modules that are required in a design. Then a circuit or logic designer can use these blocks by specifying parameters appropriate to the design task. Recently, parameterized module generators were used to specify the layout of a commercial RISC processor (see Sec. 10.11). An interesting, but unsolved problem, is to prove that the output of a parameterized module generator is correct over the valid range of parameters for the module generator.

10.2.3 Graphical Symbolic Layout

The parameterized layout representation described previously provides little insight into the geometrical relationships between circuit elements. This important insight can be provided by another symbolic form for integrated circuit description, called graphical symbolic layout. An early form of graphical symbolic layout is called Sticks.[7] Sticks and related symbolic methods provide an abbreviated, graphical description that combines circuit connectivity with layout topology information. In the Sticks symbology, circuit connections are shown with colored (or weighted) lines representing layout levels, while transistors are formed by the intersection of the lines representing polysilicon and diffusion. The entire layout diagram is composed of simple line symbols that show both connectivity and topology but not actual or relative size for geometrical constructions.

The combination of connectivity and topological information is important in the generation of integrated circuit layouts, as is shown with the aid of the circuit diagram for the quasi-static memory cell of Fig. 10.2-4*a*. This circuit diagram shows a forward path from the first inverter to the second inverter and a clocked feedback path from the second inverter to the input of the first inverter. The circuit diagram does not indicate topological requirements to realize this path.

The geometrical layout of the memory cell of Fig. 10.2-4*a* requires decisions on changes of layout levels to prevent unwanted transistors and connections. The Sticks diagram of Fig. 10.2-4*b* retains all the circuit connectivity information

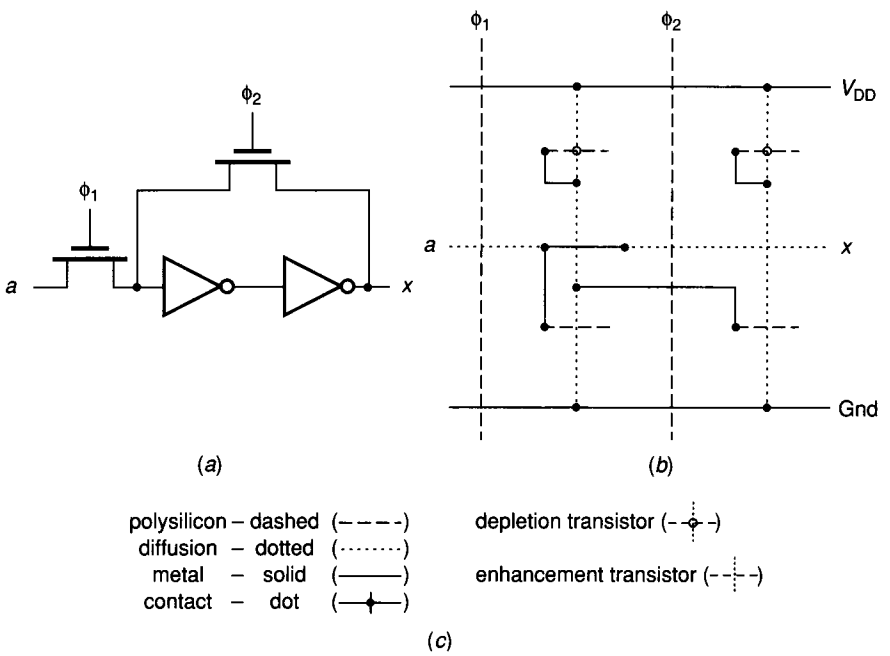


FIGURE 10.2-4
Quasi-static memory cell: (*a*) Logic diagram, (*b*) Symbolic diagram for NMOS layout, (*c*) Layer legend.

for the memory cell and also symbolically specifies the topology of the final integrated circuit layout. In particular, it shows that the feedforward path must be changed from the diffusion layer to the metal layer to cross the polysilicon ϕ_2 clock line without creating an unwanted transistor. Also, the Sticks diagram shows that the feedback transistor is conveniently formed by allowing the polysilicon ϕ_2 line to cross the diffusion feedback path. The diagram shows that power and ground are provided in metal and that the input and output signals are both in the diffusion level. The utility of Sticks and other graphical symbolic layout methods is derived from the simple abstracted notation for layout topology and circuit description.

Once a graphical symbolic layout for a circuit is generated, it is often simple for a designer to convert to a full layout form. The layout task has been simplified to the process of fattening connection lines and compacting the layout, especially if required transistor length-to-width ratios have been noted on the graphical symbolic layout. In fact, this process is simple enough to be automated.[8] If the graphical symbolic layout description has been entered into a computer, perhaps through an interactive graphics terminal, a symbolic compiler program can convert the symbolic layout to a full layout by expanding the line symbols according to a technology specification and then compacting the resulting layout.

As with most automated layout aids, a symbolic compiler usually trades reduced designer efforts for increased silicon area. An increase in the area for a layout generated with a program is not uncommon when compared to a handcrafted layout. As a result, high-volume integrated circuits such as microprocessors and memory continue to utilize handcrafted layout of replicated cells as a major design component. This does not, however, minimize the value of the symbolic representation to the designer. Capturing layout topology in symbolic form early in the layout design prevents later problems such as isolation of a circuit from direct metal connection to power buses.

