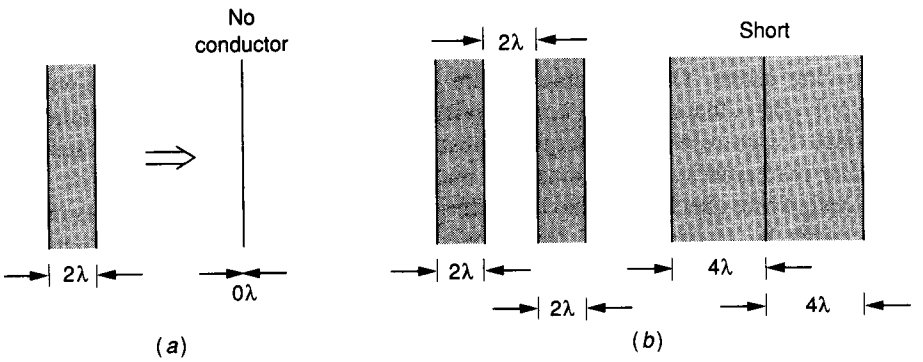

FIGURE 10.4-4
DRC fatality meta rule.

10.4.2 Computer Design Rule Checks

If a designer creates or changes a geometrical specification file manually, a *design rule check* (DRC) is required. Because of the large number of geometries and the wide variation in number and style of geometrical design rules in today's circuits, computer-based DRCs are necessary. Two different styles of DRC programs are in wide use. These can be categorized as polygonal checks and raster scan checks. Both styles will be described briefly.

Polygonal design rule checks are widely used within the semiconductor industry. The geometrical specification file is expanded to produce polygons defining all connected areas for the layer(s) of interest. Note that the layer of interest may be a composite area such as active transistor area or perhaps depletion transistor area. Or it may be a difference area such as the ion implantation overhang created by subtracting the depletion transistor area from the ion implan-

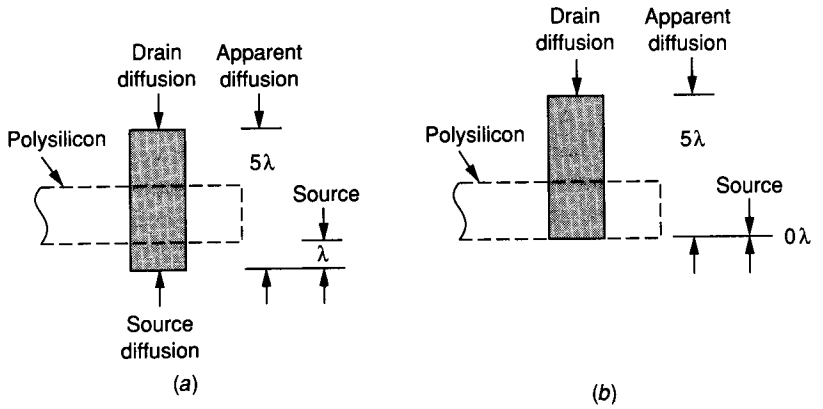

FIGURE 10.4-5
Transistor source width.

tation area. These special areas can be defined by logical operations on primitive layers. Once the polygonal definitions are formed, they can be analyzed for width and spacing errors. One valuable feature of encircling a connected area with a single polygon is that electrical connectivity information is immediately available. Polygonal design rule checks require substantial computing resources because of the many mathematical operations that must be performed during the check.

Design rule checks can also be performed in a relatively simple way as raster scan checks by passing small filters over a rasterized image of the integrated circuit. To allow this, an entire geometrical specification file is instantiated (expanded into the geometries and layers that represent the layout) within a two-dimensional array where the dimensions represent the x and y coordinates of a point and the contents are binary variables to indicate the presence or absence of each layout level. The resolution of the x and y coordinates limits the precision of the design rule checks. Filters such as a 4×4 array,[12] a "plus" symbol, or a circled "plus" symbol[13] have been used to scan the instantiated layout to check for design rule violations. These methods are conceptually simple and computationally clean, but lack the accuracy and connectivity information of the polygonal methods.

10.4.3 Design Rule Checker Output

To demonstrate the results from a raster scan DRC program, several errors were placed in a geometrical specification file. The layout for this file is shown in Fig. 10.4-6. The resulting output from the DRC program is shown in Fig. 10.4-7. The DRC program outputs a heading that gives the name of the file, the date and time, the bounding box coordinates for the checked area, and the macro number. Below the heading, a list of all vertical and horizontal errors is provided. This particular sample contains three vertical and four horizontal errors. Each violation is shown by a one-line entry containing the identification of the violated rule, the x and y coordinates of the violation, the violation or error distance, and the length over which the violation occurred. The resolution of the layout of Fig. 10.4-6 and the DRC results of Fig. 10.4-7 is 0.5 λ.

Definitions of the seven rule violations from Fig. 10.4-7 are given in Table 10.4-1. In each case these errors involve a spacing violation. For example, Rule 6.2 is a metal spacing error. A glance at the upper left corner of Fig. 10.4-6 shows a T formed by a long horizontal metal section and a short vertical metal section separated from the horizontal metal (top of the T) by about 1 λ. From Rule 6.2, the spacing must be at least 3 λ unless the two metal sections should be joined, in which case the spacing would be zero. As an exercise, the reader should find the location of each of the errors listed in Fig. 10.4-7.

Once the cause of an error is determined, corrective action must be initiated. Since the DRC output gives the exact x and y coordinates of the violation, it is usually relatively simple to use an interactive graphics CRT to display the error. Actually correcting the error may not be so simple. If the layout is loosely packed, correction in place by adjusting a single geometrical figure can possibly be done. For some layouts, however, an error will occur in a space-critical area,

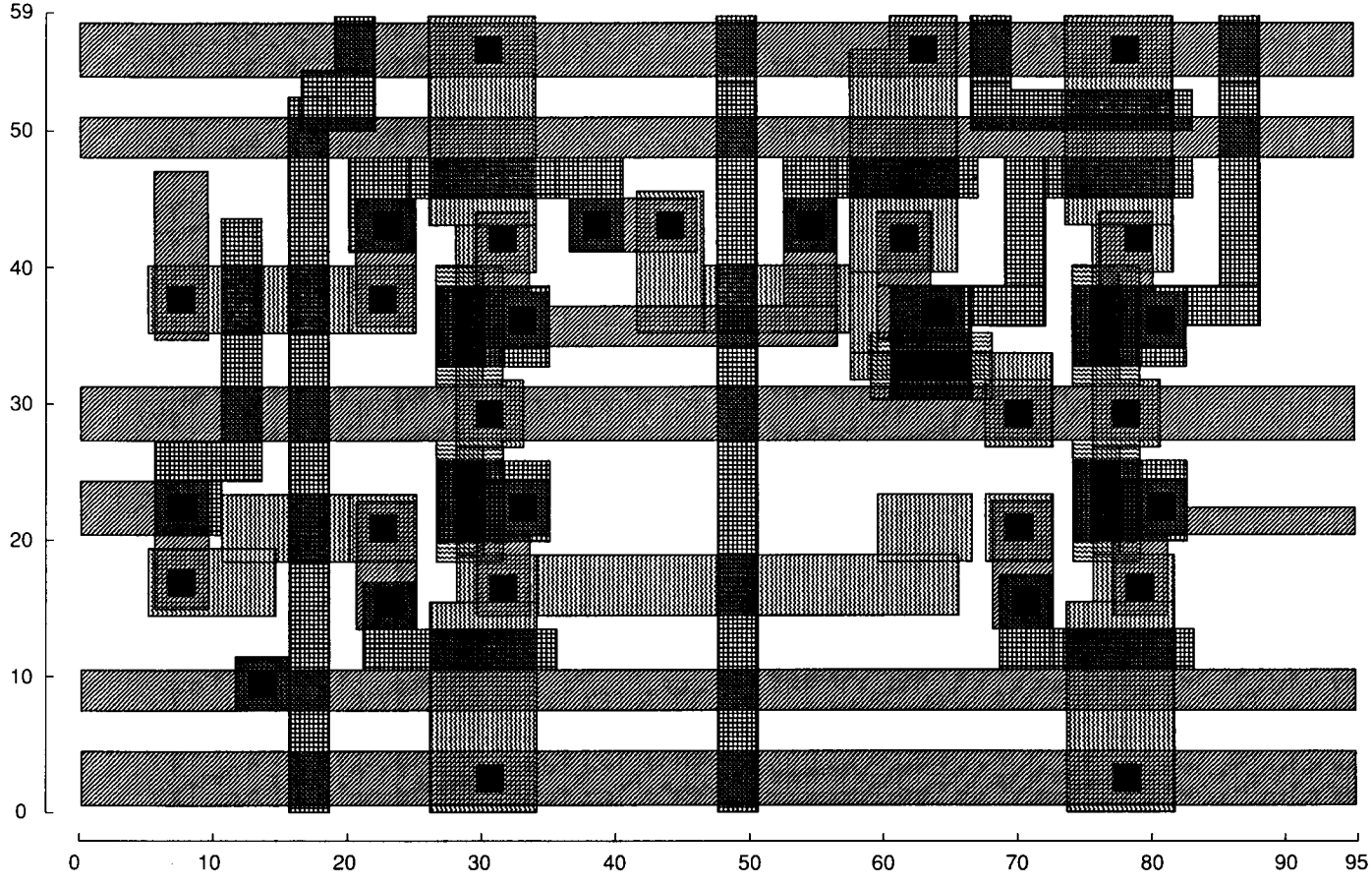

FIGURE 10.4-6
Sample layout for DRC.

