Computer-aided design of LSI circuits

C. Niessen

Introduction

In the design of LSI circuits the computer is an indispensable aid; the calculation of such a design in the conventional way 'by hand' would be a practical impossibility. For some stages of the design process special computer programs have been developed that drastically reduce the amount of work involved and minimize the chance of errors. The essential feature here is that the computer does not take over from the designer, but becomes a partner — hence the name 'computer-aided design' (CAD). In this article we shall look at a number of such programs that have been developed at Philips; since LSI techniques are mainly used for digital circuits, the emphasis here will be on the computer-aided design of digital LSI circuits.

There are several stages in the process of developing an LSI circuit (fig. 1). First, starting from a specification of the function which the circuit is required to fulfil, a circuit design is made. This is followed by a verification procedure to determine whether the circuit does in fact represent the desired function; any corrections required can be made at this stage. Next, a test procedure is devised that can be used after manufacture to make sure that the end-product is functioning properly. This is done at such an early stage because there is no certainty that all the possible faults can be found by testing. If the test procedure reveals that the circuit cannot be fully tested, the design must again be modified, and this of course has to be followed by a further verification procedure. Once a design has been produced in this way that meets all the specifications, the layout of the circuit is drawn and the photomasks for the manufacture of the IC can be made. The next stage is the actual production of the IC. Finally, the circuits from the production line are put through the test procedure. The stages of the development process for which CAD programs are now available include design verification, test preparation, layout making and mask making.

Review of CAD

During the design verification the operation of the circuit is simulated by computer. Two distinct procedures, for which various programs are available, can be used here: these are circuit analysis and logic simulation. In circuit analysis the 'analog behaviour' of the circuit is investigated; the principal currents and voltages are treated as continuous variables and their waveforms are calculated. The application of circuit analysis is confined to circuits containing no more than a few hundred components. For larger circuits the memory capacity and computer time required for these calculations would be far too large, and the designer would also find it hard to retain a general picture of the design because of the many details.

Larger circuits can only be calculated by splitting them into subcircuits. To do this two conditions have to be met: it must be possible to calculate the subcircuits without any knowledge of the total system, and it must be possible to join the subcircuits together without detailed knowledge about each of them. These

![Diagram of the various stages in the design and manufacture of an LSI circuit. Computer programs are available that the designer can use to verify his design, to prepare tests and to design the layout of the masks for the manufacture of the IC.](image-url)
The next step in the development of an IC is to convert the circuit diagram into a drawing of the various components; the procedure used for doing this is called logic simulation. Large analog circuits can sometimes be divided into a number of analog subcircuits, but usually an analog-to-digital conversion is the only possibility of making the circuit calculations. This is why systems of essentially analog nature are increasingly dependent on the use of digital techniques. In this article we shall therefore be concerned with digital circuits only.

In logic simulation the truth table or Boolean equation is checked to verify whether the circuit will perform the appropriate logic function in practical conditions. A circuit analysis must be carried out beforehand to check the analog behaviour of the basic circuits, so that it is known under what conditions these modules will continue to function as digital circuits; see fig. 2 and caption. If these requirements are taken into account in the design procedure, the circuit as a whole need only be verified with the aid of logic simulation; no further analysis at the level of currents and voltages will then be necessary.

Testing an integrated circuit is much more difficult than testing a circuit composed of discrete components. The discrete components have already been tested, of course, and there are also many internal measuring points available for testing the circuit. In testing an integrated circuit, on the other hand, it has to be assumed that any component may be faulty; the faults that may occur during manufacture, for example due to mask damage, are completely unpredictable. Moreover, internal points in an IC are not accessible, so that faults can only be detected through the signals that appear at the circuit outputs. In designing a test for an integrated circuit it is therefore necessary to ensure that a combination of input voltages can be found such that any fault will introduce a change in state that can be observed at the output (fig. 3). If it is not found possible to arrive at a set of test patterns capable of detecting an acceptable percentage of the possible faults, the only alternative is to change the design. Since it is difficult for a designer to think up such tests and almost impossible for him to assess their quality, various programs have been developed to simplify matters for him [1].

The requirements are fully satisfied by digital circuits, composed of the basic AND, OR, NAND and NOR logic circuits. With circuits of this type it is therefore possible to calculate designs containing many thousands of components; the procedure used for doing this is called logic simulation. Large analog circuits can sometimes be divided into a number of analog subcircuits, but usually an analog-to-digital conversion is the only possibility of making the circuit calculations. This is why systems of essentially analog nature are increasingly dependent on the use of digital techniques. In this article we shall therefore be concerned with digital circuits only.

In logic simulation the truth table or Boolean equation is checked to verify whether the circuit will perform the appropriate logic function in practical conditions. A circuit analysis must be carried out beforehand to check the analog behaviour of the basic circuits, so that it is known under what conditions these modules will continue to function as digital circuits; see fig. 2 and caption. If these requirements are taken into account in the design procedure, the circuit as a whole need only be verified with the aid of logic simulation; no further analysis at the level of currents and voltages will then be necessary.

