Large-scale integration from the user's point of view

by M. G. SMITH and W. A. NOTZ
IBM Watson Research Center
Yorktown Heights, New York

INTRODUCTION

The potential LSI user views LSI promise with a great deal of anticipation, but LSI problems with some trepidation. Obviously, he hopes for breakthroughs to relieve the strain of having to squeeze the last bit of cost or performance from the existing technological approaches—and of having to contend with the added hardware and software problems fostered by the need to improve his product only through system complexity. There are, in fact, very few things the system designer can do that have the impact of a significant technology advance in cost or performance.

LSI could be this sought-after breakthrough, even though some may see it as more of an evolution from low-level integrated circuits. Certainly, the basic manufacturing costs should eventually be significantly lower than today's circuit costs—and we should also see improved performance, increased reliability, smaller size, lower power, and easier serviceability. On the other hand, there are possibly, very significant negating factors, such as, high part number costs, long turn-around times, and difficult test and specification problems.

There would have to be many points of view concerning LSI from users, subject at least to the user's application and internal capabilities. Is it a large or small system; is there a high or low market volume expected; is it a high or low performance application; are there particularly sensitive cost, power or reliability requirements; what facilities does the user have at his disposal; and does the user plan to buy or make his components? Obviously, LSI is not equally attractive to all these users; particularly if we are not allowed to deal in an "ultimate" time scale.

Logic costs

Will LSI mean lower cost systems in the context which we know systems today? Consider a hypothetical CPU (Figure 1) which is somewhat representative of an intermediate-sized cost-performance system today. The example is chosen to be favorable to LSI. (In this example, we assume that the "LSI-equivalent" system is identical in function and performance although this may not be the way we would want to use LSI.) Reductions are assumed due to LSI use in the logic and special memory areas, but not in main memory. These costs assume very substantial reduction in silicon processing cost, i.e., more than an order of magnitude; such that logic circuits, for example, packaged to the module level, cost an order of magnitude less. Thus, if circuits cost, say, 50 cents today; then they would be about five cents apiece in our "LSI-equivalent" machine.

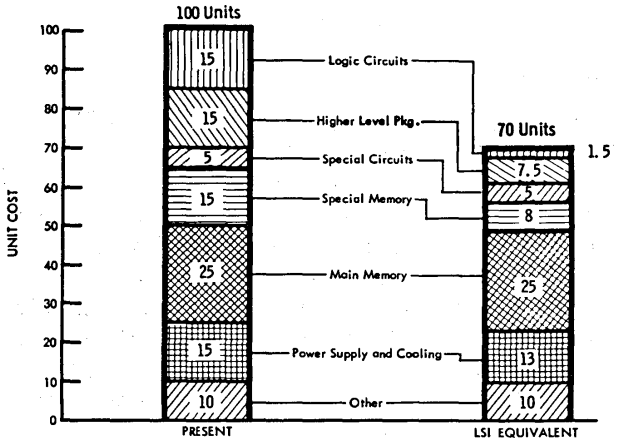

Figure 1—Hypothetical CPU costs

In overall CPU costs, we have saved 30%, a tidy sum no doubt; but far from the more than one order of magnitude in original silicon costs. However, consider this CPU in the hypothetical system reflected by the cost of Figure 2. (This system is a near minimal configuration and does not load the cost with a great deal of conventional I/O hardware, again favoring the LSI cost savings.) Our percentage saving has now been reduced to 18%.

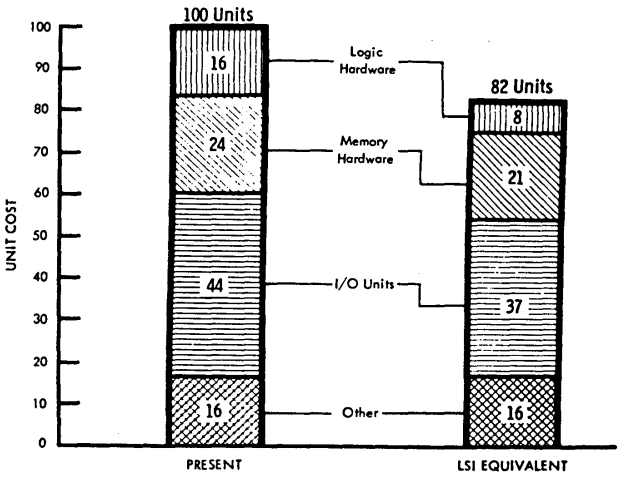

Figure 2—Hypothetical system costs

More function per dollar

What should we do with this saving? The key point here is that it could be turned back in for significantly more hardware capability; for example, about three times as much logic and twice as much special memory. Of course, there is still more of this story, as indicated in Figure 3, a hypothetical cost of ownership. (Certain costs such as training personnel are omitted.) Here our expectant user, filled with the anticipation of an order of magnitude decrease in cost, may have to settle for about 7%; and, as a matter of fact, many system examples would look worse.

