29 April 1983, Volume 220, Number 4596 SCIENCE

Design Automation for Integrated Circuits

Sydney B. Newell, Aart J. de Geus, Ronald A. Rohrer

Summary. With the ever-increasing complexity of integrated circuits, manual design methods have become intolerably slow and error-prone. The use of computers to automate some or all of the design process is necessary to minimize both design time and error incidence. In this article are discussed the design and fabrication of integrated circuits, selected techniques of design automation, and the problems associated with such automation.

The complexity of integrated circuits is constantly increasing and the physical size of their individual components decreasing; for commercially available integrated circuits the complexity doubles every year (*1–3*). Like any rapidly expanding field, microelectronics is experiencing "growing pains" because some of its areas are not keeping pace with the rest of the field.

In the progression from idea to integrated circuit, the first phase, design, encompasses all tasks up to manufacturing. The second phase, fabrication, deals with the physical creation of the integrated circuit. Of the two phases, design is by far the more expensive and time-consuming. For example, to design a microprocessor chip containing 60,000 to 70,000 transistors can require dozens of man-years and millions of dollars. But, after the initial setup of the manufacturing process, fabrication of such a chip can take just weeks and cost thousands of dollars. Progress in integrated circuit development may be slowed or halted by the time- and cost-intensive design process unless help is forthcoming (*1–5*). For this help, designers are turning more and more to computers.

Computer-aided design (CAD) has been in use almost since the inception of integrated circuits (*3*). In CAD, computers analyze circuit and system behavior and designers use the results for guidance in correcting or enhancing their designs. Thus the role of the computer in CAD is one of an assistant to the designer, who carries out the actual design tasks by making decisions based on the CAD results.

In design automation, not only the analytical but also many of the synthetic design tasks are performed by computers. Computers carry out a given design task by performing a series of iterations and are guided by the analytical results of each iteration to improve the design until the desired specifications are met. Thus in design automation using a form of artificial intelligence, the computer decides what actions to take and carries out the design tasks with little or no human intervention.

Design automation is in a relatively primitive state compared to CAD, and represents one of the last frontiers in cutting design time, cost, and errors. Several reviews (*3, 6–8*) of the status of design automation are directed to specialists in the field; this article will provide an overview of design automation for scientists and technologists who are not involved in integrated circuit design.

A brief introduction to integrated circuits and their fabrication may be helpful in demonstrating the necessity for streamlining the design process. For simplicity, throughout this article we confine our discussions to digital circuits.

Integrated Circuits and Their Fabrication

An integrated circuit is a circuit contained on (or in) a continuous piece of solid material (usually silicon) called a die or chip. Components and wiring are fabricated simultaneously onto an integrated circuit. (In contrast, discrete steps are required for placing components and wiring onto a printed circuit board.) The circuit itself is usually specified by a logic diagram, which is an interconnection of logic gates.

Digital (binary) logic is composed of logic circuits. A logic circuit is realized with logic gates, each of which has inputs and outputs and performs a logical operation. In such an operation a set of variables having the complementary values 0 and 1 are treated as "false" and "true," respectively; translated to electronics, a 0 is usually implemented with a low voltage (around 0), and a 1 is usually implemented with a relatively high voltage (around 5 volts). Addition, subtraction, storing, counting, controlling, and many other functions are achieved by specific interconnections among logic gates.

Figure 1 gives names, symbols, and truth tables for some simple logic gates and their corresponding operations. A truth table shows the inputs on the left of a vertical line and the outputs on the right. Thus the truth table for the AND gate shows that the output, Y, is 1 only if both inputs A and B are 1; the output is 0 for any other combination of inputs.

The binary logic of digital circuits may be implemented by transistors, which act as switches. By connecting transistors in different ways, any desired electronic circuit function can be realized. Integrated circuits, commonly 0.25 to 1 square

S. B. Newell is Manager of Technical Documentation and A. J. de Geus is Program Manager of Simulation and Test Automation at the General Electric Microelectronics Center, Research Triangle Park, North Carolina 27709. R. A. Rohrer is Director of Electronic Marketing at the CALMA Company, Santa Clara, California 95050.

centimeter in area and about 1 millimeter thick, owe their compactness to this basic circuit element. Today, an integrated circuit may contain from ten to hundreds of thousands of transistors, each measuring about 150 square micrometers.

The complexity of an integrated circuit is described by the number of transistors it contains. In small-scale integration (SSI) a chip contains up to 100 transistors. Subsequent levels of complexity include medium-scale integration (MSI), with up to 1000 transistors; large-scale integration (LSI), with up to 10,000; and very large scale integration (VLSI), with more than 10,000 transistors. An integrated circuit is three-dimensional and can consist of up to 12 layers, each layer being composed of a semiconductor, a conductor, or an insulator.