Testing an integrated circuit is much more difficult than testing a circuit composed of discrete components. The discrete components have already been tested, of course, and there are also many internal measuring points available for testing the circuit. In testing an integrated circuit, on the other hand, it has to be assumed that any component may be faulty; the faults that may occur during manufacture, for example due to mask damage, are completely unpredictable. Moreover, internal points in an IC are not accessible, so that faults can only be detected through the signals that appear at the circuit outputs. In designing a test for an integrated circuit it is therefore necessary to ensure that a combination of input voltages can be found such that any fault will introduce a change in state that can be observed at the output (fig. 3). If it is not found possible to arrive at a set of test patterns capable of detecting an acceptable percentage of the possible faults, the only alternative is to change the design. Since it is difficult for a designer to think up such tests and almost impossible for him to assess their quality, various programs have been developed to simplify matters for him [1].

The requirements are fully satisfied by digital circuits, composed of the basic AND, OR, NAND and NOR logic circuits. With circuits of this type it is therefore possible to calculate designs containing many thousands of components; the procedure used for doing this is called logic simulation. Large analog circuits can sometimes be divided into a number of analog subcircuits, but usually an analog-to-digital conversion is the only possibility of making the circuit calculations. This is why systems of essentially analog nature are increasingly dependent on the use of digital techniques. In this article we shall therefore be concerned with digital circuits only.

In logic simulation the truth table or Boolean equation is checked to verify whether the circuit will perform the appropriate logic function in practical conditions. A circuit analysis must be carried out beforehand to check the analog behaviour of the basic circuits, so that it is known under what conditions these modules will continue to function as digital circuits; see fig. 2 and caption. If these requirements are taken into account in the design procedure, the circuit as a whole need only be verified with the aid of logic simulation; no further analysis at the level of currents and voltages will then be necessary.

Testing an integrated circuit is much more difficult than testing a circuit composed of discrete components. The discrete components have already been tested, of course, and there are also many internal measuring points available for testing the circuit. In testing an integrated circuit, on the other hand, it has to be assumed that any component may be faulty; the faults that may occur during manufacture, for example due to mask damage, are completely unpredictable. Moreover, internal points in an IC are not accessible, so that faults can only be detected through the signals that appear at the circuit outputs. In designing a test for an integrated circuit it is therefore necessary to ensure that a combination of input voltages can be found such that any fault will introduce a change in state that can be observed at the output (fig. 3). If it is not found possible to arrive at a set of test patterns capable of detecting an acceptable percentage of the possible faults, the only alternative is to change the design. Since it is difficult for a designer to think up such tests and almost impossible for him to assess their quality, various programs have been developed to simplify matters for him [1].

The requirements are fully satisfied by digital circuits, composed of the basic AND, OR, NAND and NOR logic circuits. With circuits of this type it is therefore possible to calculate designs containing many thousands of components; the procedure used for doing this is called logic simulation. Large analog circuits can sometimes be divided into a number of analog subcircuits, but usually an analog-to-digital conversion is the only possibility of making the circuit calculations. This is why systems of essentially analog nature are increasingly dependent on the use of digital techniques. In this article we shall therefore be concerned with digital circuits only.

In logic simulation the truth table or Boolean equation is checked to verify whether the circuit will perform the appropriate logic function in practical conditions. A circuit analysis must be carried out beforehand to check the analog behaviour of the basic circuits, so that it is known under what conditions these modules will continue to function as digital circuits; see fig. 2 and caption. If these requirements are taken into account in the design procedure, the circuit as a whole need only be verified with the aid of logic simulation; no further analysis at the level of currents and voltages will then be necessary.

Testing an integrated circuit is much more difficult than testing a circuit composed of discrete components. The discrete components have already been tested, of course, and there are also many internal measuring points available for testing the circuit. In testing an integrated circuit, on the other hand, it has to be assumed that any component may be faulty; the faults that may occur during manufacture, for example due to mask damage, are completely unpredictable. Moreover, internal points in an IC are not accessible, so that faults can only be detected through the signals that appear at the circuit outputs. In designing a test for an integrated circuit it is therefore necessary to ensure that a combination of input voltages can be found such that any fault will introduce a change in state that can be observed at the output (fig. 3). If it is not found possible to arrive at a set of test patterns capable of detecting an acceptable percentage of the possible faults, the only alternative is to change the design. Since it is difficult for a designer to think up such tests and almost impossible for him to assess their quality, various programs have been developed to simplify matters for him [1].