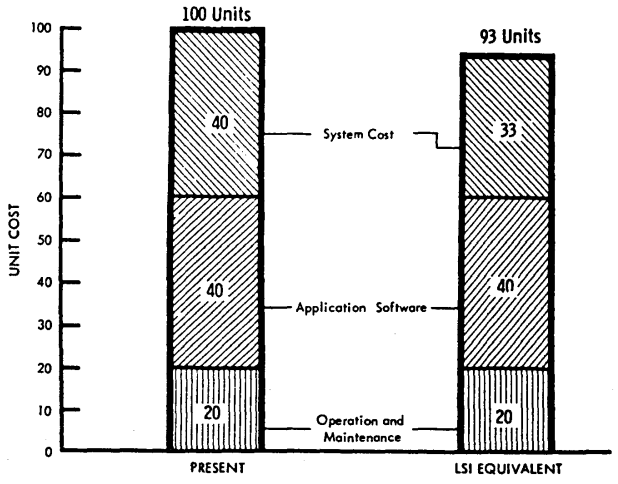

Figure 3—Hypothetical cost of ownership

A major cost to the system user is the cost of applications programming. If LSI hardware costs can be low enough, surely, here is a most challenging application, and clearly, we should be able to add significantly more circuitry to accomplish many functions which previously were either available only in the most expensive machines, or prohibitively expensive all together.

Some examples of functional expansion we would naturally consider are as follows. In the central processor LSI might be used to carry out more micro-operations per instruction; address more operands per instruction; control more levels of look-ahead; and provide both repetition and more variety in the types of functions to be executed. In system control, LSI might provide greater system availability through error detection, error correction, instruction retry, reconfiguration to bypass faulty units, and fault diagnosis; more sophisticated interrupt facilities; more levels of memory protection; and concurrent access to independent memory units within more complex program constraints. In system memory, LSI might provide additional fast local memory for operands and addresses; improved address transformation capability; content-addressable memory; and special fast program status tables. In system input-output, LSI might provide more channels; improved interlacing of concurrent input-output operations with automatic memory protection features; and more sophisticated pre- and post-processing of data and instructions to relieve the central processor of these tasks.

Special organizations

Certain special machine organizations and functional memories have considerable part repeatability and will be well suited to LSI from a fabrication standpoint. The machine organizations and functional memories referenced here have in common the characteristic that they consist of a number of identical data processing units or cells that are capable of concurrent operation. Otherwise, they can be quite different.

From a hardware point of view, they may vary from a few gates in the simplest associative memory cell or cutpoint logic cell[1] up to thousands of gates per processing unit in machines such as ILLIAC IV.[2] Included are the SOLOMON[3] and Holland[4,5] organizations.

Most of the special organizations of large arrays of identical computing units have been shown to be capable of solving any kind of data processing problem but have only been shown to be efficient relative to more conventional single processor organization in selected applications.

What then should the system organization be for LSI? Should we stick with existing organizations which characteristically leads to many part numbers, or should we consider special organizations around a universal part number? The choice here is certainly very much a matter of costs, and even more, a matter of where the costs are incurred. For example, if circuits cost nothing, but

part number charges are very high, the special organizations will be greatly enhanced. On the other hand, if part number costs (including turn-around time penalties) are negligible relative to the volume, we might expect that today's system organizations will prevail or at least will not be altered due to LSI alone.

Some studies have attempted to show the proliferation of parts economically possible if the number of part-numbers could be substantially reduced (assuming a range of system quantities, projected part number costs and other pertinent parameters).[6] Implicitly, identical system functions are assumed, although this is not likely. This proliferation is small for system quantities in the thousands and extremely high for the one-of-a-kind system. However, the LSI implementation of few-of-a-kind systems actually will be bounded by other, low-level-integration implementation means and costs. Thus, in the foreseeable future, only a small factor, generally between one and four times the amount of logic can be expended to create a one-part-number system from a many-part-number system. Generally, this is not adequate to implement the original function. This also implies that special organizations, with extensive use of repetitive logic (and memory) functions, will still have to be justified by what they can do differently and by significant improvements in performance. However, these systems may now be economically possible, at least, with LSI.

Volumes and part numbers

The importance of circuit volume and volume per part number is amplified in LSI. Aside from memory, where do we find LSI-compatible volumes? First, the total circuit volume is often higher for small machines than for the large central processing units (we include in the small-machine-category terminals and other peripheral equipment for larger machines as well). Also, the small machine, because it is usually sold in larger quantities, generally offers us more parts per LSI part number than does the conventional large machine (Figure 4). The average number of parts/part number is plotted for a plausible distribution of higher volume systems in each size category.