10.2.4 Logic Equation Symbology

If the function of a digital integrated circuit can be captured by a set of Boolean logic equations, these equations suffice to generate an integrated circuit layout. Thus, logic equations represent a fourth symbolic means to describe a combinational logic circuit. One frequently used means to convert logic equations into layout topology is with a PLA generator, as described in Chapter 9. Two other methods for generating geometrical layouts from logic equations are discussed next: the Weinberger array[9] and SLAP[10] (a methodology for silicon layout).

A Weinberger array uses a regular structure of NOR gates to implement combinational logic in an integrated circuit form. This array structure was introduced in Chapter 9. Figure 10.2-5 shows a Weinberger array used to implement the full adder carry function described by

$$K = AB + AC + BC \tag{10.2-1}$$

Since the final structure is regular, it is not difficult to construct a computer program to generate the array layout using logic equations as program input. By

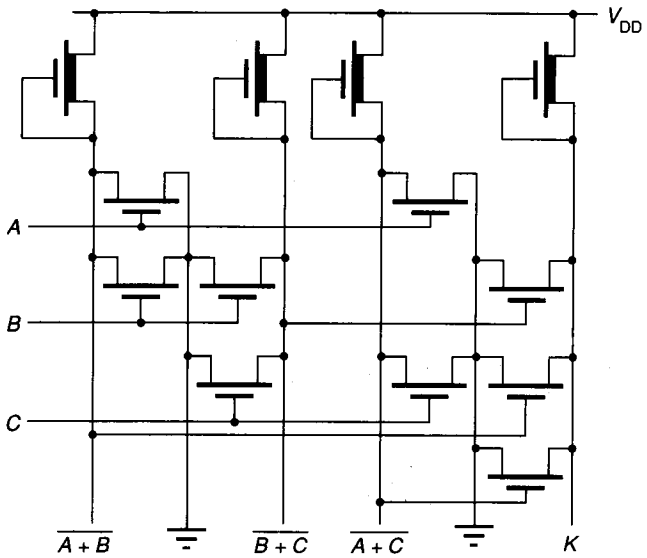


FIGURE 10.2-5
Weinberger array for full-adder carry.

use of DeMorgan's theorem, any combinational logic function can be realized using only NOR gates. In fact, the Weinberger array requires at most a series path of three NOR gates between an input and an output to realize a combinational logic function. Remember that a single-input NOR gate is an inverter. Thus, a first NOR gate may be required to provide the complement of an input while the final two levels use the NOR-NOR logic form to realize the logic function in product-of-sums form.

The use of NOR gates for a Weinberger array allows a constant size for the pullup devices even though the number of inputs and their corresponding pulldown devices may differ for each gate. Careful design allows adjacent gates to share a single ground path, as shown in the layout of Fig. 10.2-6. This array structure can be easily expanded by adding input variables at the bottom and NOR gates to the right without changing the existing structure.

A comparison of the Weinberger array with the PLA yields an interesting result. Even though the logic of a PLA is realized entirely with NOR gates, the AND-OR logic form corresponding to a sum-of-products description is normally used. The AND-OR logic form can be realized with NOR gates only by inverting both the inputs and outputs. This requires a series string of four or five NOR gates between the PLA inputs and outputs, thus causing more delay for a PLA implementation of logic than for a Weinberger array implementation which requires only three levels of logic.

In contrast to the PLA and the Weinberger array, both with predefined array structures, a third method called SLAP has been proposed to compile logic equations into layout form. SLAP first converts logic equations into a directed graph with a graph level for each level of the logic equations. If double-rail inputs