The semiconductor is usually silicon to which small, controlled amounts of an impurity (called a dopant) have been introduced to provide carriers for current. Semiconductors are of two types. In an n-doped semiconductor, a group V element (for example, phosphorus) is the dopant and provides electrons as current carriers. A p-doped semiconductor contains a group III element (for example, boron), which provides positive, electron-deficient regions called holes as current carriers. Larger amounts of doping produce higher conductivities; the substrate, or supporting material for the circuit, is usually lightly doped with either type of semiconductor. Electrodes of transistors are made of semiconductor material, but usually contain more dopant than the substrate.

The conductor may be a metal (for example, aluminum) or polysilicon, a polycrystalline form of silicon that has been heavily doped. Strips of conductors form electrical connections within and between circuit elements. The insulator is usually silicon dioxide and is used to separate conducting regions where no connections are desired.

Integrated circuits contain several layers of various combinations of differently doped material, interconnections, and silicon dioxide. In the fabrication of integrated circuits, the layers are added one at a time. A template called a mask determines the pattern for each layer; 12 masks would be required to make an integrated circuit with 12 layers. Fabrication takes place by various multistep combinations of oxidation, mask protection, etching, diffusion or ion implantation, and vapor deposition. The whole cycle can take 4 weeks to several months on a commercial production line.

At various stages the chips are checked for possible defects, and thorough functional testing is done after their completion. Because of the integral nature of integrated circuits, a defective chip cannot be repaired; it must be discarded. Defects have two origins: manufacturing and design.

Manufacturing defects are random and will affect a certain percentage of chips on a statistical basis. The yield, the number of good chips divided by the total, decreases as the active chip area (the area occupied by components and wiring) increases. For a chip 0.2 inch on a side, doubling the active chip area causes a nearly sixfold decrease in yield (*9*).

A design defect is traceable to one or more of the masks and will, of course, affect all chips in that fabrication series. Some design defects are detectable early in the fabrication process; if a design defect is found, the process can be halted and the defect corrected. In some cases a second iteration of design and fabrication is needed to correct design defects in the first-pass chip.

Two economic reasons dictate that redesign and refabrication be held to a minimum. First, design and production of an integrated circuit are extremely expensive. Second, delay in getting the chip to the marketplace results in lost sales and loss of a potential share of the market. To minimize the necessity for redesign and refabrication, the design process must generate a valid set of masks.

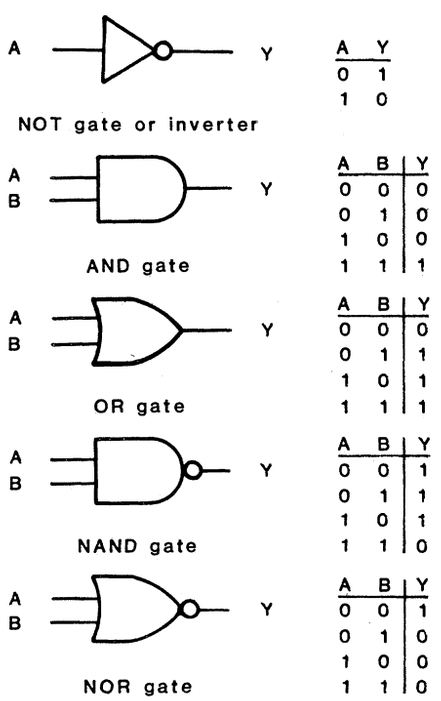

Fig. 1. Names, symbols, and truth tables for some simple logic gates and their corresponding logical operations.

General Design Procedure

The mask set that begins the fabrication process is the goal of the design process. To design an integrated circuit means to transform a functional specification into masks that are ready for fabrication.

Integrated circuit design can be partitioned into two major tasks: logic specification, in which the goal is a logic diagram that accurately represents the desired electronic function, and physical specification, in which the goal is an exact description of the physical locations of all circuit elements and their interconnections on the chip. The design tasks are carried out by continuously iterating between synthesis, the creative act of constructing a given part of a design, and verification, determining whether or not the design will perform according to specifications. That is, a designer creates part of a design, verifies it, uses the results of the verification to modify or correct (recreate) the design, reverifies it, and so on until, much later, the entire design is completely synthesized and verified.

Often a hierarchical approach is taken in which a total design is decomposed into several simpler, functional modules, each of which may be broken into submodules, and so on until the submodules are simple enough to implement. (Hierarchy has the effect of reducing a large, unsolvable problem to several smaller, solvable problems.) Figure 2 illustrates hierarchical levels of an integrated circuit. At the highest level, the circuit itself can be partitioned into functional modules. At lower levels, each functional module can be decomposed into logic gates which, in turn, are represented by interconnections of transistors. At the lowest level, the transistors and their interconnections are described by physical structures.

At each level, each submodule and its interconnections with other submodules must be specified. Both synthesis and verification are performed at all levels, and verification is always needed when going from one level to another in either direction in the hierarchy.