The requirements are fully satisfied by digital circuits, composed of the basic AND, OR, NAND and NOR logic circuits. With circuits of this type it is therefore possible to calculate designs containing many thousands of components; the procedure used for doing this is called logic simulation. Large analog circuits can sometimes be divided into a number of analog subcircuits, but usually an analog-to-digital conversion is the only possibility of making the circuit calculations. This is why systems of essentially analog nature are increasingly dependent on the use of digital techniques. In this article we shall therefore be concerned with digital circuits only.

In logic simulation the truth table or Boolean equation is checked to verify whether the circuit will perform the appropriate logic function in practical conditions. A circuit analysis must be carried out beforehand to check the analog behaviour of the basic circuits, so that it is known under what conditions these modules will continue to function as digital circuits; see fig. 2 and caption. If these requirements are taken into account in the design procedure, the circuit as a whole need only be verified with the aid of logic simulation; no further analysis at the level of currents and voltages will then be necessary.

Testing an integrated circuit is much more difficult than testing a circuit composed of discrete components. The discrete components have already been tested, of course, and there are also many internal measuring points available for testing the circuit. In testing an integrated circuit, on the other hand, it has to be assumed that any component may be faulty; the faults that may occur during manufacture, for example due to mask damage, are completely unpredictable. Moreover, internal points in an IC are not accessible, so that faults can only be detected through the signals that appear at the circuit outputs. In designing a test for an integrated circuit it is therefore necessary to ensure that a combination of input voltages can be found such that any fault will introduce a change in state that can be observed at the output (fig. 3). If it is not found possible to arrive at a set of test patterns capable of detecting an acceptable percentage of the possible faults, the only alternative is to change the design. Since it is difficult for a designer to think up such tests and almost impossible for him to assess their quality, various programs have been developed to simplify matters for him [1].

The requirements are fully satisfied by digital circuits, composed of the basic AND, OR, NAND and NOR logic circuits. With circuits of this type it is therefore possible to calculate designs containing many thousands of components; the procedure used for doing this is called logic simulation. Large analog circuits can sometimes be divided into a number of analog subcircuits, but usually an analog-to-digital conversion is the only possibility of making the circuit calculations. This is why systems of essentially analog nature are increasingly dependent on the use of digital techniques. In this article we shall therefore be concerned with digital circuits only.
verification drawings for checking whether certain general design rules have been properly observed [2]. After these checks have satisfied the designer that the layout is good and that the description in the computer language is correct, the computer can then supply a punched tape for drawing the photomasks on a numerically controlled drawing machine [3]. The punched tape can also be directly used for controlling an optical pattern generator, such as the 'Optyco-graph' [4].

The CAD facilities so far mentioned may be described as passive: the designer makes a design, the computer verifies it and finds out whether the circuit can be tested; the designer makes a layout, the computer makes the necessary checks and produces the control tapes. If the creative steps in the design process are also to be automated, it will be necessary to make programs that will enable the computer to rival the designer's knowledge, inventiveness and repertoire of alternatives. This does not seem to be a practical possibility for the electrical and logic design, but the layout design is something that can be entrusted to the computer.

The programs that have been developed for layout design result in a layout that has a more regular structure than a conventionally produced one, and hence a lower packing density for the components on the chip (the 'layout efficiency'). Whereas optimum layout efficiency was a necessary condition for achieving an acceptable yield in the early days of integrated circuits, nowadays the technological advances made (smaller components, larger chips) allow efficiency to be traded for faster and cheaper layout design. This is particularly important for small production runs, where the production costs are low compared with the design costs. With circuits for mass production, the production costs will be the main consideration, and it will be worth going to the expense of additional development work to reduce the production costs. The layout will then be conventionally designed. An intermediate form, where the layout is partly made by computer and partly by conventional methods, is also occasionally used.

The CAD programs mentioned above, each with their own input language, each deal with an individual aspect of the complete LSI design. A consequence of this is that a designer has to make different design descriptions for one particular problem, e.g. one for the logic simulation and another for the layout. This demands a great deal of work from the designer and also increases the risk of errors in the descriptions. In some cases, for example for LOC/MOS [5] and 1PL [6], it has therefore been useful to develop programs for linking the various design phases. If these programs are unified ('integrated') in such a way that they are all based on the same circuit description (the 'source information')
and always refer back to the result of a previous calculation, we have a CAD system.

A number of programs developed at Philips for the various design stages will now be discussed. The program functions will be described and where possible explained with the aid of examples, without going into many of the details required for computer processing. At the end of the article two integrated CAD systems will be described.

Circuit analysis

To calculate the analog behaviour of a circuit, the computer must be supplied with the circuit diagram and at the same time given an instruction describing the nature of the investigation to be carried out, e.g. the determination of the frequency response of an amplifier. Generating a diagram in computer language presents no problems with the interconnections between the components, but the components themselves cannot usually be described by a few simple mathematical equations, because of effects such as parasitic capacitances and leakages. In the documentation of a transistor, for example, this problem is solved by using graphs from which the principal analyses relate simulated experiments that would be difficult if not impossible to carry out.