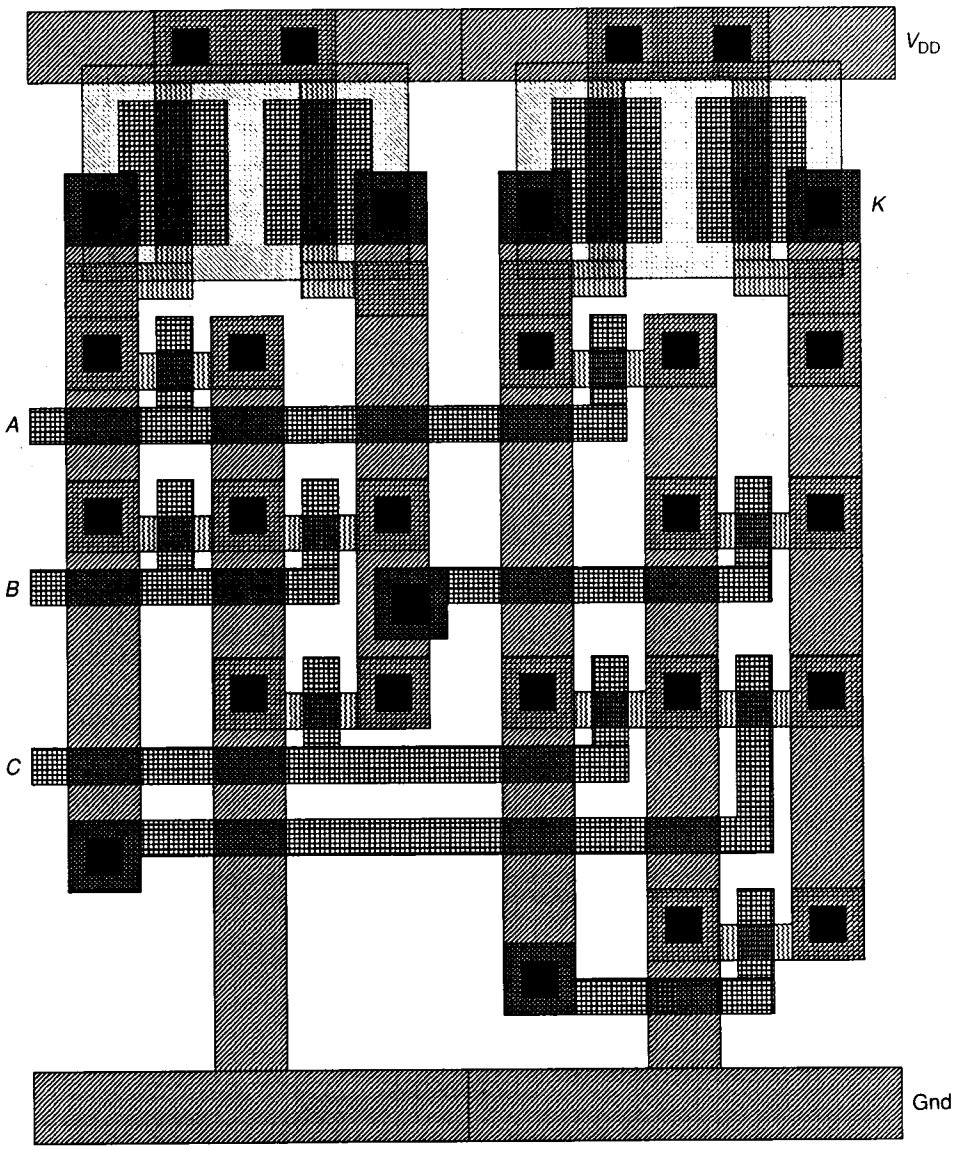


FIGURE 10.2-6
Layout for Weinberger array.

are available, at least two levels of gates are required to implement a general logic function. The SLAP methodology, however, allows realization of intermediate outputs that may then be used as inputs for other logic functions. A graph with an arbitrary number of levels may be required, depending on the particular representation for the logic. Figure 10.2-7 shows the directed graph for the logic functions of the following equations.

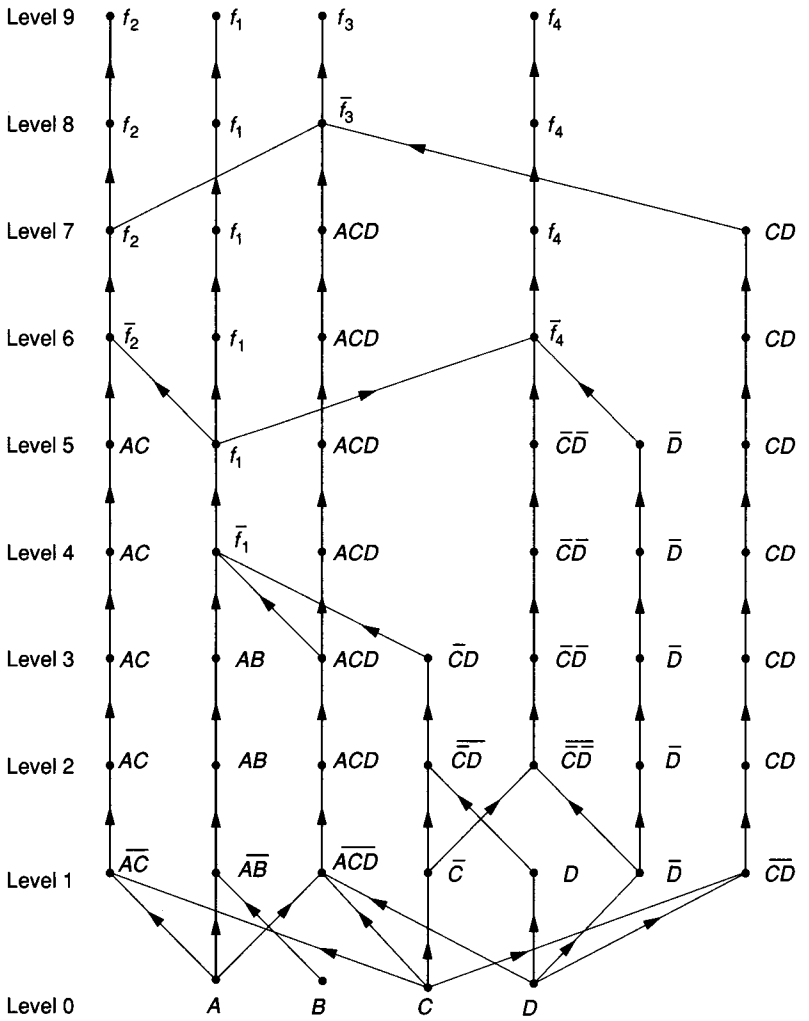


FIGURE 10.2-7
Directed graph for Eqs. 10.2-2 through 10.2-5.