LDRC version 3.115

Design rule check of file: BGA.TAM
Date 9-MAR-89 Time 21:08:05
X min= 0.0 X max= 95.0
Y min= 0.0 Y max= 59.0

Macro name is BGMLT
Macro number is 99

Vertical errors

Rule	X loc	Y loc	Error	X len
6.2	5.5	47.5	1.0	4.0
5.3	22.0	17.0	0.5	2.0
6.1	82.5	20.5	1.0	12.5

Vertical error count: 3

Horizontal errors

Rule	X loc	Y loc	Error	X len
4.3	10.5	20.5	0.0	3.0
4.2	18.5	41.5	0.5	7.0
1.2	66.5	18.5	1.0	5.0
5.6	80.0	41.5	1.0	2.0

Horizontal error count: 4

Total number of Design rule violations: 7

Design-Rule Checker Execution:

CPU Time 0: 0:26.06
Page Faults 354

FIGURE 10.4-7
DRC output for Fig. 10.4-6.

requiring changes of a large number of geometries. For this reason, it is crucial to generate a correct layout through automatic means or, in the case of a handcrafted design, to check the layout frequently for geometrical design-rule errors as it is generated. With care, errors are caught early before correction causes difficult problems.

TABLE 10.4-1
Design rule error definitions

Rule	Length	Definition
1.2	3 λ	Diffusion spacing
4.2	2 λ	Polysilicon spacing
4.3	λ	Polysilicon-to-diffusion spacing
5.3	λ	Polysilicon larger than contact
5.6	λ	Metal larger than contact
6.1	3 λ	Metal width
6.2	3 λ	Metal spacing

The DRC program used here was run in the batch mode on a computer after the layout was complete. Many CAD systems allow DRCs as geometries are entered through an interactive graphics CRT using an incremental DRC program. Either the designer is prevented from placing geometries that would violate design rules, or a pending violation is flagged immediately by an error message. This minimizes the need for major changes after the layout is almost complete.

DRCs are one of the more time-consuming, yet important, design verification steps. Both polygonal and raster scan DRCs are possible. A good DRC program provides output that accurately identifies the type and location of each error. A good interface between the DRC program and an interactive graphics editor is important for displaying and correcting DRC errors.

10.5 CIRCUIT EXTRACTION

After the design and layout process is complete, MOS circuits are characterized by a machine-readable specification prior to the mask-making step. This specification is usually a geometrical specification file as described earlier. This file contains all the information about the geometries, levels, and placements for the circuit to be fabricated. Because geometrical specification files contain large quantities of detailed information about the integrated circuit, it is difficult for a designer to determine whether this information accurately describes the circuit that was intended. Fortunately, computerized methods exist to extract the circuit information from the geometrical specification file. The process of extracting the circuit information from the geometrical description is called *circuit extraction.*

A circuit extraction program expands the geometrical specification file of the integrated circuit into a layer-by-layer description of the geometries and their placements. This description is then scanned to locate all transistors and interconnections for the circuit. A result of the circuit extraction program is a net list. A *net list* is a set of statements that specifies the elements of a circuit (for example, transistors or gates) and their interconnection. Individual transistors are described along with the nodes to which they connect. This information allows creation of a circuit diagram based on the actual geometrical specification file. Most importantly, the extracted circuit can be compared with the original circuit specified by the designer so that differences are annotated. A difference usually indicates an error that must be corrected. This comparison is called an LVS (layout versus schematic) design verification step.

In addition to providing the details of circuit interconnections, circuit extraction is useful for calculating layout areas and perimeters for each integrated circuit layer at each node of the circuit. These layout areas and perimeters can be used to accurately calculate the parasitic capacitances and resistances that load the active devices. Prior to the layout and extraction step, most circuit parasitics can only be estimated by the designer. With accurate capacitance and resistance values from circuit extraction, a design can be accurately simulated to ensure correct operation. Thus, circuit extraction is an essential design verification tool for accurate characterization of modern integrated circuits.

10.5.1 A Simple Circuit Extraction Algorithm

One simple method of circuit extraction consists of two main steps. First, the geometrical specification file is instantiated as a set of coordinates and levels within a computer memory. This is essentially the same operation that was required for the raster scan DRC described in the previous section. This method requires a large computer memory to store the integrated circuit levels at a resolution matching the smallest features of the integrated circuit. For example, a 5 mm by 5 mm die using a process with a λ of 1 μ could require over 25 million individual memory locations to store the instantiated layout, where each memory location corresponds to a 1 λ by 1 λ cell of layout. For a CMOS process, roughly 14 possible layout levels must be remembered for each location. This results in a storage requirement of more than 350 megabits. One useful approach to minimize the memory requirements is to instantiate the design file in overlapping strips. All required circuit information is extracted from each strip before the next strip is instantiated.

The second main step in circuit extraction is the extraction of transistor and connectivity information from the instantiated layout. This is a straightforward, but time-consuming task. The instantiated layout is scanned using a format typical of that used to display television images. The scanning order described here is left-to-right and top-to-bottom, with all integrated circuit levels scanned in parallel. Information on the extent of each level is obtained, and relations between levels that form transistors and contact cuts are derived.

A simple algorithm to determine connectivity at each level can be described as follows. This algorithm requires the program to look at the current cell, the cell to the left, and the cell above. Figure 10.5-1 shows conditions of interest where a "–" indicates no level present and an "m" indicates the presence of a level (e.g., metal). If the current cell does not contain a level, action is not required. This condition is shown in Fig. 10.5-1*a* by a template (upper part) and a 5 λ by 5 λ layout sample (lower part). If the current cell contains a level, four possible cases are of interest; these are shown in Figs. 10.5-1*b* through 10.5-1*e*.

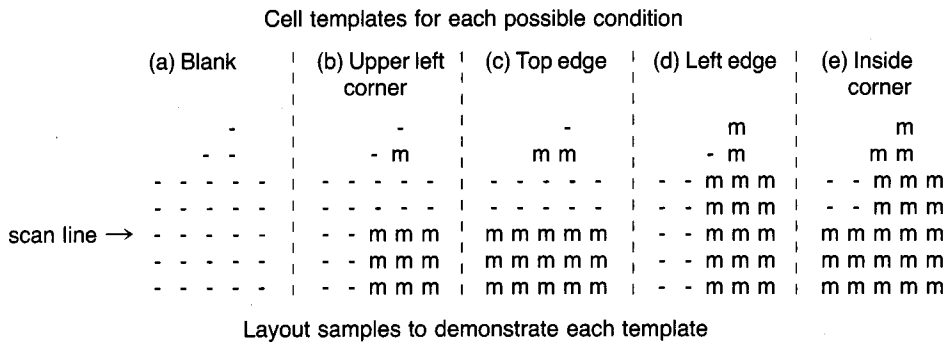

FIGURE 10.5-1
Connectivity extraction.

If the current cell contains a level, say metal, then four cases must be examined. First, if neither the cell to the left nor the cell above contains metal, then an *upper left corner* has been encountered as in Fig. 10.5-1*b*, and a new node number must be assigned to this location. As a second case, if the cell to the left contains metal but the cell above does not, as in Fig. 10.5-1*c*, then the extractor is moving along a *top edge*, and the current node is assigned the same node number as the cell to the left. As a third case, if the cell above contains metal but the cell to the left does not, as is shown in Fig. 10.5-1*d*, a *left edge* has been found, and this node is assigned the same node number as the cell above. As a final case, if both the cell above and the cell to the left contain metal, either an *internal point* or an *inside corner* has been found. If the node numbers for these cells are different, they should be merged. The inside corner template and sample layout section are shown in Fig. 10.5-1*e*.