The circuit-analysis procedure can also include printer. The circuit-analysis procedure can also include generating a diagram in computer language presents no problems with the interconnections between the components, but the components themselves cannot usually be described by a few simple mathematical equations, because of effects such as parasitic capacitances and leakages. In the documentation of a transistor, for example, this problem is solved by using graphs from which the principal analyses relate simulated experiments that would be difficult if not impossible to carry out.

A circuit-analysis program takes the input information and uses it to enter into the computer memory a set of equations that are an equivalent of the circuit. The calculations required are then made from these equations. For example, it is possible to calculate the voltages at the various nodes of the circuit when the supply voltage has been applied. The response to input signals of various kinds can also be calculated, and the voltage waveform can be displayed on a cathode-ray tube screen or an x-y plotter, or printed out on a line printer. The circuit-analysis procedure can also include simulated experiments that would be difficult if not impossible to carry out on the actual circuit; for instance, the gain of a transistor can be varied so as to find its optimum value.

A circuit-analysis program widely used at Philips is PHILPAC (PHILips Package for Analysis of Circuits). This can be used for analyses of various types on both bipolar and MOS circuits. The principal analyses relate to d.c. and a.c. behaviour and to transient effects, such as switching transients and pulse responses. It is also possible to carry out various statistical calculations with this program. In the development of PHILPAC much attention was paid to the convenience of the user. The descriptive language for the circuits is simple and its terminology is closely related to that used by the designer. For instance, the value of a resistance of 2700 ohms is fed in as 2K7. Along with the basic elements such as resistances, voltage and current generators, etc., the program has an extensive 'library' with models of commonly used transistors and diodes. The designer can also define his own models if he wishes.

As an example the input and output will be shown for a PHILPAC transient analysis of the amplifier stage of fig. 5. To describe the circuit it is first of all necessary to number the nodes of the network. Fig. 6 shows the network description in the PHILPAC coding. For every element of the circuit this contains an instruction consisting of a code for the kind of element, its name, a list of the nodes to which it is connected and the value of the element. Thus, the instruction 'R2 (2,0) 3K9' means: 'a resistance with the name R9, connected between the nodes 2 and 0, and with a value of 3900 ohms' (each instruction ends with the symbol $)$. The code 'BC 108-IX' as the 'value' of the transistor $TN_{1}$ in fig. 6 is a reference to the model of this transistor in the PHILPAC library. The last instruction in fig. 6 defines the current generator $J_{1}$; the code 'SINSQ' here refers to a standard function from the PHILPAC library, a pulse train whose repetition frequency, amplitude, rise time, length and decay time must be specified in the expression (in fig. 6 it relates to a current pulse with an amplitude of 0.1 milliamperes and a total duration of $7 + 36 + 7 \mu s$).

In addition to the network description a calculation instruction must also be fed in; fig. 7 shows this instruction for the analysis of the amplifier stage. The request is for a transient analysis of the response to the pulse signal from the current generator $J_{1}$, during a 60-$\mu$s period and calculated in 40 steps (the period of 60 $\mu$s corresponds to slightly more than one pulse). It also states that the output should consist of graphs of the voltage waveform at the nodes 1 and 4, the input and the output.

Fig. 8. Results of the analysis of fig. 7, produced by a line printer. The graphs indicate the voltage variation at the nodes 1 (a) and 4 (b) of fig. 5 during the period in which the current generator $J_t$ delivers one pulse. The time is plotted vertically in 40 steps of 1.5 $\mu$s; the voltages are plotted along the horizontal axis in millivolts. The voltage pulse of (b) is the circuit response to a voltage pulse with the shape of (a) at the input.
After the data have been read in, the computer carries out some checks for faults. It examines, for example, whether all the resistances between two nodes have been connected. The actual analysis then follows, and then the output is printed out on a line printer. Fig. 8 shows the input and output pulse shapes calculated in this way. The designer can now examine these results to find out whether the circuit meets his requirements and expectations.

The example given demonstrates only a few of PHILPAC's capabilities. There is insufficient space here for a full discussion of all the other facilities it offers. One particular facility deserves a brief mention, however. When a circuit such as that of fig. 5 is designed as an IC, the various components will exhibit deviations from the desired values. A designer obviously wants to know the extent to which his circuit is sensitive to variations of this kind. He can find this out by making a few statistical calculations with PHILPAC. These can be made by adding to the circuit description some data on the spread in the component values; the computer can then calculate, e.g. by means of a Monte-Carlo analysis, their effect on the behaviour of the circuit. (A Monte-Carlo analysis is a series of analyses in which different sets of random component values are chosen within the range of spread.)