In the era of SSI, a circuit could be verified by physically constructing it from discrete circuit elements on a circuit board and submitting it to exhaustive electronic testing. A major problem was that the discrete circuit elements often introduced extraneous electrical effects not reflective of the integrated circuit. With increased levels of integration the circuit board approach soon became impractical, and it is in this

verification category that much of the existing CAD tools have been created. In particular, simulation, in which the operation and performance of a circuit or system is predicted by a computer program, is now widely used by designers.

Figure 3 presents an overview of the design process, fabrication, and testing. Steps (*a*) through (*h*) constitute the tasks necessary for logic specification; normally many iterations are needed before a successful first-pass specification is obtained. At (*a*) a functional conception of the desired system is generated (synthesis). As a first approach to verification, a functional or system-level simulator (*b*) can check for satisfactory communication among functional blocks in a total system. Next, at (*c*) each block is expanded into smaller modules that go through a complex cycle (*r*) of synthesis and verification and can be added to a library (*d*) for reuse in future designs. A part of the designer's input is a net list, which specifies the modules and how they are connected. These modules can be retrieved from the library (*d*) or newly constructed if they do not already exist. At (*e*) a logic simulator can check the design for correct function (verification); if the verification fails, then the designer modifies the specification [back to (*c*) or (*a*)]. Next, a timing simulator (*f*) can determine whether the performance speed requirements of the circuit will be satisfied and can detect the critical paths (the paths whose speed limits the performance speed of the chip). Timing simulators also indicate whether expected delays of the circuit elements will cause undesirable circuit behavior, such as racing (an error that occurs when a device receives conflicting inputs at nearly the same time). The design may again be modified until acceptable results are obtained.

An activity usually performed in parallel with the design is the creation of a set of test vectors (*g*) that will be used to test the chip after fabrication. Test vectors are sets of values that, when applied to the inputs of an integrated circuit, generate outputs whose values are known for a properly working circuit. An ideal set of test vectors thoroughly exercises all parts of the chip and detects all faulty circuit elements. At (*e*) an initial set of test vectors was created to be used for the logic simulation; usually the results of the simulation indicate changes to be made in the design and in the test vectors. The modified set of test vectors is used next in the timing simulation, and perhaps is modified again. In practice, a set of test vectors covering all possible faults may require too much testing time; a compromise subset of test vectors is often selected, and a fault simulator (*h*) can determine whether the subset is good enough to test the finished device. Often, further design modifications may be needed if fault simulation indicates that some possible key faults cannot be detected because they are buried too deeply in the design.

At (*i*) physical specification begins with layout, which determines the exact locations of the transistors on the chip and specifies a wiring pattern that will interconnect them. This phase of the design involves translating every element and its interconnections into a physical description and assigning it a location. [The average number of transistors that can be manually laid out per person per day lies somewhere between 3 and 40, depending on the regularity of the configuration (*10*).] Next, more verifications (*j*) must be performed to make sure that design rules (rules pertaining to width, length, and spacing imposed on the geometrical features of an integrated circuit by the process technology) and electrical rules (for example, each element must be connected and there must be no shorts between power and ground) are obeyed. Software tools currently available for these tasks are the design rule checker (DRC) and the electrical rule checker (ERC). If the design violates any rules the layout must be modified until all rules are obeyed.

When a satisfactory layout has been obtained, yet another verification is

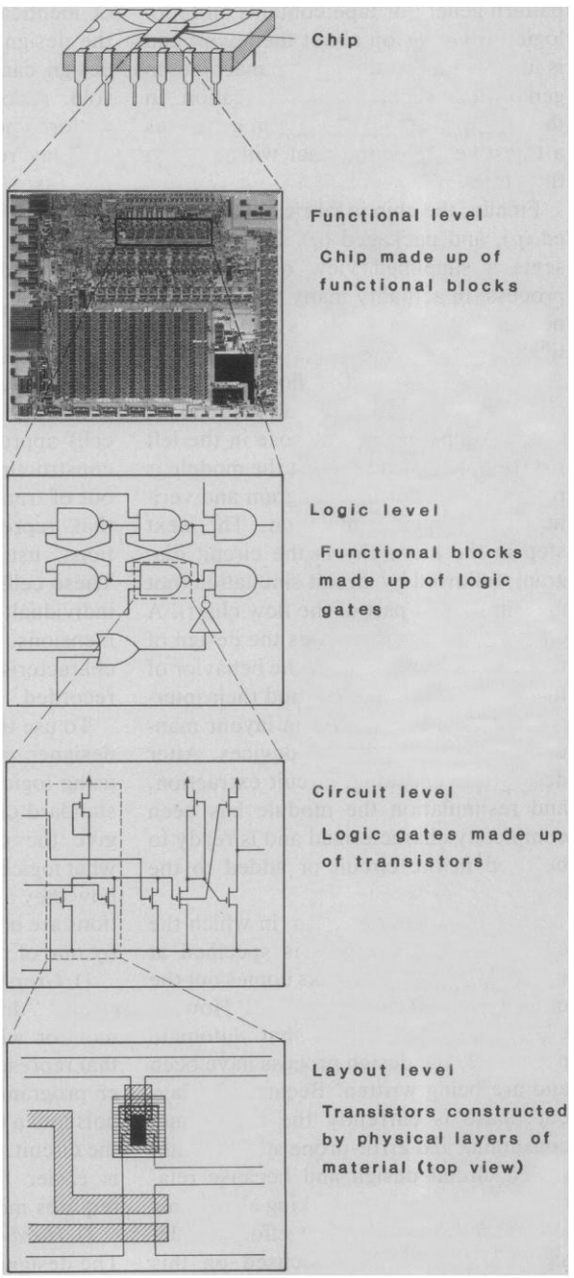

Fig. 2. Hierarchical decomposition of an integrated circuit into functional, logic, transistor, and physical levels.