$$f_1 = AB + \overline{C}D + ACD \tag{10.2-2}$$

$$f_2 = AC + f1 \tag{10.2-3}$$

$$f_3 = ACD + CD + f_2 \tag{10.2-4}$$

$$f_4 = \overline{C}\,\overline{D} + \overline{D} + f_1 \tag{10.2-5}$$

This directed graph is formed by placing logic gates with external inputs at the first level, secondary logic gates at the next level, and so on. Heuristics are then used to improve the organization by reducing the number of required levels, if possible, and to reduce the resulting layout area required. The layout density achieved with this method is about the same as that accomplished with gate array

structures. An important characteristic of the SLAP methodology is that general logic structures can be compiled directly into a geometrical layout, whereas the PLA format forces a two-level logic realization.

In this section, four methods of generating layout from symbolic representations were introduced. Of the first two, parameterized layout representation and parameterized module generation, the second is growing in popularity for layout of today's designs. Graphical symbolic layout also enjoys success as a technique for layout of random logic. Synthesis of layout directly from the fourth symbolic form, logic equations, is fast becoming a widely used technique for generating integrated circuit layout.

10.3 COMPUTER CHECK PLOTS

Generation of a layout plot from a geometrical specification file for an integrated circuit is often desirable. In the past, large-scale plots, some almost big enough to cover one end of a basketball court, were generated so that visual checking of circuit layout could be performed. Most of these visual checks can now be performed directly from a computer-based geometrical specification file without manual intervention. A computer program can verify fixed rules for the millions of geometrical figures used to describe VLSI circuits without tiring and without error—a task that is essentially impossible for humans. However, human capability to critique overall structure or to detect inconsistencies in an otherwise regular design is difficult to duplicate with computer-based checks. As a result, hardcopy plots of integrated circuit designs are still used for finding errors, for promotional literature, and for many other purposes. Such plots are called *computer check plots*.

Computer check plots for integrated circuit designs are created in both soft- and hardcopy form on CRTs, printers, and plotters using color or black on white representations for the layout artifacts. Check plot devices range from monochrome CRTs, with only 24 × 80 character resolution for the entire display, to laser printers with 300 dots per inch or higher resolution. To compare the maximum usable display capability over this range of resolution, an example using a static memory cell is examined next.

The static memory cell of Fig. 10.1-7 has dimensions of 16 $\lambda \times 30\ \lambda$ for an area of 480 λ^2. A monochrome alphanumeric CRT using character graphics with 24 lines by 80 columns can display an area of 1920 λ^2, although the effective area is somewhat less because of the 1:3 aspect ratio of the CRT display resolution. All details of the static memory cell are visible in the CRT display, as shown in the hardcopy plot of Fig. 10.3-1, but the cell's relation to other cells is lost. As a second example, a dot matrix drawing normally requires a resolution of at least 5 dots per λ to define the smallest details of a circuit. For a printer with a resolution of 100 dots per inch, the static memory cell requires a plot that is about 0.8 × 1.5 in. to show the details of the circuit. Figure 10.3-2 provides a plot of this size for the memory cell of Fig. 10.1-7. If the memory cell were part of a 1K-bit memory (32 cells × 32 cells), a high-resolution plot of the entire memory array would require 25.6 × 48 in. Of course, the general form of the memory area could be discerned with a much smaller plot. Figure 10.3-3 shows a plot at one-tenth this scale (2.56 × 4.8 in.) for the 32-by-32 cell array.