Logic simulation

Verification of the design of digital circuits, as discussed in the introduction, takes place in two phases. First, it is necessary to investigate whether the basic units are working properly; since this concerns the analog behaviour of the circuits, this is done with a circuit-analysis program. This analysis also determines the conditions under which the abstraction of analog voltages to the logic levels '0' and '1' remains valid (e.g. the 'fan-out'; see fig. 2 and caption). If these conditions are taken into account when the logic circuit is designed, the complete circuit only needs to be simulated for proper functioning at the logic level. This verification of logic behaviour is done by means of logic simulation; in this process the computer investigates systematically the response of the circuit to the combinations of input signals devised by the designer.

The program developed at Philips for performing the logic simulation is known as PHILSIM (PHIlips Logic SIMulator). As an example of how the program works the simulation of a 1-bit adder will be described here. A circuit of this type, usually referred to as a 'full adder', a single section of an adder, for example, in the arithmetic unit of a computer, receives as its input signals two binary numbers \( A \) and \( B \) and an incoming carry \( C_{in} \) from the preceding section. The circuit is required to produce the sum \( S \) of these, and also the outgoing carry \( C_{out} \) to the next section. This logic behaviour is described in the truth table in Table I.

To design a circuit that works in accordance with the truth table, we first derive from the table the Boolean equations for \( S \) and \( C_{out} \):

\[
S = C_{in} \cdot (\bar{A} \cdot B + A \cdot B) + C_{in} \cdot (\bar{A} \cdot B + A \cdot \bar{B}),
\]

(1)

and

\[
C_{out} = A \cdot B + C_{in} \cdot A + C_{in} \cdot B.
\]

(2)

The operator '•' here denotes the AND function and the operator '+' denotes the OR function; the operator '·' has priority over '+'. Using the rules of switching algebra, we can derive a simpler expression for \( S \) from equations (1) and (2):

\[
S = C_{in} \cdot A \cdot B + C_{out} \cdot A + C_{out} \cdot B + C_{in} \cdot S.
\]

(3)

Equations (2) and (3) indicate directly the composition of the network of AND, NAND and NOR gates that will give \( S \) and \( C_{out} \) from the signals \( A \), \( B \) and \( C_{in} \); see fig. 9.

![Fig. 9. Diagram of a 1-bit adder (full adder), used as a section of a larger adder. This circuit takes the binary numbers \( A \) and \( B \) and the incoming carry \( C_{in} \) to form the sum \( S \) and the outgoing carry \( C_{out} \).](image-url)
In the PHILSIM program the input, as in PHILPAC, consists of two parts: the network description and the simulation instruction. Fig. 10 shows the description of the network in fig. 9. It consists of a series of instructions each of which contain the data for one gate: the output signal, the function and the input signals, in that order. For this description a number of internal signals of the circuit are designated by $H_1$ to $H_7$.

Although the description in fig. 10 does not mention gate delay times, these are nevertheless taken into account in the simulation. The PHILSIM program also digitizes the time; in the simulation the state of the circuit is calculated from the time zero and after every successive interval of one time unit. For each simulation the designer can assign any value he wishes to this time unit. This implies that events in the network (e.g. changes in input signals) can only take place at integral multiples of this time unit. The gate delay times must also amount to an integral number of time units; this number can be indicated beside each gate in the network description. When, as in fig. 10, no gate delay time is indicated, PHILSIM automatically assumes a delay time of one time unit per gate.

Fig. 11 shows the second part of the input, the simulation instruction. Lines 1 and 2 contain a few indications concerning the output; the subsequent combinations of instructions 'IT' ('initialize to') and 'SU' ('simulate until') give the actual simulation instruction. Thus, line 3 means 'set the time to zero and assign to $A$, $B$ and $Cin$ the value '0'; and line 4 means 'simulate, starting from this situation, the passage of 10 time units'. In the following lines all the possible combinations of input signals are run through systematically. The simulation time required for this network is easily determined: it follows from fig. 9 that forming the sum signal requires a maximum of five time units, i.e. two for forming $Cout$ and three for determining $S$. A simulation time of 10 units is thus amply sufficient for all gates to return to the steady state, even in the event of errors in the design or in the network description.

The result of the simulation is printed out in the form shown in fig. 12. The left-hand column gives the time axis, on which the time is reset to zero at each 'IT' instruction. In the adjacent columns the calculated values of $A$, $B$, $Cin$, $S$ and $Cout$ are printed out, but only at the
times zero and at the times when there is a change in one of these values. An asterisk indicates that the state of the associated output is not known at that moment; as a consequence of the gate delay times the effect of the input signals has not yet made itself felt. As can be seen in fig. 12, first a C_{out} appears, immediately followed by S. Comparing this result with the truth table in Table I, we see that there is complete agreement, so that the simulation shows that the circuit correctly represents the desired function.

In practice the simulation will have to extend over many more time units than in our example. A simulation of a period of 10 milliseconds of 'real time', with a time unit of 10 ns, would require calculations for 1 000 000 time steps, which would require far too much computer time. This difficulty can be overcome to some extent by using 'event simulation'. This kind of simulation is not controlled by times but by events: the circuit is not calculated after every time unit but only at the times when a signal changes state. The consequences of such a change are then administered as future transactions, and the simulation steps from event to event, under the control of this administration, missing out the times when the circuit is in the steady state.