needed. Up to this point, timing simulations had been based on approximate delays between circuit elements, because their exact locations were not yet known. Exact delays depend on the length and composition of the conductive paths: a signal takes longer to attain its final value through a long polysilicon path than through a short metal one. Now that all locations and path lengths are known exactly, this information is used to extract exact delays (*k*), which are fed back into the timing simulator (*l*). If any timing errors or unacceptable delays are found, the layout or the design must again be modified and the DRC and ERC again used.

When a satisfactory layout is obtained, a program decomposes all geometrical data into rectangles and generates a pattern generator tape (*m*). The pattern generator tape contains all topological information about the layout and is used in a mask-making machine to generate the mask set for fabrication. In the test vector loop a program generates a test vector tape (*n*) that will program the tester.

Finally, the chip is fabricated (*o*), tested (*p*), and packaged (*q*). Figure 3 presents a simplified view of the design process; in actuality many iterations are needed between synthesis and verification.

On the right of the flow chart the characterization of the small modules (*r*) follows steps similar to those in the left part of the flow chart. First the module is represented by a logic diagram and verified with timing simulation. The next step in the hierarchy is the circuit diagram, followed by circuit simulation (not used in the left part of the flow chart). A circuit simulator fine-tunes the design of the module by simulating the behavior of the individual transistors and their interconnections. Next, custom layout manually defines all physical devices. After design rule checking, circuit extraction, and resimulation the module has been completely characterized and is ready to be used in the circuit or added to the library.

Total design automation, in which the desired circuit behavior is specified at one end and a set of masks comes out the other is, as yet, seldom realized. However, computer programs that automate portions of the design process have been and are being written. Because the layout phase is currently the most time-consuming and error-prone stage of integrated circuit design and because relatively simple decision-making algorithms are needed here, the first efforts in design automation have focused on this phase. Existing design automation programs use various structured approaches to the method of design (called the "design methodology").

In a "full-custom" chip, transistors are manually placed one at a time, requiring the full fabrication process for each design (other methodologies to be discussed allow some degree of prefabrication). Although hierarchical design can reduce time and cost to some extent, the full-custom chip has the longest turnaround time and highest cost of any methodology. Still, in some applications this is the methodology of choice. Manual placement by human beings is the best (albeit the slowest) way to achieve the most efficient packing of transistors and, therefore, the smallest size (smaller size results in lower manufacturing cost and higher yields). If a very large number of identical chips are to be fabricated, the design time and cost of full-custom design can be amortized over all chips sold. Also, applications in which high performance (speed and power) is critical may require the full-custom design process.

Established Design Automation Methodologies

At present, only two methodologies enjoy widespread use: standard cells and gate arrays. In the standard cell (or polycell) approach the designer, instead of constructing a circuit "from scratch" out of transistors, uses a library of cells that represent predefined logic functions, usually at the logic gate level. These cells have been constructed from individual transistors, and the cell dimensions, performance, and electrical characteristics have been optimized and recorded.

To use the standard cell approach, the designer first constructs the design by using logic functions represented in the standard cell library. The next step is to give the computer information telling what logic functions are in the design and how they are connected. Design descriptions are entered into a computer system by one of two methods:

1) *Graphics schematic entry*. The designer "draws" the schematic onto a monitor with a set of graphic symbols that represent logic functions. A computer program translates the graphic symbols into a machine-usable description of the circuit. Of the two methods, this one is easier for the designer to use but requires more hardware and software.

2) *Hardware description language*. The designer translates the design into a hardware description language, generating a written description similar to a computer program. This description is in turn translated into a machine-usable circuit description (as in graphics schematic entry). The method requires less hardware and software than graphics schematic entry, but the designer must learn a specialized language. The possibility of creating hard-to-detect discrepancies between the schematic and its description is a major drawback of the method. Also, the danger of introducing errors during design revision has doubled because changes in the design must be made both to the schematic and to the language description.

The circuit is then verified by simulation, using the known, predefined characteristics of the cells in the cell library. The next task is placement, in which cells are laid out in rows with spaces reserved between the rows for wiring channels. Connection points for input and output, called bonding pads, are placed around the periphery of the chip. Cells having many connections in common are placed close to each other, and those that connect to bonding pads are placed near the edge of the chip. In some design automation systems, placement is done automatically by the software; in others, the designer specifies the placement; in still others, software placement is done initially with designer intervention if difficulties or special cases arise.