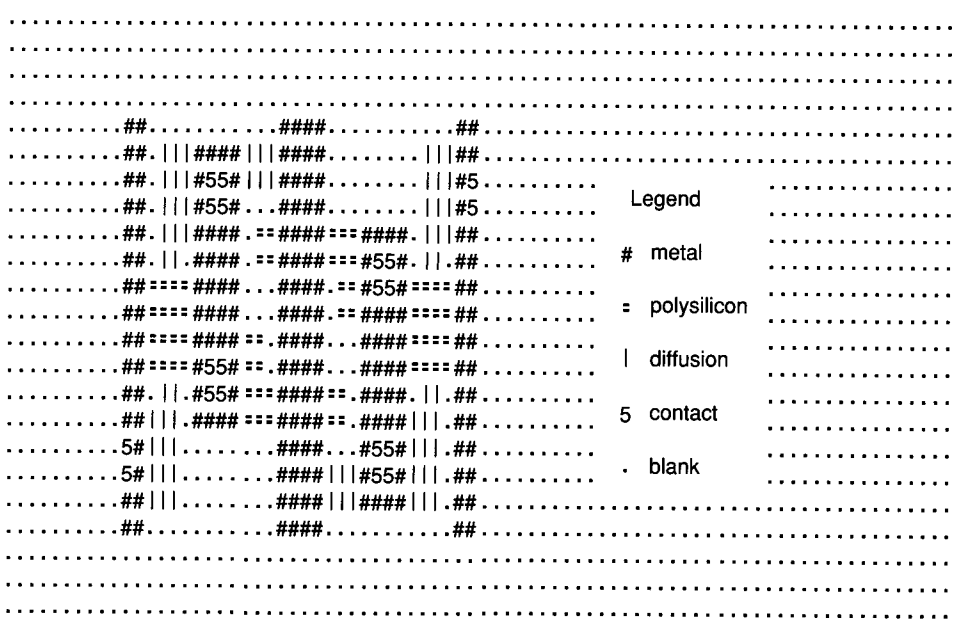


FIGURE 10.3-1
Hardcopy plot of SRAM cell as displayed on 24 line by 80 character CRT (λ = one character width).

A typical graphics CRT display with a 19 in. diagonal screen (15 in. horizontal $\times$ 11 in. vertical) might have a resolution of 760 by 480 dots. This is roughly 50 dots per display inch. Based on the analysis above, the details of a 152 λ by 96 λ circuit could be displayed in its entirety on the screen. This would correspond to about a five-by-six array of the memory cells described above. Figure 10.3-4 shows a hardcopy plot of the memory cells that could be seen on the CRT display. Of course, an entire chip can be displayed if the layout is scaled so that the finer details of the chip are lost. Figure 10.3-5 shows the entire layout for a 220 $\times$ 230 mil image-processing chip composed of sixteen, 12-bit serial multipliers with associated circuitry and input/output pads.

Color displays and plots are always a higher-cost feature than black and white; where color is available, each integrated circuit layer is represented using

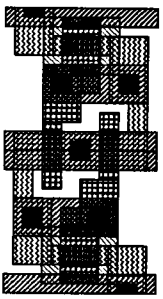


FIGURE 10.3-2
Minimum size plot for Fig. 10.1-7 with 100 dots/inch resolution.

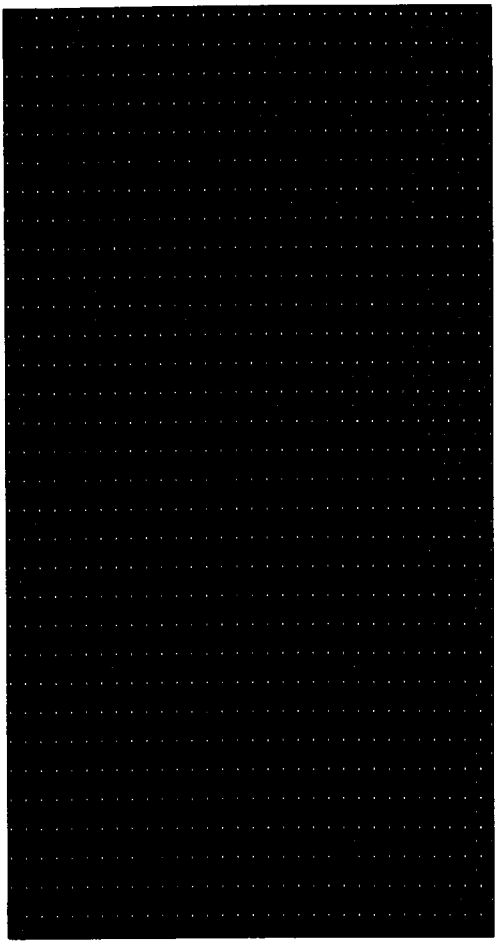


FIGURE 10.3-3
Plot of 32 × 32 memory cell array.

a different color. Aside from their aesthetic appeal, color renditions of circuits show higher information content per unit area, allowing display of larger circuits in a given area. For a color display, only 2 to 3 dots per λ of resolution are necessary to delineate circuit details. Additionally, individual color levels can be used to show labels, flag geometrical design rule violations, or highlight specific features of a circuit. Most modern graphics workstations provide color displays.

When black on white plots are generated, two primary methods are used to distinguish individual layers. Line drawings, with each layer represented by a different style of line (solid, dotted, dashed, dot-dash, etc.) are producible on almost any printer with dot graphics capability (see Fig. 10.3-6). Filled drawings with different layers shown by characteristic area fill pattern (fine dots, heavy dots, diagonal lines, vertical lines, etc.) are popular, even at the expense of increased computer time to generate the plots, greater wear on the printer mechanism, and longer time to print the plots. Laser printers provide good resolution