A further simplification that can shorten the simulation time considerably is to change over from an asynchronous to a synchronous circuit. Let us assume that the adder in our example derives its input signals from a logic circuit L_1 (see fig. 13a) and passes on the output signals to a logic circuit L_2. In the simulation of circuit L_2 account must then be taken of the delays both in the circuit L_1 and in the adder, because they affect the moment at which the signals S and C_{out} become available. This is no longer necessary when synchronization registers are included between the circuits, as shown in fig. 13b. At an instruction from a synchronization pulse, each of these registers takes over the output signals from the preceding circuit and holds them until the next pulse becomes available in the form of input signals from the following circuit. In the intervening period each circuit can then perform its function independently, without being disturbed by the delay effects of the preceding circuits. If the period between the pulses — which is usually constant, hence the term 'clock pulses' — is made long enough to ensure that all gates in the various circuits can return to the steady state during that period, there is no longer any need for the simulation to take the gate delay times into account. Only one calculation per clock pulse then has to be performed for each circuit, and this is much fewer than in the asynchronous case of fig. 13a. Synchronous circuits can thus be simulated up to several hundred times faster than asynchronous ones.

The 1-bit adder dealt with here will in practice be part of, say, a four-bit adder, and this in turn will be part of a larger system. Since it is convenient for the designer if this structure is expressed in the network description for the simulation, the PHILSIM program is provided with a facility for representing such subcircuits by a single instruction. This consists of the name of the circuit, the input signals and the output signals. The circuit is thus regarded as a 'black box', designated in the PHILSIM code by 'macro'; the contents of a macro are specified in a separate description.

**Layout**

The IC technology used (e.g. MOS, LOCMOS, I^2L) has a considerable influence on the way the layout is arranged. Thus, I^2L gates are situated on either side of the injector, the conductor that provides the current supply. In dynamic MOS circuits the gates are arranged in rows to avoid loss of chip area because of bends in the clock lines. Again, the arrangement with a two-layer wiring system will be different from that for single-layer wiring.

So far it has not been found possible to translate the craftsmanship required for making good layouts into...
computer strategies that produce equally compact layouts. In spite of the availability of computer programs, layouts are still frequently made in the conventional way. However, programs have been developed to take over the routine work such as the repetition of patterns and checking the layout. DRAW [7] and CIRCUIT-MASK [8] are examples of such programs. The descriptive language CIRCUITMASK is used for defining all the geometric elements that can be used to form a layout. A number of basic figures are available for this; the most important ones are rectangles and polygons. Circular elements are also available for special purposes. A transistor could be described in CIRCUITMASK as shown in fig. 14. This example uses the instruction ‘RR’ for rectangles and ‘PATH’ for the U-shaped interconnection tracks of base and collector.

The correctness of the layout can be checked with the aid of a drawing made on a computer-controlled drawing machine; see fig. 15. This can also be done with a graphical display, i.e. a cathode-ray tube connected to the computer; see fig. 16. The designer can then immediately correct any faults with a keyboard and a light pen; he does not have to wait for drawings.

In visual inspection there is always a chance that one or more faults may be overlooked. For this reason various automatic checking methods have been developed that make it possible to determine whether the layout design rules have been observed. For example, the computer can be made to check whether all the interconnection tracks have a specified minimum width and a minimum spacing. The instruction ‘CHECK GAPS (15,10 IN)’ means ‘check whether all the tracks in the mask IN (IN is the mask for interconnection tracks) are at least 15 \( \mu \text{m} \) wide and have a spacing of at least 10 \( \mu \text{m} \).’ The faults found by the program can be shown on the graphical display or printed out in the form of coordinate tables by the line printer.

If, after the checks have been completed, the designer finds that the layout is correct, he will then have a control tape made for the mask machine. Since this involves the use of costly precision equipment, the CIRCUITMASK program will arrange the information on the tape in such a way as to minimize the time taken by the mask machine.

Programs that make layouts by automatic methods are nearly always specifically designed for a particular technology. Although all these programs broadly follow the same strategy, differences in technology nevertheless rule out any universal solution. The operations performed by such a program, on the basis of the logic description of the circuit, can be divided into three groups:

- partitioning of the circuit into a number of modules, and broadly allocating the modules a position in the layout;
- positioning the components in the modules, and
- interconnecting the components as required by the logic description.

Fig. 14. Description of a transistor in CIRCUITMASK code. The first instruction assigns to this pattern the name ‘TRANS’. The lines numbered 1 to 7 contain the description. Each line begins with an indication of the type of figure (‘RR’ is rectangle, ‘PATH’ is interconnection track), followed by an indication of the masks these figures have to be placed in (e.g. ‘DP’ means deep P, the isolation diffusion) and also the coordinates where the figures have to be positioned. The final instruction completes the definition of the pattern ‘TRANS’.