Most successful placement algorithms are heuristic and use directed forces, vectors representing the direction and distance between interconnected blocks. In most placement algorithms the criteria for success are minimization and uniformization of the crossing count (the number of wires crossing each terminal position in a cell row) and minimization of the combined wire length of all connections (*11*).

Two major groups of placement algorithms are constructive placement and iterative improvement of placement. The constructive placement algorithms include the epitaxial growth algorithm, in which manual placement of a few modules is used to start the process. The algorithm finds the next unplaced module with the maximum number of connections to the placed modules. Then it moves the module into the best available position, finds the next unplaced module with the maximum number of connections, and so on, until all the modules are placed. The best position for a module is found by trying all available positions and minimizing the length of the connections or by placing the module into a zero-force position. [For every module

there is an equilibrium position where the total pull from all other modules is zero. A zero pull is equivalent to the minimum length of the wire for all connected signals (*11*)].

The placement improvement group makes small local changes, such as pairwise exchange of modules, in an attempt to improve placement. After one exchange the crossing count or wire length is recalculated. If the exchange improves the placement it is retained. Some schemes accept some interchanges that worsen the placement in order to improve routability; others accept interactive placement by users.

Following placement comes routing, in which wiring paths among the cells are defined. Routing is done by software that attempts to minimize wire length or follows other optimization criteria. In standard cells the width of each wiring channel is varied to accommodate the wires it contains at the most populated point. In some software systems algorithms are used to iterate between placement and routing to optimize the total wire length and critical path length.

The channel router has been the main routing algorithm for standard cells for many years, and is designed for routing where the points to be connected are in parallel rows. Routes are wired by using horizontal tracks on one layer and vertical tracks on another layer. The variable channel width guarantees that all connections can be made.

In Fig. 4 the layout of a typical standard cell is compared with that of another methodology, the gate array. In the standard cell the width of the wiring channel is variable; wiring is clustered toward the center of the wiring channels, with some wasted space toward the outer ends.

Advantages of the standard cell methodology are:

1) *Rapid design turnaround time.* If everything is done with software, layout of an LSI circuit of around 10,000 transistors may be accomplished in a few months instead of a year or more. Logic specification, layout, optimization, and characterization with regard to delays, drive capability, and loading are all established when the cells are added to the cell library, and these operations do not need to be repeated at the cell level for each design.

2) *Flexibility.* Designers can handle special functions by creating new cells, characterizing them, and adding them to the library. And this flexibility is self-propagating: the larger the library, the greater the flexibility for future designs.

Disadvantages of standard cells are:

1) *Wasted chip area.* The area occupied by the wiring channels can easily exceed 50 percent of the total chip area. Because channel width is variable, the width of a wiring channel must accommodate its greatest requirements. Some designs may turn out to be impossible for the computer to place and route within the area restrictions of the chip. If a great deal of designer intervention is needed, the advantages of automation are lost.

2) *No savings in fabrication time.* Each chip must go through the complete fabrication process.

Fabrication time can be saved by using a programmable array, which contains repeated cells independent of any particular circuit implementation and which can be customized by modifying specific mask layers. Programmable arrays are partially prefabricated chips; that is, large volumes of identical arrays are manufactured and stockpiled. Then, when a designer wishes to implement an integrated circuit, the interconnections for the particular circuit are specified in the final layer or two.

The most common programmable array is the gate array (also called a master-slice or uncommitted logic array), a