The procedure just described produces a list of nodes for each level and a list of nodes that should be merged. Other information is also kept: for example, a count of the number of times each node is encountered (the area), a count of the number of nodes along an edge (the perimeter), and the location of the first occurrence of each node. In addition, relationships between levels such as contact cuts result in a second node merge list. This node merge list must be kept separate from the homogeneous node merge list since the contact cuts represent nodes of different materials that are connected. Electrically they represent the same circuit node, but for capacitance and resistance calculations their individual identity, area, and perimeter must be maintained.

Other interactions between levels must also be considered. Wherever polysilicon and diffusion are coincident, an additional level (transistor) must be created. This artificial level is processed in a manner similar to the other levels to generate individual transistors and maintain their areas for capacitance and drive strength calculations.

10.5.2 Circuit Extractor Output

As a minimum, the output from a circuit extraction program should contain a complete list of transistors showing the type of transistor (p-channel, n-channel, depletion, etc.) and the nodes to which the transistor is connected. The circuit of Fig. 10.5-2 was extracted to show typical output. A sample of such output, called a net list, is shown in Fig. 10.5-3.

The extracted output of Fig. 10.5-3 lists an arbitrary transistor number; the drain (DS1), source (DS2), and gate (G) connections; the type of transistor (enhancement or depletion); the shape (*ok* means rectangular); the length and width of the transistor; and the x and y coordinates of the upper left corner of the transistor. All dimensions are based on the parameter λ. The resolution of Fig. 10.5-2 and its extracted output listings is 0.5 λ. With this information, transistor size can be verified, individual transistors can be located, and the V_{DD} connection for the depletion transistors (the normal case) can be verified. The net list provides sufficient information to allow reconstruction of a transistor-level circuit diagram

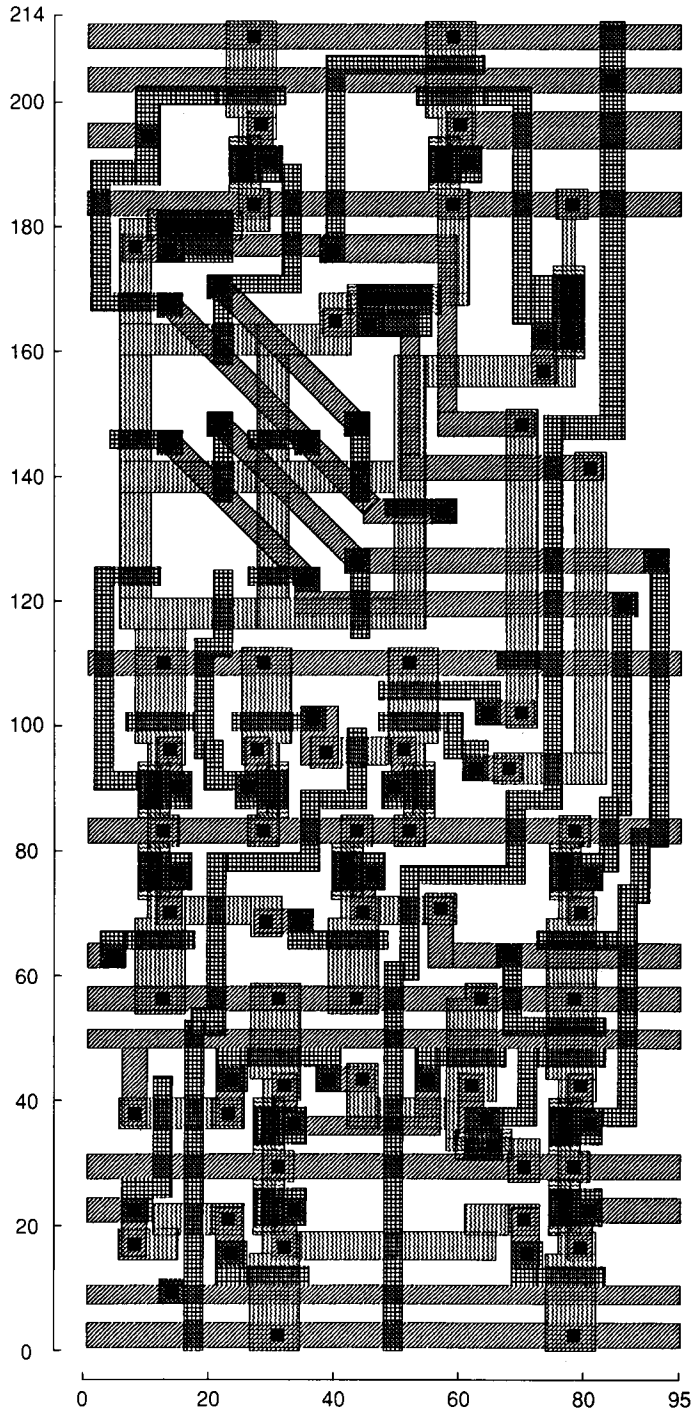

FIGURE 10.5-2
Sample layout for circuit extraction.

LEXTRACT version 3.337

Date 4-MAR-89 Time 19:54:46

X min = 0.0 X max = 95.0
Y min = 0.0 Y max = 214.0

Macro name is BGMLT
Macro number is 99

Final merge node list

Num	DS1	DS2	G	Type	Shape	Length	Width	X-loc	Y-loc
1	GND	42	3	enhN	ok	3.0	8.0	54.0	208.5
2	GND	6	4	enhN	ok	3.0	8.0	22.0	203.5
3	42	7	5	enhN	ok	3.0	8.0	54.0	203.5
4	6	VDD	6	depN	ok	6.0	2.0	24.0	194.0
5	7	VDD	7	depN	ok	6.0	2.0	56.0	194.0
6	3	VDD	3	depN	ok	12.0	2.0	11.0	182.0
7	VDD	5	5	depN	ok	12.0	2.0	76.0	173.0
8	3	51	4	enhN	ok	3.0	5.0	5.0	170.0
9	9	VDD	9	depN	ok	12.0	2.0	43.0	170.0
10	51	9	6	enhN	ok	3.0	5.0	20.0	165.0
11	51	55	11	enhN	ok	3.0	5.0	5.0	148.0
12	9	12	4	enhN	ok	3.0	5.0	27.0	148.0
13	55	12	10	enhN	ok	3.0	5.0	20.0	143.0
14	12	5	6	enhN	ok	3.0	5.0	42.0	143.0
15	5	13	4	enhN	ok	3.0	5.0	49.0	137.0
16	55	GND	14	enhN	ok	3.0	5.0	5.0	126.0
17	12	17	11	enhN	ok	3.0	5.0	27.0	126.0
18	GND	17	15	enhN	ok	3.0	5.0	20.0	121.0
19	17	13	10	enhN	ok	3.0	5.0	42.0	121.0
20	3	18	2	enhN	ok	3.0	5.0	67.0	112.5

FIGURE 10.5-3
Partial net list generated from Fig. 10.5-2 by circuit extractor (VDD and GND labels entered by user).

(not shown) for the integrated circuit. The extracted circuit diagram can be compared with the intended circuit diagram for omissions or errors.

Additional information based on the circuit extraction should be provided. For example, for each integrated circuit layout level, a complete list of nodes with their corresponding areas and perimeters can be provided. If the capacitance per unit area is known for each level, the circuit extraction program can provide an accurate estimate of the capacitance at each node. Figure 10.5-4 provides a partial circuit extractor output for the layout of Fig. 10.5-2 showing the details of the integrated circuit layers that form the nodes of a circuit. For each extracted geometry, this output lists the area, top edge length, left edge length, x and y coordinates of the upper left corner of the geometry, the new merged node number, the old node number assigned to the geometry during extraction, the layout level, and the node name (if any).