Fig. 15. Drawing of the ‘TRANS’ pattern made on a computer-controlled drawing machine (plotter) after processing by CIRCUITMASK.
The partitioning of the circuit is the main and most difficult phase in designing the layout; the aim is to substitute a number of simple problems for the actual problems to be solved. Good partitioning is obtained when the modules have high internal cohesion but few interconnections between modules. A poor choice made during partitioning will result in a large layout, and this makes it important for the designer to be able to intervene at this stage. In positioning the components in the modules the usual tactic is to arrange the components in rows. As the size of the modules is limited simple iteration procedures can therefore be used. Since the aim is to minimize the area of the circuit, the objective in such a search process could for example be the minimization of the thickness of the interconnection channels.

After positioning of the components the interconnections have to be made. With multilayer wiring this is not so very difficult, but with a single-layer wiring the computer may often only find a partial solution. The designer will then have to do the rest of the work himself.

This account of the mechanical design of a layout may give the impression that there are three autonomous stages. It might seem that it would be sensible to return to a previous stage if it appeared later that a poor choice had been made. In conventional circuit design such interaction does in fact exist between partitioning, positioning and the making of interconnections. In spite of the advantages of such interaction, it is nevertheless seldom used when working with a computer; the computer time required would usually be far too long.

Testing

So that the ICs can be tested after manufacture it is necessary to design a test procedure that will enable all internal faults that can occur in the IC to be measured at a circuit output. The first step in drawing up such a test procedure is therefore to identify all the faults that can possibly occur. For each fault a test pattern must then be produced that will enable the fault to be detected at one of the outputs when the pattern is presented to the inputs. The actual faults in a circuit—breaks, short-circuits, faulty areas—are too complicated to be handled by a computer program. A sim-

---

Fig. 16. Design and correction of a layout with a graphical display. The light pen is used for indicating and moving figures on the screen; the keyboard (right) is used for typing in new values.

---

[7] See the article mentioned in note [3].

Fig. 17. Part of the layout of a LOCMOS circuit; the black blocks in rows contain the logic modules. The blocks were positioned in the row by the designer. Under each row the modules have been interconnected in two-layer wiring by the computer (red and blue).

A simplified model is therefore used: faults can only occur in interconnections, and it is assumed that such faults will either make the signal on this interconnection remain at the value corresponding to the logic '0' ('stuck at zero', 'SA 0') or to the logic '1' ('stuck at one', 'SA 1'). As a further simplification it is assumed that the circuit contains no more than one fault. A circuit that passes the test for all possible single 'stuck at' faults is said to be 'good'. The basic consideration is the reasonable, although unproved, assumption that all faults, even multiple ones, can be found in this way. A set of test patterns can be produced in three ways: automatically by means of a computer program, by the designer, or by means of a 'random' generator.

The automatic method uses the knowledge of the circuit structure, and is referred to as 'structural testing' [9].
Although this is in principle a superior test method — a complete test can be generated — the other two are also used. The reason is that the automatic method can require a great deal of computer time, especially for large circuits with many internal memory elements.

In the second method the designer generally uses the patterns he has used for simulation, supplementing them until he considers them sufficient for the test. The amount of work necessary for tracing all the possible faults in this way is impossibly large, however. It has meanwhile become common practice to accept incomplete tests. This method of test designing is the most widely used today.

The determination of test patterns by means of a 'random' generator generally gives disappointing results. In some cases, however, a test made by the designer can be supplemented in this way.

The quality of a test procedure is assessed by means of test-verification programs. For every possible fault in a circuit a verifier performs a simulation at the same time as the simulation of the fault-free circuit. The input for the simulation is the series of test patterns. A fault can be detected if different states appear at the outputs of the two circuits. The result of the verification is a list of detectable and non-detectable faults.

Testing is the most difficult subject for CAD. Since the complexity of LSI circuits may be expected to increase quite considerably, the means at present used will undoubtedly prove to be inadequate. Instead of being a tedious procedure after the design, testing will have to become an integral part of the design procedure. Experiments in this direction show promise.

CAD systems

A CAD system contains, in addition to the computer methods required during a design phase, such as logic simulation, a variety of aids that can be used in the transition from one design phase to the next. The interface of the electrical or logic design with the layout is by far the most important of these. This interface may be a constructive connection — so that the layout follows more or less automatically from the design — or a monitoring connection. In the monitoring case the designer still makes the layout, but he receives facilities for proving the correctness of the layout. With the conventional method a compact layout can be made if the designer is prepared to spend a lot of time on it; the automatic method produces a far less densely packed layout with very much less effort. There are also intermediate forms. For example, the automatic method can be 'opened up'; in this case a program proposes a layout, and the designer can then improve on its shortcomings. In such cases the CAD program will usually check whether these improvements are free from faults.