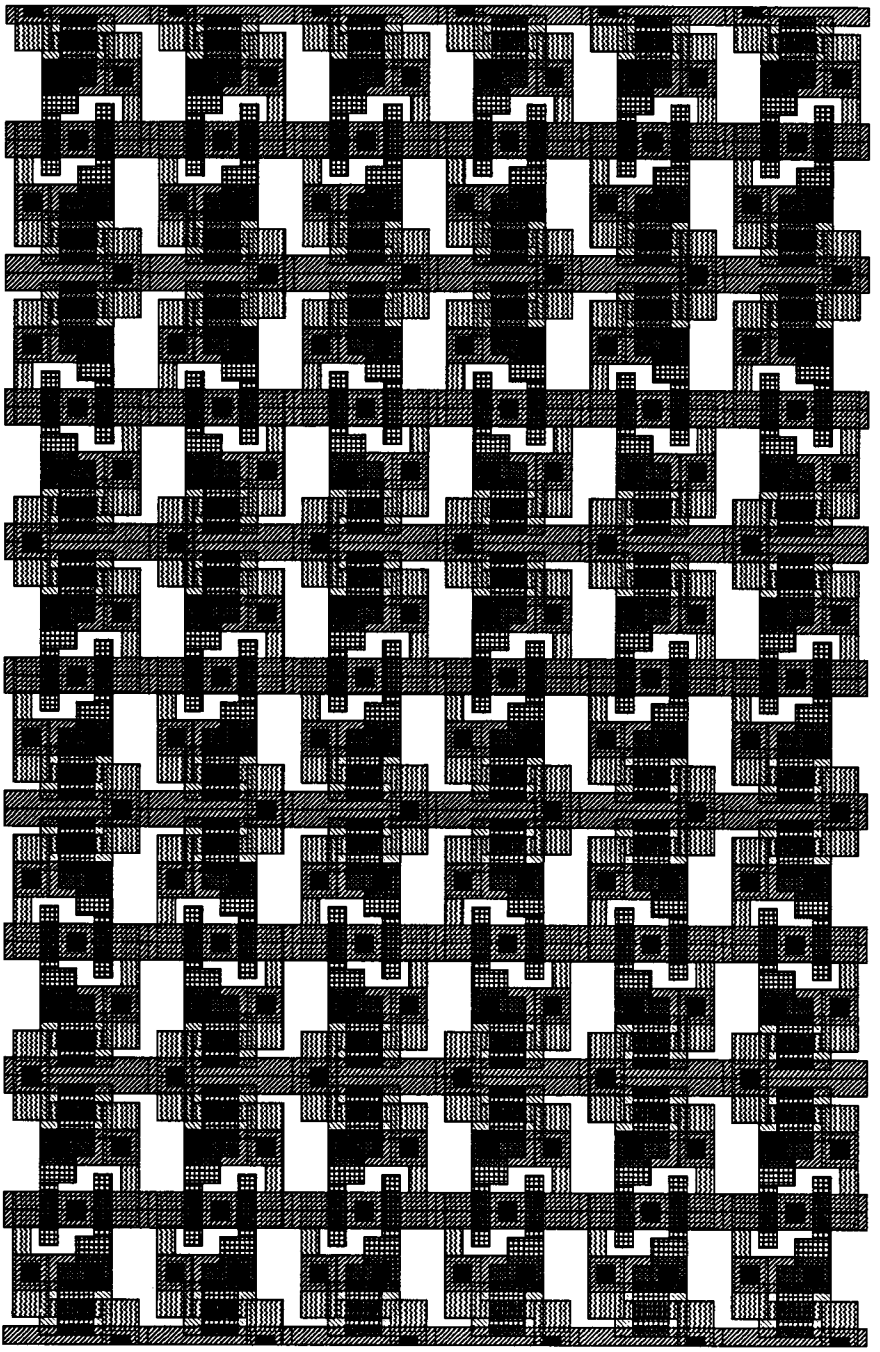


FIGURE 10.3-4
Hardcopy plot of memory cells visible on typical graphics CRT display.