The output of Fig. 10.5-4 shows that node 1* is composed of a diffusion geometry (level 1) with area of 84 square units and perimeter of 37 units, a metal

LEXTRACT version 3.337

Date 4-MAR-89 Time 19:54:46

X min = 0.0 X max = 95.0
Y min = 0.0 Y max = 214.0

Macro name is BGMLT
Macro number is 99

Final merge node list

Area	Top	Left	X-loc	Y-loc	New	Old	Lev	Name
84.0	8.0	10.5	22.0	214.0	1*	1	1	
4.0	2.0	2.0	25.5	212.5		5	5	GND
380.0	95.0	4.0	0.0	213.5		4	4	
4.0	2.0	2.0	57.5	212.5		6	5	GND
44.0	8.0	5.5	54.0	214.0		2	1	
254.0	4.0	63.5	82.0	214.0	2*	3	3	Phi-2
4.0	2.0	2.0	83.0	205.5		11	5	
154.0	13.0	38.0	73.0	150.5		87	3	
380.0	95.0	4.0	0.0	206.5		9	4	
90.0	10.5	22.5	65.5	112.5		135	3	
54.0	9.0	12.0	67.0	90.0		193	3	
97.5	20.0	15.5	50.0	78.0		222	3	
195.0	5.5	62.5	47.5	62.5		260	3	
27.0	5.0	6.0	24.0	188.0	8*	33	1	
4.0	2.0	2.0	25.5	185.5		38	5	VDD
12.0	6.0	2.0	23.0	182.0		46	1	
380.0	95.0	4.0	0.0	186.5		37	4	
4.0	2.0	2.0	57.5	185.5		39	5	VDD
87.0	5.0	18.0	56.0	188.0		34	1	
12.0	6.0	2.0	55.0	170.0		68	1	
4.0	2.0	2.0	76.5	185.5		40	5	
43.0	5.0	14.0	75.0	187.0		36	1	
193.0	15.0	20.5	5.0	123.0	16*	122	1	
4.0	2.0	2.0	11.0	111.5		137	5	GND
380.0	95.0	4.0	0.0	112.5		134	4	
4.0	2.0	2.0	27.0	111.5		139	5	GND
84.0	8.0	10.5	24.5	113.0		132	1	
335.0	95.0	15.5	0.0	51.5	30*	273	4	
4.0	2.0	2.0	6.5	39.0		308	5	B-in
27.5	5.5	5.0	5.0	40.5		299	1	

FIGURE 10.5-4
Partial layer detail generated by circuit extractor for Fig. 10.5-2.

geometry (level 4) with area of 380 square units, another diffusion geometry (level 1) of 44 square units area and 27 units perimeter, and two contacts (level 5) with area 4 square units each. The x and y coordinates of the upper left corner of each geometry are given, allowing location of the geometry on a display or plot. With the area and perimeter sizes determined, calculation of interconnection capacitances is relatively easy using the values from Table 10.5-1. Example 10.5-1 demonstrates this calculation.

TABLE 10.5-1
Typical capacitance values (from Table 2B)

Layer	Capacitance
Metal	0.025 fF/μ^2
Polysilicon	0.045 fF/μ^2
Gate	0.7 fF/μ^2
Diffusion (bottom)	0.33 fF/μ^2
Diffusion (sidewall)	0.9 fF/μ

Example 10.5-1 Calculation of nodal interconnect capacitance. For a typical MOS process, parasitic capacitance values to ground are given in Table 10.5-1. Determine the total capacitance for node 1* of the circuit extraction output given in Fig. 10.5-4. The units of extracted dimensions are μ.

Solution: The total capacitance to ground at node 1* is the sum of the capacitance of the layers that compose the node (the contact capacitances are neglected). The capacitance can be calculated as follows.

$$C_{\text{total}} = C_{\text{diff}} + C_{\text{sidewall}} + C_{\text{poly}} + C_{\text{metal}}$$

$$C = (84 + 44)0.33 + (37 + 27)0.9 + (0)0.045 + (380)0.025 \text{ fF}$$

$$C = 42.24 + 57.6 + 9.5 \text{ fF}$$

$$C = 109.34 \text{ fF}$$

If the geometrical specification language allows names to be assigned to nodes, the names can be associated with their respective nodes by the circuit extraction program. The ability to name nodes adds to the complexity of the circuit extraction program since the name information must be kept after the geometric layout is instantiated. This adds substantially to the active computer memory required during a circuit extraction.

A node list with associated names is particularly valuable when checking for open circuits and short circuits. For example, if all power and ground nodes are named (V_{DD} or GND) and an individual node is associated with both the names V_{DD} and GND, a short circuit between power and ground is indicated. This is *not* desirable! Conversely, if the name GND is associated with two disjoint nodes, an open circuit may be indicated for the GND node. Of course, these same name tests can be applied to signal nodes and names, and this can be automated to report potential problems. Figure 10.5-4 shows circuit extractor output for a circuit with named nodes.

The nodes of Fig. 10.5-4 are named GND, Phi-2, VDD, and B-in. The fact that two separate nodes (1* and 16*) are named GND is cause for suspicion. This may indicate a discontinuity in the ground connection or, as in this case, it may be the result of extracting a partial layout. It is very important to provide node names early in a design and carry these names through the layout and simulation steps.

10.5.3 Interface to Other Programs

The output from a circuit extraction program can provide valuable input to circuit and logic simulation programs. Without circuit extraction results, circuit and logic simulations are based on manual input of the intended circuit connections and estimated circuit parameters. If certain process characteristics such as layer capacitance and transistor conductance are provided, a computer program can combine the circuit extraction output with process characteristics to create an input file for circuit simulation and logic simulation. Automatic generation of the input files eliminates human error in providing these data and allows accurate specification of capacitance values and transistor sizes.

Many modern integrated circuits are designed with a high-level circuit description provided in the form of a hardware design language (HDL). If this high-level description allows specification of circuit connections, a particularly important check on circuit integrity can be performed as a result of circuit extraction. The top-down circuit description from the HDL can be compared directly with the bottom-up circuit description from the extracted circuit. This check is valuable because it allows comparison between the designer's intent and the actual computer specification used to generate the fabrication masks.

Circuit extraction is a valuable design verification tool. With the aid of an LVS program, the extracted circuit can be compared to the intended circuit. Circuit extractor derived capacitances and resistances are extremely valuable for accurate circuit simulation. The use of named nodes in the geometrical specification file and subsequent extraction of these nodes allows open, short, and circuit continuity tests.

10.6 DIGITAL CIRCUIT SIMULATION

Accurate circuit simulation is essential for the design of analog circuits such as filters, comparators, and operational amplifiers. The need for circuit simulation extends to the design of semiconductor memory chips even though their data are stored in binary or digital form. For example, extremely sensitive sense amplifiers are required within DRAM circuits to respond to the small change in voltage caused by selecting a storage cell. SRAM circuits often use differential sensing circuits to increase the speed of the data access operation. Both of these memory types require accurate circuit simulation for proper design. Circuits whose external operation is totally digital may require accurate circuit simulation to model critical signal-delay paths. Circuit simulations of high accuracy are almost universally performed with a version of the SPICE circuit simulator described in Chapter 4.

Because of the large number of transistors in digital circuits such as microprocessors, peripheral controllers, and digital signal processors, it is not computationally feasible to perform a circuit simulation for the entire circuit. Since the execution time of circuit simulation programs increases at a rate that is only slightly less than the square of the number of nodes under consideration, verification of the operation of large circuits must be accomplished by other means. Many times a simulation at the logic or switch level (described in the next section) can

provide sufficient verification of a digital circuit's functionality. Sometimes even logic simulation programs are too slow to model an entire processor's behavior. Special-purpose hardware simulators are required in these cases.[14]

An intermediate class of circuit simulators is being investigated to provide accurate circuit simulation without the computational penalty of a full circuit simulator.[15,16] These new simulators usually depend on one of two characteristic features of digital circuits. First, most digital circuits are loosely coupled. This means that disjoint parts of the circuit may be relatively independent of one another. There are methods that take advantage of this structure by partitioning the network to simplify the equations that must be solved during a simulation. Second, only a small part, perhaps 25%, of a digital circuit is active during each clock cycle. If a circuit simulator can take advantage of those quiescent portions of the circuit, then only a small part of the circuit will result in simulation calculations at any given time. In either of these two cases, accurate digital circuit simulation can proceed at a relatively rapid rate compared to standard circuit simulation. Nonetheless, digital circuits of any size are rarely simulated in their entirety with circuit-level simulators. Rather, switch-level or logic-level simulators are preferred. Such simulators are described in the next section.