A "normal" part number repeatability in a total system is illustrated in Figure 5. Obviously it can be very high in memory, both within a system and in some cases across system types. Registers, of course, fall into a similar category. Certain portions of the control can also be fast-read, slow-write memory. Much of the logic, however, will be unique to the system. In the area of I/O and remote terminals, there is little repeatability within units, but many such units are employed within the system and across the system types.

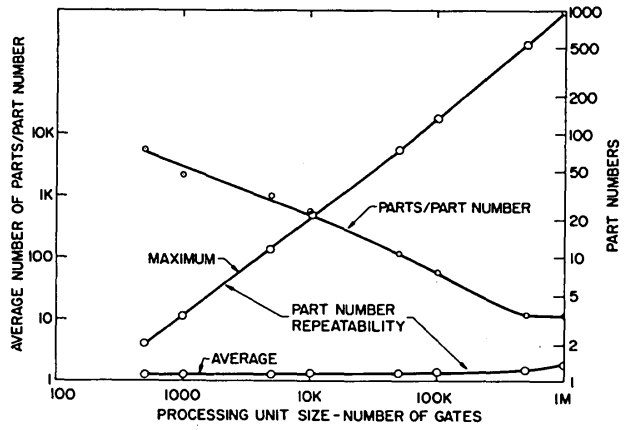

Figure 4—Plausible part number quantities and distributions for "conventional" systems (integration level, 100 gates)

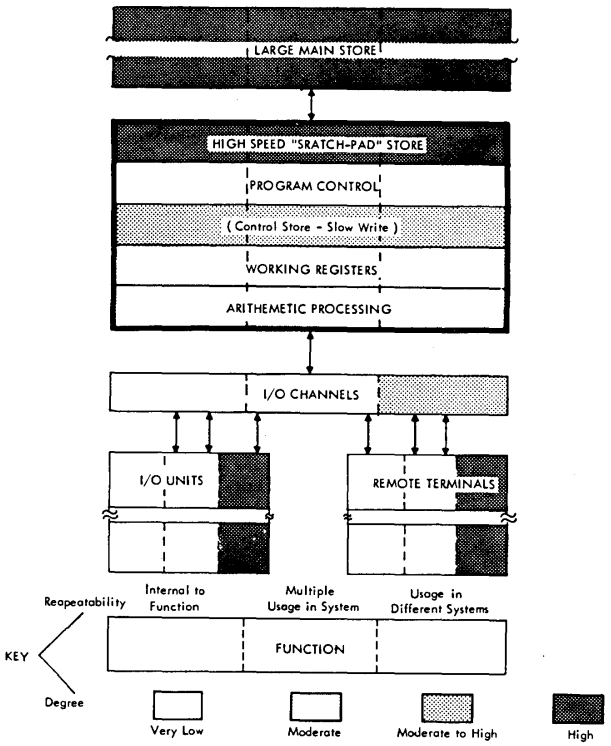

Figure 5—Part repeatability

Manufacturing and implementation costs

Most users must really believe they will have low costs before they will put much time and money into devising schemes to proliferate components in a system. Thus, we should expect that many early users will be timid. Even accepting that the raw manufacturing costs will be low, how much will it cost to generate an LSI part number? It is not unusual today for the mask set

alone to cost $5000 for one part number, assuming a 100-circuit integration level. If the machine size is modest and if it has a high market volume, we could use many manual techniques. Thus, these small, high volume systems or subsystems, along with memory applications, give us a place to start and afford an evolution to more extensive applications as the problems are solved. However, at present costs (Figure 6), the very small-volume machine would require more just for the LSI masks than for implementing the logic hardware by other means—and we have not included the LSI costs for partitioning, placement, wire routing, testing, and an unprecedented amount of interface time; easily another $5,000, today. Of course, under these circumstances, the small-volume system user will have to be motivated by something other than cost *per se,* if he uses LSI.

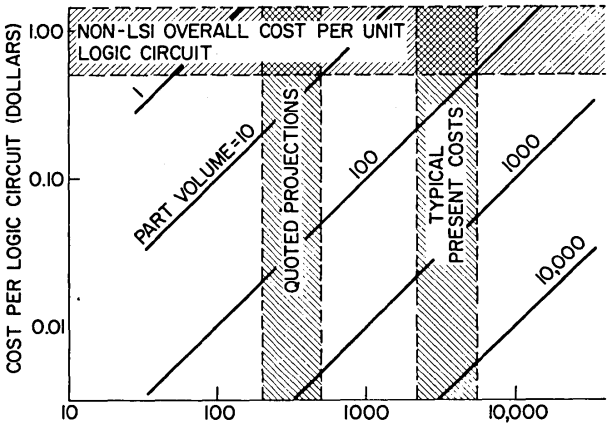

Figure 6—Mask generation cost (dollars)

However, an even greater problem exists for some who can get by all the previous hurdles. This is the delay which may be encountered by the potentially long turnaround times in a system environment already well-known for its high incidence of change; compounded by a much higher susceptibility of error, and costly errors potentially engendered by LSI. Can we live with this? Some system designers have speculated that the development of LSI machines would take 50% to 100% longer. If so, this is a significant loss of investment and sales or revenues. Add to that, delays due to changes after the product is released, and losses after a heavy investment in inventory, and the systems man could have paid many times the amount per circuit in a non-LSI form. In addition, the old problem of early technological obsolescence is worsened.