Two CAD systems will now be briefly outlined. In view of their different objectives, the two systems are somewhat different in nature.

**CAD systems for LOCMOS circuits**

Circuits made with the LOCMOS technology have extremely low dissipation and are capable of reasonably high switching speeds. The range of LOCMOS applications varies from simple standard circuits such as simple gates to complex LSI circuits.

The CAD system for LOCMOS comprises a number of logic modules, whose electrical operation is verified beforehand by means of a circuit-analysis program. The system consists of a method for logic simulation (PHILSIM) and a semi-automatic layout system. It also contains facilities for automatically converting the test procedure made by the designer into a test program for the test equipment. A number of rules have been drawn up for automatically making the layout for LOCMOS circuits. The layout is organized in a number of rows. Each row is divided lengthwise into three parts: the central part contains a number of modules, while the lower and upper parts contain the interconnections between the modules, which contain the logic functions. The layout of these modules has previously been made as compact as possible by hand, and the correctness of the layout has been verified by extensive checks and pilot production.

Designing in LOCMOS amounts to selecting the modules required, allocating them their positions in the layout, making the interconnections between the modules and finally making the connections to the outside world.

Although the allocation of a module to its place in the row can be automated, a 'manual' method has been chosen for this system. The main reason is that the quality of the positioning determines to a great extent the final compactness of the layout. For the time being the designer appears to be more capable of making the right decisions here, as compared with automated procedures.

Making the interconnections between the modules is the most labour-intensive part of the work. This has been fully automated in the CAD system described here, but there is still a facility for the independent specification of interconnections considered to be important, or of proving them under the supervision of the system if the automatic procedures have made less suitable choices. Part of a layout made in this way can be seen in fig. 17.

---

[9] See the article mentioned in note [1].
[10] See the article mentioned in note [5].
CAD OF LSI CIRCUITS

The associated data, represent an I\(^2\)L gate or an electrical interconnection, for example. The punched tape made with the digitizer is then read into a computer. From this tape CADILL can now form a CIRCUIT-MASK description, which will give a drawing of the layout. Up to this point CADILL does not differ from any other non-automated system. However, since under the CADILL rules some extra data are fed into the punched tape during the digitizing, a logic-network description can also be made from this tape. This description can now be used for a further logic simulation, not to verify the design this time, however, but to check the layout against the specification. CADILL thus offers the designer the means of verifying whether the conventionally made layout is equivalent to the logic design. In addition to this facility, CADILL also possesses a few other less spectacular but particularly useful functions. For example, CADILL can draw attention to obvious faults, such as unconnected inputs, at the stage of making the logic description from the layout. CADILL can also be used for plotting symbolic layouts. A symbolic layout deviates in some respects from the normal technological rules for presenting the circuit in a more readable form; for example, interconnection tracks are shown narrower than they are in reality.

Summary. In the design of LSI circuits it is necessary to have CAD facilities available. Circuit-analysis techniques can be used to verify analog circuits that are not unduly large, such as digital modules. Large LSI circuits cannot be handled in this way, and digital design aids are far more useful here. For this reason, and also because the digital technique is simpler for the designer, LSI circuits will increasingly be made with the aid of digital techniques, even for functions that are essentially of an analog nature. The repertoire of CAD software available includes logic-simulation programs for verifying the correct operation of the circuit, programs that provide help in layout design or that produce the layout completely automatically, and programs that make test procedures for LSI circuits and check their quality. Finally, for a number of widely used technologies there are also CAD systems in which the computer program includes the transition from one design phase to the next.

CAD system for I\(^2\)L circuits (CADILL)

Designs for the I\(^2\)L technology can be made with the CADILL system (Computer Aided Design for Injection-Logic Layouts). CADILL is aimed at the category of designs where no loss of chip area can be tolerated, and where the designer thus wants to make use of all the facilities a technology can offer. In such a case it is not possible to use automated design aids in view of the associated limitations, and so all the work will have to be done by the designer. This type of design will therefore be labour-intensive, and if no special measures are taken there is a serious risk of errors. The CADILL system seeks to facilitate the tracing of errors made during a design without in any way restricting the freedom of the designer. Broadly speaking, designing with CADILL can be described as follows. First, the logic system is checked against the specification in the usual way by means of logic simulation. Next, the designer produces the layout, which is then scanned and coded with the aid of a 'digitizer' (see fig. 18). All the coordinates in the layout are then punched on tape in accordance with rules imposed by CADILL. This means that special codes on the kind of information have to be fed in at the same time; these codes, together with

Fig. 18. 'Digitizer' for coding on a punched tape a layout made by the designer (right). The layout is held flat against a table; a cross-hair cursor system can be moved across the table in the x- and y-directions. A magnifying glass can be used for accurate positioning. The various instructions and the coordinates of the points selected on the drawing can be punched in the tape by pressing the appropriate buttons.