10.7 LOGIC AND SWITCH SIMULATION

Digital integrated circuits are designed to operate with binary representations for data. The basic presumption is that only two logic states are important for each signal line. Thus, knowledge of a precise voltage versus time characteristic for each node in the circuit is not necessary to design or analyze digital circuits. For many purposes, this simplifies both the circuits and their analysis compared to analog circuits. Nevertheless, computer simulation and verification of a circuit's functionality are necessary. Even though a digital circuit is designed based on logic gates, the logic gates are fabricated from the basic transistors and conductors allowed by the integrated circuit process. Therefore, it is often the case that the electrical operation of a simple logic circuit must be characterized by using a circuit simulator such as SPICE.

Though circuit simulation of digital circuits is frequently used, such circuit simulation has several drawbacks. As described in the previous section, the large number of logic gates in most digital integrated circuits precludes circuit simulation of the entire system because of the extended computer time required. Also, standard circuit simulators provide more detail about circuit voltages than is required to analyze a logic circuit. In an effort to reduce computer simulation time and to provide appropriate data to characterize the operation of digital circuits, *logic simulators* were developed.

10.7.1 Logic-level Simulation

Logic simulators allow specification of the operation of a circuit block in terms of its behavior. For example, a simple logic gate is described by its behavior, such as AND, OR, or NOT. More complex digital blocks such as full adders

and multiplexers are each described by their corresponding behavior rather than their circuit structure. The circuit inputs are specified as binary values that change at discrete time intervals. Logic simulator outputs are provided as binary values as well. Pure logic simulation does not model time delays through logic blocks. Only the logical behavior of the simulated system is considered, although the concept of sequence wherein one action precedes another is important. Timed logic simulation considers the delays of logic gates and blocks in determining when outputs will change. Because a logic simulator models the circuit in terms of an abstracted (less detailed) representation, larger circuits can be simulated in a much shorter length of time than with circuit simulation. Consider the following example.

Example 10.7-1 Comparison of circuit and logic simulation. In terms of the number of circuit elements, nodes, and calculations, compare circuit simulation and logic simulation requirements for a full adder built from a classical CMOS circuit and from CMOS gates.

Solution

Circuit simulation. The two-level logic circuit for a classical CMOS full adder requires 56 transistors and 33 nodes. This circuit is shown in Fig. 10.7-1. In addition, continuous input waveforms that generate the eight possible logic input conditions of three inputs must be provided. Each of these conditions must be stable for a length of time sufficient to allow the sum and carry outputs to stabilize. This requires about 100 to 200 time steps for each input condition. As a rough estimate, a minimum of 800 to 1600 time-step calculations would be required to characterize the full-adder operation.

Logic simulation. A classical two-level logic circuit for a full adder requires three inverters, three 2-input NAND gates, five 3-input NAND gates, and a 4-input NAND gate, for a total of 12 logic gates and 15 nodes. The logic gate implementation is given in Fig. 10.7-2. Eight possible input combinations exist for the full adder. Each of these combinations generates a digital value for the sum and carry outputs. Correctness of the sum and carry outputs is easily verified by these eight calculations.

Thus, circuit simulation requires approximately 1600 time-step calculations involving 56 transistors and 33 nodes at each calculation. Logic simulation, on the other hand, requires only 8 calculations, involving 12 logic gates and 15 nodes for each calculation. Clearly, if simulation of the logical operation of the full adder is the goal, logic simulation is simpler and faster. If accurate signal propagation time or waveform characteristics are required, then circuit simulation is necessary.

Commercial logic simulators model digital logic in terms of four or more states. As a minimum, these states include 1, 0, X, and Z. The logic values 1 and 0 model the high and low logic states. The value X is used to model an unknown condition. For example, the value of an internal logic node may be unknown when simulation is started. The value Z is used to model high-impedance (undriven) nodes. A tri-state bus with all driving circuits turned off is an example of this condition. Additional states may be defined to model the relative driving strength of logic outputs. Of course, as the number of allowable states increases, the simulator complexity and run time increase correspondingly.

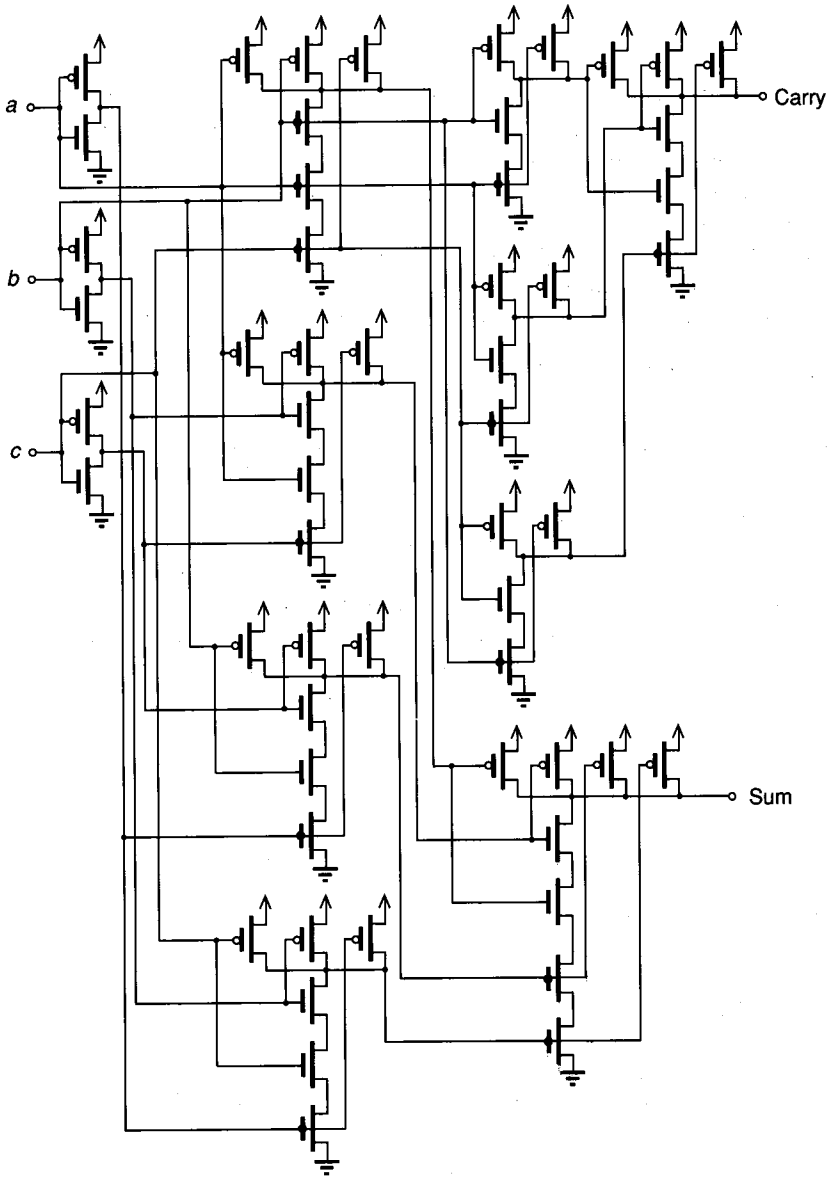

FIGURE 10.7-1
Classical CMOS full-adder circuit.

Many logic simulators provide a variety of digital blocks for use in modeling a digital system. Besides the simple logic gates and more complex logic blocks mentioned previously, models for large digital blocks such as ROMs, RAMs, PLAs, ALUs, and even FSMs are often provided. Simulation capability is normally provided for both synchronous and asynchronous sequential circuits in addition to simple combinational logic.