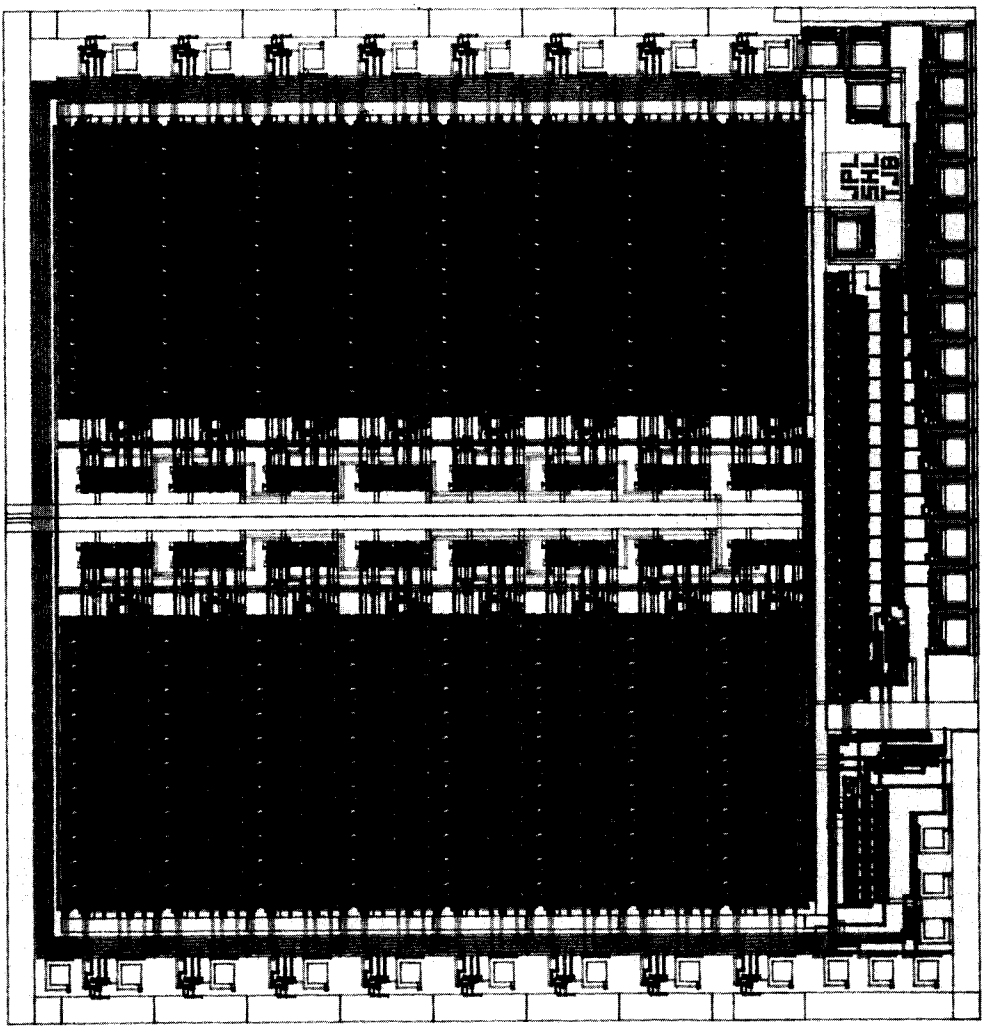


FIGURE 10.3-5
Image-processing chip (220 mil × 230 mil).

(300 dots per inch) and are frequently used for area fill check plots. A primary advantage of the filled drawing of Fig. 10.1-7, compared with the line drawing of Fig. 10.3-6, is that the concept of area for integrated circuit layers is quickly conveyed to the viewer by the filled drawing. This concept is important to the designer since the fabrication process operates on contiguous areas rather than the individual boxes used to describe them.

In this section, a short summary of integrated circuit display media and their corresponding resolution requirements was presented. It is important to have high-resolution display and hardcopy capability for integrated circuit layout design.

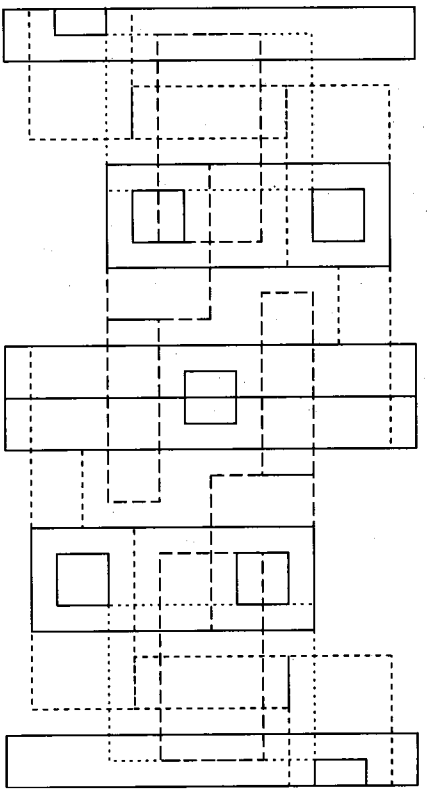


FIGURE 10.3-6
Line check plot of layout of Fig. 10.1-7.

10.4 DESIGN RULE CHECKS

Integrated circuits are created from several layers whose geometrical structures are defined by photolithographic masks. At any given time, a minimum resolution exists for the structures that can be fabricated on silicon because of lithographic and processing constraints. Any attempt to define structures that require higher resolution or accidental specification of a higher resolution through carelessness may lead to nonfunctional circuits. Also, violation of certain geometrical relationships among layers may cause failures because of processing constraints. For each process, a set of guidelines called *design rules* is specified to encapsulate geometrical fabrication constraints. The design rules for the CMOS process described in Table 2B of Appendix 2 are used as the basis for the following discussion. However, most of the rules are determined by general lithographic and processing constraints so that similar rules apply to other processes as well.

10.4.1 Geometrical Design Rules

A conceptual explanation of geometrical design rules is provided in this section. Design rules were introduced in Sec. 2.3 of this text. Geometrical design rules for a single integrated circuit layer are simple; they involve only spacings and widths. Figure 10.4-1 demonstrates a 2 λ spacing between polysilicon conductors

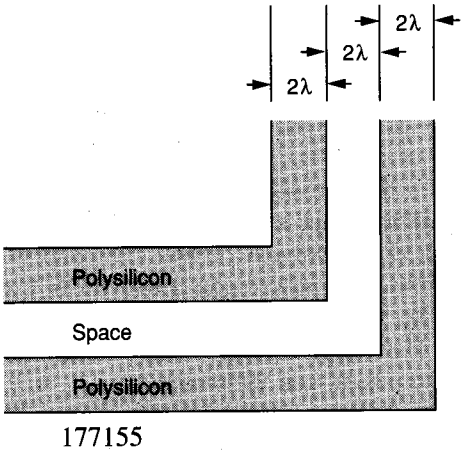


FIGURE 10.4-1
Minimum width polysilicon conductors.

that are each 2λ wide. It is worth noting that if a mask layer is complemented, all widths become spacings. This is shown in Fig. 10.4-2, where the complemented polysilicon conductor widths from Fig. 10.4-1 appear as spacings. Therefore, if width is considered in terms of the complement of the layer definition, all single-layer rules can be treated as simple spacing rules. This means that the same computer algorithm can be used to check for both width and spacing errors.