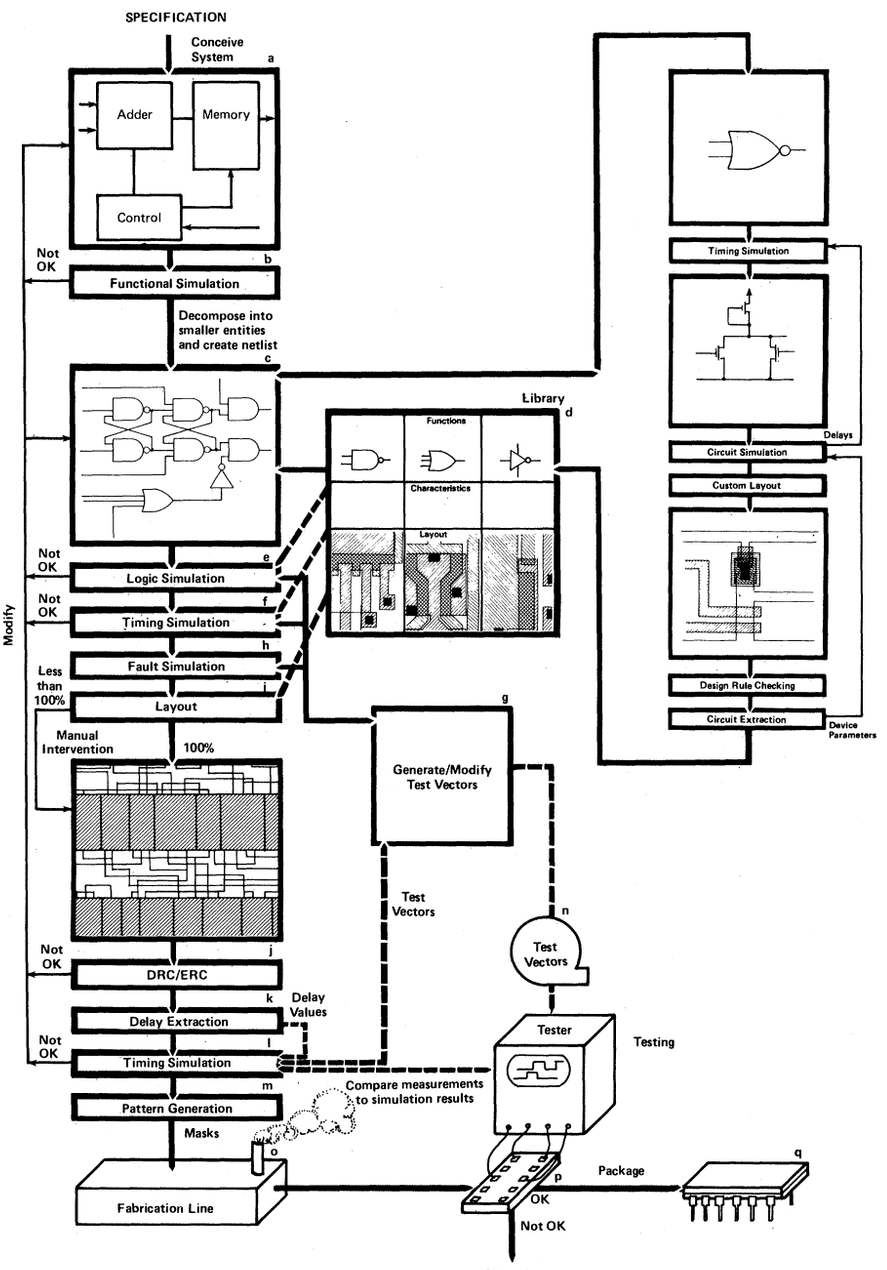

Fig. 3. Design and fabrication of an integrated circuit. A procedural flow is followed that iterates between synthesis and verification. Solid lines mark the flow of the design itself, while dashed lines indicate the transfer of information.

two-dimensional matrix of identical cells, each containing a fixed number (4 to 20) of uncommitted (unconnected) transistors separated by wiring channels (*12*). A circuit is constructed by specifying the interconnections among the transistors within and between cells on the final contact and metallization layers.

From a user standpoint, gate arrays are like standard cells. Gate array designers construct their circuits by using a cell library (sometimes called a macro library) of predefined logic elements, and use graphics schematic entry or a design language to enter the schematic into the computer system. The task of placement is similar to that with standard cells.

Routing of a gate array may be started with a channel router and then "cleaned up" with a Lee or line search router. The Lee (grid expansion) router works on a grid and is based on expanding a wave from one point to another. At each step, grids on a diamond-shaped wave front are expanded one step further, avoiding obstacles and previously used grid points. Each grid point through which the wave passes is marked with a code that stores the direction to the source of the expansion. Once the target point has been reached codes are followed in reverse order to yield the shortest path.

The line search algorithm is gridless and finds a connection through a maze of obstructions. It runs vertical and horizontal expansion lines from the two points to be connected. If any line encounters an obstacle, the router takes a perpendicular path until a line parallel to the original one can pass by the obstacle. Two expanding nets are thus created and the process is terminated when expansion lines from both nets intersect, creating the desired connection. Although this algorithm does not yield the shortest path, it requires substantially less computer memory and runs faster than the Lee algorithm in most cases.

Table 1. Comparison of design methodologies.

Characteristic	Gate array	Standard cells	Full custom
Design time	Short	Short	Long
Fabrication time	Short	Long	Long
Chip area	Large	Intermediate	Small
Cost	Low	Intermediate	High
Versatility	Low	Intermediate	High
Turnaround time for minor redesign	Short	Intermediate	Long