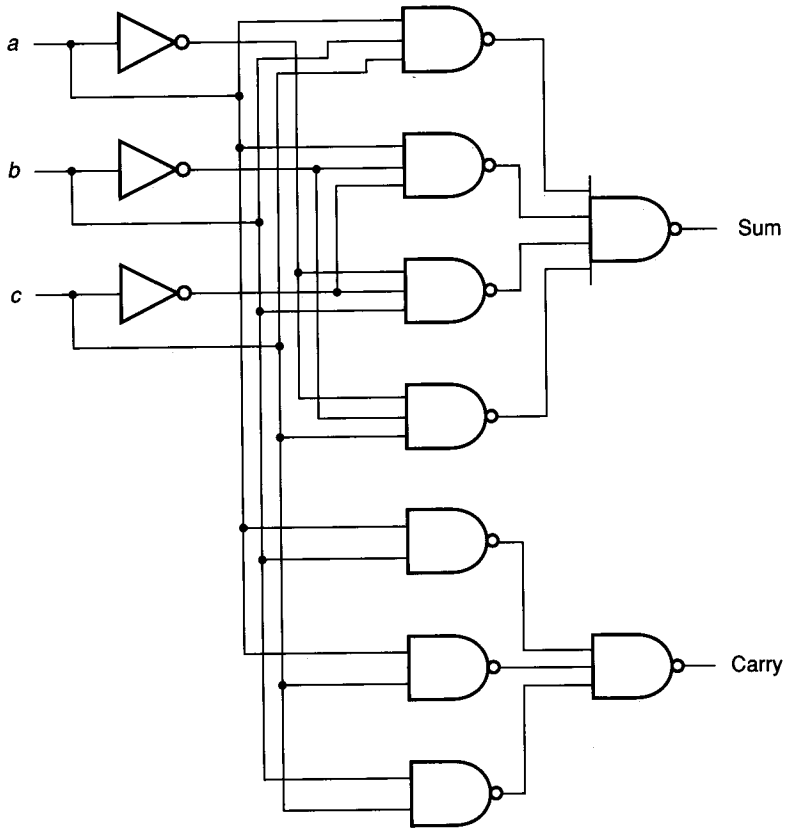

FIGURE 10.7-2
NAND-NAND full-adder logic diagram.

Most logic simulators today are *event driven*. That is, calculations are required only in response to external or internal events. External events include changes in the state of inputs to the circuit. An internal event occurs when the output of a logic function changes in response to changes in its inputs. For example, when the input to an inverter changes, the corresponding change in the inverter output is considered an event. The use of event-driven rather than fixed time-step simulation algorithms reduces the time required for simulation of a circuit.

The capability of logic simulation is often measured in terms of *events per second* or *evaluations per second*. Whenever the inputs to a logic block change, an *evaluation* must occur to determine the correct output for the logic block. Thus, an evaluation is the application of a circuit's inputs to its behavior in order to determine its outputs. An average factor of 2.5 evaluations per event is typical for digital circuits. The performance of logic simulators depends on many factors including the number of logic states, the cleverness of the algorithms chosen for simulation, and the execution speed of the computer on which the simulator is run. An execution rate of several thousand events per second is common for today's logic simulators on typical computer workstations.

10.7.2 Switch-level Simulation

MOS integrated circuits present special problems for standard logic simulators because of bidirectional pass transistors, transmission gates and charge storage. Pass transistors are used frequently because of their desirable power dissipation and interconnection characteristics. Pass transistors are difficult to simulate as simple logic gates with a standard logic simulator. It might seem that the pass transistor of Fig. 10.7-3*a* could be simulated by using the AND gate of Fig. 10.7-3*b*. The following discussion shows why this is impractical.

A simple analysis of the operation of the circuits of Fig. 10.7-3 shows that the two circuits are not equivalent. Assume initially that both inputs and the output are low for both circuits. If a logic 1 is placed on a single input, the output remains low for both circuits. If a logic 1 is placed on both inputs, the output goes high for both circuits. If a logic 0 is placed at the *i* input of the two circuits, the output goes to a 0 for both circuits. Thus far, the operation of the two circuits seems identical. However, assume that all inputs and outputs are initially high. Further, consider that the source diffusion of the output of the pass transistor provides parasitic capacitance to ground. If the *c* input to both circuits is moved to a logic 0, the AND gate output goes to a logic 0 while the pass transistor output remains high because of the charge storage at its output. Clearly, the operation of the pass transistor cannot be accurately modeled in this fashion. Either a more complex logic circuit is required, or the logic simulator must be modified to account for drive strengths and charge storage. Examples of drive strength are *driven*, *resistive pullup*, and *undriven*. The output of the pass transistor just considered is undriven when its gate terminal is at 0 V.

Because selector circuits constructed from pass transistors and transmission gates are widely used within MOS circuitry, a logic simulator for MOS must allow specification of individual transistors and their connections in addition to simple logic gates. When a logic simulator can describe transistors in addition to standard Boolean logical primitives, it is called a *switch-level simulator*.

A typical switch-level simulator operates on circuits described by nodes, transistors, and logic gate primitives. Nodes are equipotential points to which one or more terminals of one or more transistors or logic primitives are connected. Each node has an associated name, logic state, capacitance (to ground), list of events, and perhaps other information. Each transistor has a type (n-channel, p-channel, or depletion), effective resistance (width and length are required), and node connection for its terminals. Macros are often allowed to describe circuits composed of several transistors; for example, logic gates may be constructed from nodes and transistors. These logic gates are then used as primitives.

A byte-wide MOS binary adder circuit will be used as an example to show the operation of a switch-level simulator.[17] The circuit for a full-adder stage is

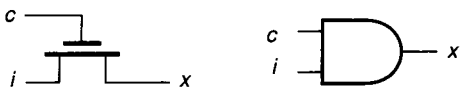

FIGURE 10.7-3
(*a*) Pass transistor logic, (*b*) AND-gate logic.

given in Fig. 10.7-4, and the corresponding input net list for the switch-level simulator is provided in Fig. 10.7-5. This net list describes the circuit of Fig. 10.7-4 in terms of four primitive elements: invert, trans, nor, and pulldown. The net list starts with a definition of a single-bit adder macro and its five inputs {*a b cif phi1 phi2*} and two outputs {*sum cof*}. Additionally, seven local signals {*af bf ci p pf k phi2f*} that are internal to the full-adder macro are specified. Each primitive element is then instantiated with its connections to other circuit nodes defined by arguments. The formats for these four procedure calls are: (invert out in), (trans gate source drain), (nor out in0 in1 in2), and (pulldown drain gate).

Next, eight single-bit full adders are combined to define a byte-wide binary adder, as shown in Fig. 10.7-6. The external nodes of the byte-wide full adder are first defined. The **a**, **b**, **cof**, and **sum** nodes represent 8-bit vectors that are expanded by the repeat statement. Signals *phi1* and *phi2* are the nonoverlapping two-phase clock inputs. The connect statement joins the *cifi* carry-input scalar to the first carry-in bit, *cof.0*. The repeat statement next creates eight copies of the full-adder circuit.

The results of a sample switch-level simulation run for the byte-wide adder are explained next. The input vector **a** was set to 11111111, while the input vector **b** was set to 00000000. This condition provides the longest carry propagation path for the full adder. The initial carry-in bit *cifi* is set to the low-true condition. A nonoverlapping two-phase clock is defined with each phase high for 90 ns and a 10 ns separation between phases. The results from a simulation for a complete

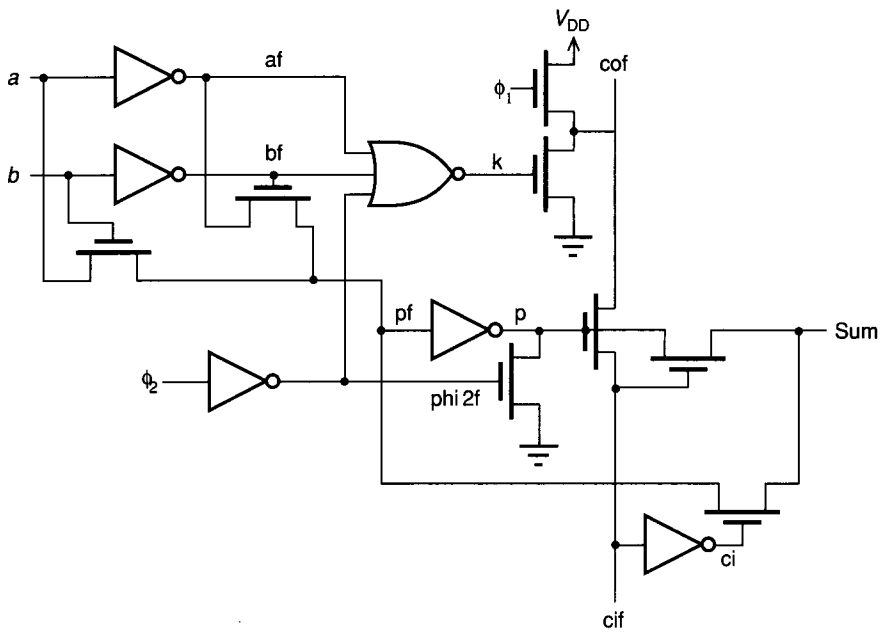

FIGURE 10.7-4
Single-bit slice of clocked full-adder circuit.

```
;Begin Full-Adder Macro
(macro adder (a b cif phi1 phi2 sum cof)
  (local af bf ci p pf k phi2f)
  (invert bf b)
  (invert af a)
  (trans b pf a)
  (trans bf pf af)
  (invert (p 2 16) pf)
  (invert (ci 2 16) cif)
  (trans cif sum p)
  (trans ci sum pf)
  (invert phi2f phi2)
  (nor k af bf phi2f)
  (pulldown cof k)
  (pulldown p phi2f)
  (trans phi1 cof vdd)
  (trans p cof cif)
)
;End Full-Adder Macro
```

FIGURE 10.7-5
Input net list for logic simulator describing circuit of Fig. 10.7-4.

```
;Instantiate Byte-Wide Adder
(node a b cifi phi1 phi2 sum cof)
(connect cifi cof.0)
(repeat i 1 8
  (adder a.i b.i cof.(1 - i) phi1 phi2 sum.i cof.i)
)
;End of Byte-Wide Adder
```

FIGURE 10.7-6
Input net list for a byte-wide binary adder.

cycle (200 ns) of the two-phase clocks are given for the first and last sum (*sum.1*, *sum.8*) and carry-out (*cof.1*, *cof.8*) bits only. Only changes in logic value of these bits are provided; that is, only simulator events for these bits are included. A typical event produces a statement with the format: name = value @ time.