An interesting conceptual understanding of design rules was provided by Lyon.[11] His explanation is based on the scalable parameter λ, which is said to describe the minimum resolution of the fabrication process. In practice, fabrication processes are usually characterized by their minimum transistor length. The parameter λ is normally specified as half the minimum transistor length. Thus, a 2 μ process has a minimum gate length (and width) of 2 μ, and λ would be set to 1 μ. Thus, λ is not directly a measure of process resolution, but rather is proportional to the minimum device length. With this in mind, the following two meta rules (a meta rule is a rule about rules) were proposed by Lyon to generalize geometrical design rules in terms of λ.

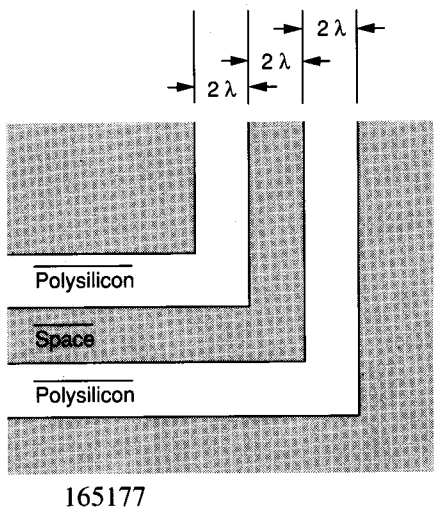


FIGURE 10.4-2
Complemented layout, where spacings become widths.

1. A 1 λ error should not be fatal, although the intended performance of the integrated circuit may be degraded.
2. A 2 λ error may be fatal and almost certainly will degrade the performance of the integrated circuit.

Consider the minimum width of 2 λ for the polysilicon conductor shown in Fig. 10.4-3*a*. If the width of the actual polysilicon that is fabricated on the chip is 1 λ less than this minimum, as in Fig. 10.4-3*b*, the polysilicon will still conduct, although its resistance will double. If the fabricated polysilicon conductor is 1 λ wider than the minimum, as in Fig. 10.4-3*c*, the resistance is lowered, but the polysilicon still functions as a conductor. Thus, a change in width of 1 λ does not cause an obviously fatal problem for the polysilicon interconnection.

Now consider a 2 λ deviation from the design width of 2 λ. If a minimum width polysilicon conductor is narrowed by 2 λ, as in Fig. 10.4-4*a*, there is no conductor left—certainly a fatal error unless the connection was redundant. If the width is increased by 2 λ as in Fig. 10.4-4*b* and the minimum polysilicon spacing is 2 λ, there is a chance that the polysilicon conductor will contact an adjacent polysilicon conductor, causing a short circuit—also a fatal error.

Other design rules involve more than one level and are harder to remember and to verify. As an example of a two-level rule, consider that a transistor is created by the area common to polysilicon and diffusion. This transistor area must satisfy the 2 λ minimum length rule, so the smallest transistor size is 2 λ by 2 λ. The diffusion areas for the source and drain of a transistor also must satisfy a 2 λ minimum length. This rule is sometimes confusing from a layout viewpoint since the source, the drain, and the transistor gate area appear as one contiguous diffusion area. Thus, a source area 1 λ long combined with a transistor area 2 λ long and a drain area 2 λ long, shown in Fig. 10.4-5*a*, appears as a diffusion area 5 λ long and does not seem to violate the 2 λ diffusion length rule. However, Fig. 10.4-5*b* shows that a source only 1 λ long could disappear as a result of a 1 λ alignment error between polysilicon and diffusion—thus a 2 λ rule must be specified for the transistor source/diffusion dimensions. Typical design rule sets for several processes, including NMOS and CMOS, are provided in Appendix 2.

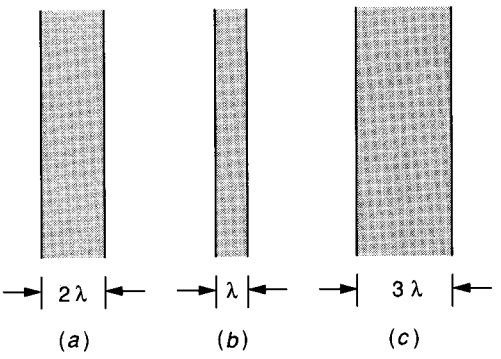


FIGURE 10.4-3
DRC degradation meta rule.