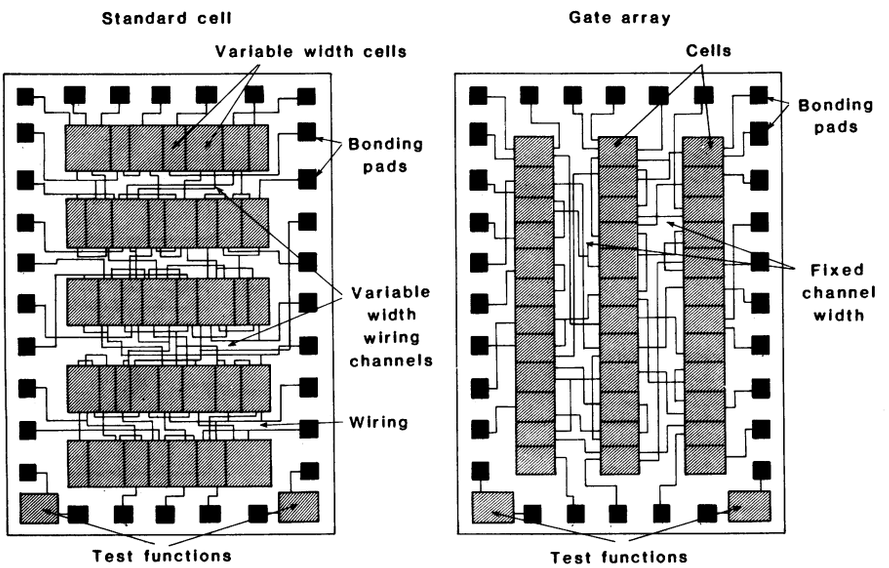

Fig. 4. Comparison of standard cell and gate array methodologies. The standard cell has variable cell and wiring channel width, whereas both are fixed in the gate array.

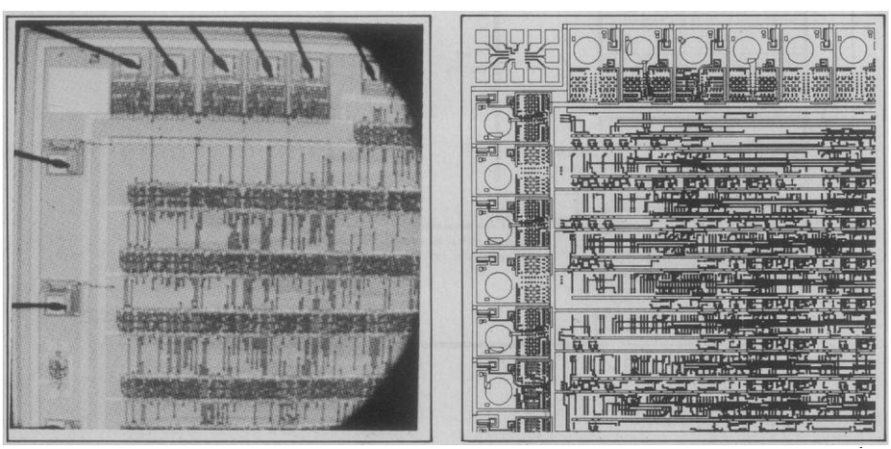

Fig. 5. Completed integrated circuits. On the left is a photomicrograph of a portion of a circuit implemented in standard cell technology. On the right is a computer-generated graphics representation of a routed gate array circuit.

Because of the restrictive wiring capacity of gate arrays, either of the two routers may not always achieve 100 percent routing, and user intervention may be necessary.

One major difference between standard cells and gate arrays is in the construction of the logic elements. Whereas a standard cell logic element is custom-built from individual transistors, a gate array logic element is defined by connecting transistors already contained in array cells. A second major difference has already been mentioned: whereas the width of the routing channels is variable in the standard cell, the array cell's routing channels have a fixed (but not necessarily uniform) width (Fig. 4).

Advantages of the gate array methodology are:

1) *Rapid design turnaround time*. The savings in design time are the same as in the standard cell methodology.

2) *Low cost*. By prefabricating the chips the cost advantages of mass production can be realized for low-volume production.

3) *Short fabrication time*. Since only the final metallization layers need to be made, fabrication time is drastically cut.

Disadvantages of gate arrays are:

1) *Wasted chip area*. Gate arrays typically waste more area than standard cells do because the individual positions of the transistors cannot be optimized; in fact, some transistors may go unused. Also, 10 to 30 percent of the gate array cells may be wasted because wiring channels may run out of space if more than 70 to 90 percent of the available cells are occupied.

2) *Decreased flexibility*. Fewer circuit

functions can be realized than in standard cell or full custom methodologies (for example, analog functions are difficult to implement optimally on a gate array).

3) *Possible wiring restrictions.* Because of the fixed wiring channel width, a particular design may contain too many logic elements to be routable. In that case the recourses are (i) use of a larger array if one is available, (ii) designer intervention and hand-routing of difficult interconnections, (iii) partitioning of the device onto more than one chip, or (iv) reverting to the standard cell methodology.

Figure 5 shows parts of a wired standard cell design and of a gate array. The photograph on the left is of an actual standard cell, while the photograph on the right is of a plot of a gate array obtained with an interactive graphics system. Notice the differences in wiring channel width between the two methodologies and the amount of space devoted to interconnections and bonding pads.

The difficulties of placement and wiring may limit both standard cells and gate arrays to the LSI level of circuit complexity (*3*). On the other hand, the difficulties encountered in the layout of VLSI chips may require that automation be used for large designs. At the present time both the standard cell and gate array approaches provide designers with quicker, less expensive alternatives to full-custom chips. Also, designers in either standard cell or gate array methodology need not be "silicon sophisticates." Table 1 compares the gate array, standard cell, and full-custom design methodologies.