ϕ_1 cycle: (t = 0 ns to 90 ns), precharge
cof.8 = 1 @ 2.4
cof.1 = 1 @ 2.6
sum.1 = 0 @ 2.8
sum.8 = 0 @ 3.2

ϕ_2 cycle: (t = 100 ns to 190 ns), evaluate
cof.1 = 0 @ 103.2
cof.1 = 1 @ 104.4
sum.8 = 1 @ 104.9
cof.1 = 0 @ 109
sum.8 = 0 @ 129.8
cof.8 = 0 @ 130

Note that all carry bits are precharged to a 1 during each ϕ_1 cycle. According to the simulation results shown, the *cof.8* bit changed to 1 at 2.4 ns and the *cof.1* bit changed to 1 at 2.6 ns after ϕ_1 was set high. As can be determined from the circuit connections of Fig. 10.7-4, the sum bits should be set to 0 during each ϕ_1 precharge cycle. The *sum.1* bit went to 0 at 2.8 ns and the *sum.8* bit went to 0 at 3.2 ns after ϕ_1 was set high.

During the ϕ_2 evaluate cycle, the carry and sum bits are set according to the sum of the two addends **a** = 11111111 and **b** = 00000000 and the carry in *cifi* = 0 (indicates a carry in). During the evaluate cycle *cof.1* changed to 0 at 3.2 ns, to 1 at 4.4 ns, and then back to 0 at 9.0 ns after ϕ_2 was set high. The most significant carry bit, *cof.8*, was set to 0 some 30 ns after ϕ_2 was set high. Also, *sum.8* was set to 1 at 4.9 ns and then to 0 at 29.8 ns after ϕ_2 was set high. For the input vectors given, each full-adder stage should have set its sum bit to 0 to indicate a sum of 0 and its carry bit to 0 to indicate a carry out of 1 (the carry bits use negative logic). The final results from simulating the first clock cycle are as expected. Note that the final event (*cof.8* set to 0) occurred 30 ns after the ϕ_2 clock was set high.

Prior to the second clock cycle, the carry-in bit is set to a false condition (*cifi* = 1). The following simulation results are for the second clock cycle (200 ns $\leq t <$ 400 ns).

ϕ_1 cycle: (t = 200 ns to 290 ns), precharge
cof.8 = 1 @ 200.2
cof.1 = 1 @ 200.4

ϕ_2 cycle: (t = 300 ns to 390 ns), evaluate
sum.8 = 1 @ 304.9
sum.1 = 1 @ 304.9

During the second ϕ_1 cycle, the carry bits change as they are each precharged to 1. The sum bits do not change during ϕ_1 since they were already each left set to 0 after the previous ϕ_2 cycle. During the second ϕ_2 cycle, the sum and carry bits should be changed to indicate the sum of the two addends **a** = 11111111 and **b** = 00000000 and the carry in *cifi* = 1. Thus, all sum bits should be set to 1 and all carry out bits should be set to 1 indicating no carry out. The simulation results show that the sum bits are each correctly set to 1 during the second ϕ_2 cycle. The carry bits do not change since they were each set to 1 during the precharge cycle.

For a third clock cycle (400 ns $\leq t <$ 600 ns), the carry in bit is set to 0 again (*cifi* = 0) and the results of the first clock cycle are repeated. These results are as follows.

ϕ_1 cycle: (t = 400 ns to 490 ns), precharge
sum.1 = 0 @ 402.8
sum.8 = 0 @ 403.2

ϕ_2 cycle: (t = 500 ns to 590 ns), evaluate
cof.1 = 0 @ 503.2
cof.1 = 1 @ 504.4
sum.8 = 1 @ 504.9
cof.1 = 0 @ 509
sum.8 = 0 @ 529.8
cof.8 = 0 @ 530

The previous results for three clock cycles demonstrate the operation of a switch-level simulator. Both the timing of the byte-wide adder and the correct logical operation of the adder are observed for the input conditions provided. Other switch-level simulators have different input and output formats and different capabilities, but all operate assuming discretized values for the circuit variables, and all produce results much faster than complete circuit simulation.

10.7.3 Hardware Logic Simulation

Even with the increased speed of logic simulators as compared with circuit simulators, full simulation of large digital circuits via general-purpose computers is not practical. An alternate approach is in use by several companies. Special-purpose hardware that executes many simulation steps in parallel has been developed to speed the simulation process. One of the early, large parallel simulators was the YSE (Yorktown Simulation Engine)[18] developed by IBM. This hardware consists of hundreds of identical processing units that each simulate part of the target circuit. By spreading the calculations over a large number of processors, even large-mainframe computers can be simulated in detail. Of course, such special-purpose hardware is expensive to build and to operate. Even so, several companies now offer hardware accelerators to enhance the speed of logic simulation.

In the future, methods of machine verification other than total logic simulation must be found. Logic simulation time increases exponentially with the number of logic components to be simulated. Thus, faster computers are necessary to simulate next-generation computers that contain more logic components. But how can the next-generation computers be built if the simulation capability of present-generation computers is inadequate?

Two current approaches to this problem are verification proofs and hierarchical simulation. For relatively simple hardware, it has been possible to verify correct logical operation by mathematical proofs. Unfortunately, the utility of this method diminishes quickly as the size and complexity of the hardware increase. The second method, hierarchical simulation, attempts to model the target machine at various levels of abstraction. Small blocks of hardware are verified by logic simulation. These blocks are then interconnected and simulated together without the internal detail of each block. Neither of these methods has been entirely successful, and both are now active areas of research and development.

10.8 TIMING ANALYSIS

For most digital circuits, a very important parameter is the maximum rate at which the circuit can correctly process data. For microprocessors, the processing speed is usually given in MIPS (millions of instructions per second); for scientific calculations, the rate of execution is given in FLOPS (floating-point operations per second); and for logical inference machines, the characteristic measure is LIPS (logical inferences per second). The execution rate of each of these machines is limited by parasitics and governed by its input clock. A primary goal in the design of a digital computing machine is to operate with the fastest possible input clock.

Each digital integrated circuit has a maximum rate of operation. This rate of operation is limited by the output drive capability of its logic elements and by the capacitance and resistance of the loads they must drive. In a FSM (finite-state machine), the clocking rate is limited primarily by the longest path through its combinational logic section. For integrated circuits composed of large blocks of circuitry, the maximum clocking rate may be limited by signal lines that must carry information between the blocks. The designer's task, then, is to find those paths in an integrated circuit design that cause the maximum delay and then to modify the circuitry to minimize that delay.

Finding the longest delay paths, called *critical paths*, for an integrated circuit is not a simple task. Until recently, the most common technique for finding critical delays was for the designer to perform detailed circuit simulation on the paths that were suspected of contributing long delay times. Of course, using circuit simulation for this task was not foolproof. Many times an unsuspected path that was not considered for simulation would limit the maximum clock speed. More recently, computer programs have been designed specifically to seek out delay paths directly from the circuit definition without requiring simulation. This type of computer analysis is called *timing analysis*.