Programmable Logic Arrays

Another programmable array that is established but not as universally applicable as gate arrays is the programmable logic array (PLA). It consists of two rectangular arrays of gates called AND and OR planes. The gates in both planes can be customized by connections in the final metallization layers.

Any desired logic function may be realized by combinations of AND and OR functions and their complements. In a PLA, ANDing is done first by entering the inputs into the AND plane along parallel connections. The results of the AND operations are then entered into the OR plane perpendicularly along parallel connections. The results of the OR operation are output on parallel connections and can be fed back into the AND plane for another set of operations if needed to perform the desired function. Area reduction of the PLA is achieved by logic minimization, folding (permutation and splitting of rows and columns), and partitioning (*13*).

Programmable logic arrays represent almost completely automated design—in areas where they are applicable. To use a PLA the user specifies the logic functions to be implemented. After being processed by logic simplification software, logic equations are used to program the PLA for the desired function. The regular structure of PLA's makes it possible to go directly from the simplified equations to the mask set without placement, routing, or any other intermediate steps. Often the design cycle is shortened further by mapping into a prefabricated PLA structure, thus requiring a single masking step. PLA's, however, have limited applicability; they are poor for many logic functions, especially where timing is critical. PLA's find the most use as specialized parts of other chips (for example, control logic of microprocessors). In addition, field-programmable logic arrays are available that can be customized electrically by the user.

In addition to the global design methodologies already mentioned, several shortcuts are being developed to optimize various steps in the design process.

Design Shortcuts

Symbolic layout. Designers describe transistors and their interconnections and locations in a particular circuit by using predefined symbols on a cathode-ray tube terminal. Once stored, a symbolic layout may be called up and automatically implemented in a particular technology; thus the same design may be adapted to changes within a technology without changing the layout description. Thus, when an existing technology is scaled down (dimensions are reduced by some factor), only a few key design rules need be changed in the stored layout description; a new layout is not required. In one symbolic layout system, designers represent their designs on a floating grid by manipulating shapes and lines, mapped one-to-one with transistors and interconnections (*3*). Compaction programs exist that attempt to condense the layout to improve chip area use.

Bristle blocks. In bristle blocks functional circuits are defined as rectangular modules having specific interconnections ("bristles"). These modules have been predefined and presimulated, and are ready to be connected to one another by "intermeshing" the bristles. The claimed advantage of the bristle block approach is that there are no routing paths needed outside the blocks; all the necessary connections are made automatically at the edges. However, a major limitation is that each block must interact only with its immediate neighbors. Because of this limitation, bristle blocks may find the most use in computer-type chips, where all blocks are organized around a common data path.

Conclusions

Chip design is a long, complicated, and expensive task, and even the smallest error can be fatal to a project. As circuit complexity increases to VLSI, design tasks are becoming astronomically expensive, time-consuming, and error-prone, and manual implementation is impractical or unfeasible. A constraint that design automation will always face is that of solving problems of the next generation with tools of (at best) the current generation.

Design automation remains the hope of VLSI designers for getting their chips to the marketplace in a reasonable amount of time and for obtaining a competitive price. Design automation not only saves money by reducing design and fabrication time, but also helps the community of systems and logic designers to work more effectively and innovatively. A coherent, user-friendly, completely automated system of integrated circuit design that requires little or no human intervention has yet to be realized by many design institutions. Improving and integrating existing design automation software packages and inventing algorithms will continue to occupy industrial and academic institutions for many years to come.

References and Notes

1. C. Mead and G. Lewicki, *Electronics* **55**, 107 (1982).
2. L. M. Rosenberg, *J. Digital Syst.* **5**, 301 (1981).
3. P. Losleben and D. F. Barbe, Eds. *Very Large Scale Integration (VLSI): Fundamentals and Applications* (Springer-Verlag, Berlin, 1980), pp. 89–127.
4. M. Marshall, *Electronics* **53**, 73 (1980).
5. S. Trimberger, *IEEE Spectrum* (June 1982), p. 38.
6. M. Breuer, A. D. Friedman, A. A. Iosupovicz, *Computer* **14**, 58 (1981).
7. T. C. Raymond, *ibid.*, p. 89.
8. J. F. Skalski, M. J. Howes, D. V. Morgan, Eds., *Large Scale Integration Devices, Circuits, and Systems* (Wiley, Chichester, England, 1981), pp. 319–343.
9. S. McMinn, *VLSI Des.* (July/August 1982), p. 16.
10. L. Lopp, *Lambda* (Second Quarter, 1981), p. 52.
11. J. Soukup, *Proc. IEEE* **69**, 1281 (1981).
12. A. R. Newton, *ibid.*, p. 1189.
13. G. D. Hachtel, A. R. Newton, A. L. Sangiovanni-Vincentelli, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **1**, 63 (1982).