10.8.1 Timing Analysis Methodology

Timing analysis differs from circuit and logic simulation in that all possible signal paths are considered. Circuit simulation and logic simulation both require the specification of input signals to control the simulation. Thus, only delay paths that are exercised by the particular set of inputs are tested. For many digital circuits, it is computationally impossible to provide sufficient input conditions to test the circuit fully. Timing analysis tools work by tracing *signal paths* instead of simulating the circuit for specific inputs. Specifically, timing analysis uses *state-independent* path tracing. Each time a logic gate is encountered, the gate is assumed to pass the signal regardless of the state of the other inputs to the gate. A signal path is terminated only when an output is reached or a clocked storage element is found. With this method, all possible delay paths are tested.

An example of timing analysis signal propagation through two logic gates is shown in Fig. 10.8-1. The signal path starts at input x and reaches the NAND gate. Inputs a and b for the NAND gate are assumed high to allow continuation of the signal path. When the signal reaches the NOR gate, input c is assumed low

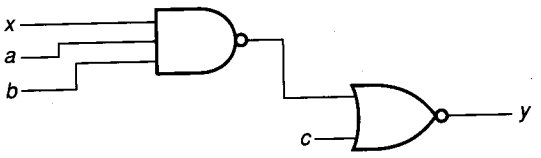

FIGURE 10.8-1
State-independent path trace.

to allow continued propagation of the signal. Finally, the signal reaches an output y, where it terminates. The delay for this signal path includes the contributions of the NAND gate, the NOR gate, and the series interconnections. The delay paths from x to y, b to y, a to y and c to y would all be found by a timing analysis of this circuit.

A second example shows a deficiency of timing analysis. From Fig. 10.8-2, signal paths from a to b and from a to c are expected. However, state-independent path tracing will also find a signal path from b to c and vice versa. Although the path from b to c is a real path, it will not normally be exercised within this circuit because node n is actively driven by the inverter. Analysis of additional paths that will not be exercised during operation of a circuit can degrade the performance of a timing analysis program. Circuit-level timing analyzers allow direction setting for pass transistors and transmission gates to circumvent this problem. Unfortunately, unless this is carefully done, some critical signal paths may be eliminated from consideration.

10.8.2 Timing Analysis Tools

To provide further insight into the capabilities of circuit-level timing analysis programs, two such programs will be described here. The first of these, called TV,[19] attempts to set directions for circuit elements by using rules. These rules, by setting some transistor directions, minimize the number of false paths that are found. The second tool, Crystal,[20] provides a wide range of capability, including improved delay models and coverage for circuits built from CMOS technology.

TV timing analyzer for NMOS designs, operates from extracted circuit parasitics and considers only stable, rising, and falling signal values. Program execution time is minimized by a *static analysis* that sets signal flow direction and clock qualification where possible. Otherwise, signal flow direction is determined from a set of direction-finding rules. Some of the rules are independent of design style. For example, the *constant-propagation rule* says that any transistor source

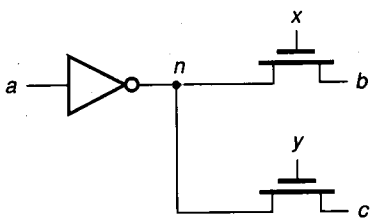

FIGURE 10.8-2
Problem paths for timing analysis.

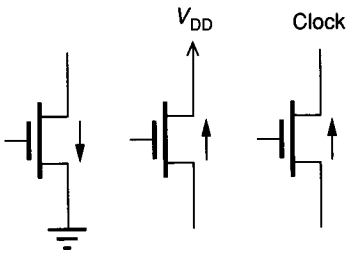

FIGURE 10.8-3
Constant propagation rule to set directions.

or drain connected to power, ground, or a clock must be a sink of signal flow, while the other terminal must be a source. Figure 10.8-3 demonstrates this rule, which by itself sets the directions for more than half the transistors in a typical circuit. Another rule, demonstrated in Fig. 10.8-4, is based on Kirchoff's current law. This rule, the *node current rule*, states that if all but one of the transistors to a node have a known direction, and the known transistors all sink or all source signal flow, then the unknown transistor must transmit flow in the opposite direction relative to the node.

Other signal-flow rules depend on technology or design style. For example, in an NMOS technology design, the *k-ratio rule* for inverters can be used to set direction. This rule is based on a standard device sizing ratio k as discussed in Chapter 7 for ratio logic. By finding the minimum resistance to ground through each unset (direction not specified) transistor connected to a pullup, a transistor can be considered as a pulldown (signal flow toward the pullup) or a pass transistor (signal flow away from the pullup), depending on the resistance ratio. The reasoning is that resistances to ground that satisfy the device sizing ratio k with respect to the pullup path must be part of the pulldown circuit for a logic gate. Transistors that cannot satisfy ratio rules can be safely classified as pass transistors and their direction set accordingly. Other rules cover pass transistors connected to a common node and having a common gate signal, and analogous structures where the direction of a boundary transistor can be determined, thereby allowing arrayed versions of the structure to have their directions set accordingly.

Signal path analysis is started from the clock or other input nodes. Paths are investigated in a breadth-first manner in accordance with the transistor directions that were set by the static analysis. Delays for paths are calculated based on the capacitance of the interconnections and the resistance of driving and series pass

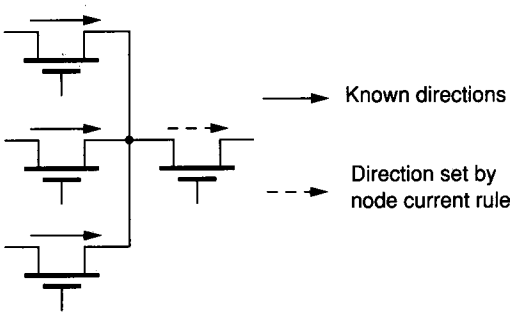

FIGURE 10.8-4
Node current rule.

transistors. Transistors are assigned a rising and a falling resistance from tables based on their use in the circuit. Signal direction changes are propagated so that a rising input signal to an inverter produces a falling output signal and vice versa. Pass transistors continue the direction of the input signal. Because path delays are calculated from a linearized model, the delays may differ from actual circuit values by 30% or more.

Output of the TV program includes a user-selectable number of the worst-case paths. Equivalent paths, such as parallel paths in a data bus, are condensed in the output list so that only the last path in the list is reported. Other useful information such as slack time for paths, excessive power used to drive a noncritical path, and nodes with unusually high capacitance are reported. The TV timing analyzer was successfully used in the analysis of the MIPS series of microprocessor chips[21] developed at Stanford University.

Another timing analysis tool, Crystal, was developed to analyze the RISC computer chips[22] developed at the University of California at Berkeley. This tool has found widespread use throughout the VLSI design community, particularly within universities. The timing analysis is based on a circuit description that is extracted directly from a geometrical specification file. This description includes transistor sizes and types, interconnection capacitance, and a rough calculation of interconnection resistance. A simple delay model is used for each stage to provide quick calculation of signal propagation delays along a path.

The Crystal timing analyzer was developed for MOS circuits with multiple nonoverlapping clocks. The program attempts to determine how long each clock phase must be to allow all signals to propagate to their destinations. The analysis is state-independent, so all possible paths are checked. The user must specify a minimum of information to begin the analysis. For two-phase clocking schemes, only two signals must be specified. One of the clock phases is specified as a rising edge or a falling edge to trigger the analysis. The other clock phase is specified as a stable low value. The reason for this can be seen from the shift register circuit of Fig. 10.8-5. Here a signal path trace is started from the ϕ_1 clock. Without a specified value for the ϕ_2 clock, the signal path would continue through all the stages shown. If the ϕ_2 clock is set to a stable low condition, then the signal path will terminate correctly after the first stage. The path delay

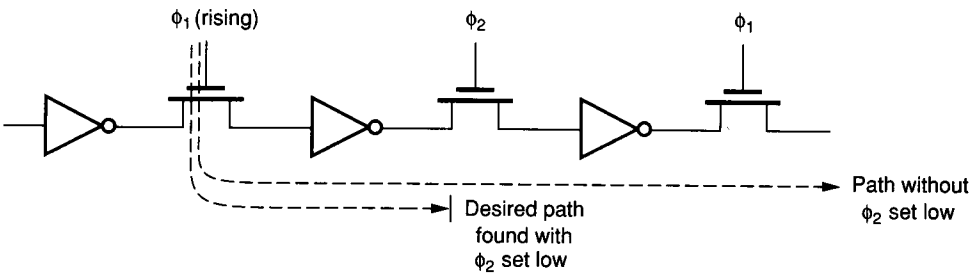

FIGURE 10.8-5
Clocked path analysis.