Implementation aids

We are all familiar with the major elements in these implementation procedures (Figures 7 and 8). Obviously, we cannot eliminate many, if any, of these functions; so what can be done to cut the operating times

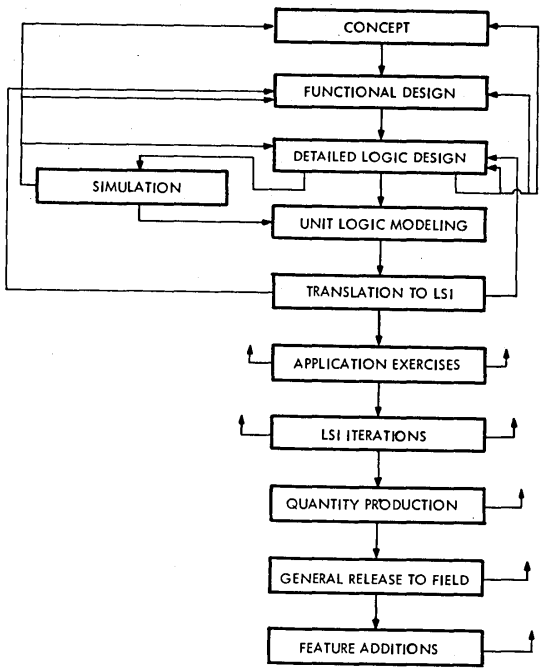

Figure 7—Typical small machine development and manufacturing sequence

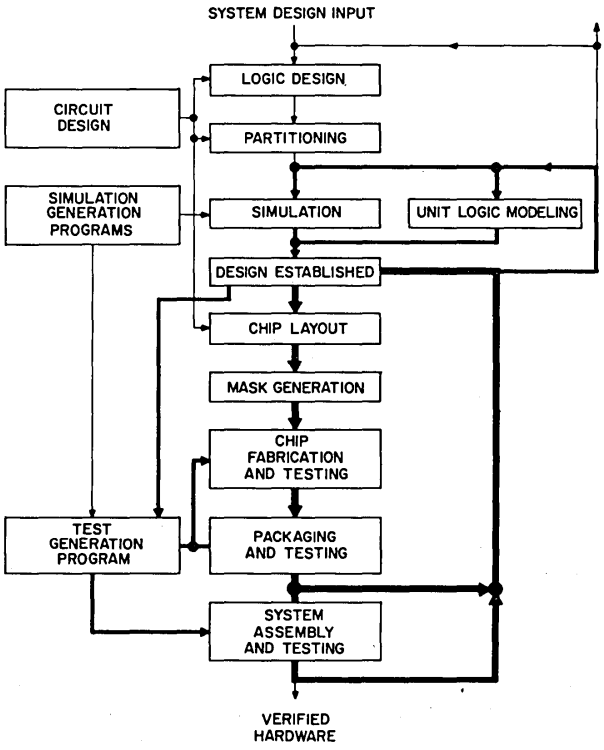

Figure 8—Development-hardware implementation

and costs, and the incidence of iterations? Fortunately, some of these functions are not in the main iteration loops and some of the costs involved are not unique to a particular part number, i.e., they can be shared over a range of part numbers or applications. However, we need much more.

Obviously, we must question both the manufacturer and the user in this area. Can some of these functions be gotten out of the main stream? Can operations be paralleled? Can we collect some function together so that the product spends a minimum amount of time in "in" and "out" baskets? Even while "in process," much time is lost in queuing.

In addition, can we assure that iterations will not take nearly as much time as the first time through? Can the hardware design impact this? That is, should we commit to master-slice approach with only a final metalization; or if we have a very high volume item, should we use a master-slice in development and a more efficient *ad hoc* layout in production? What are the trade-off's?

The user generally likes to construct the system (or a significant part, at least) as an integral part of design procedure. Should we also model hardware in unit logic parts in advance of generating the LSI parts? Also, we have been able to make hardware changes easily in the past, and it has become a way of life. This will have to change.

As indicated, not all the functions have to be in the main iterative loops and, in fact, it is really these loops which the user worries about the most, namely, the loops which he may have to go through if he desires (or must) make changes. Perhaps this is because the total design sequence may be measured in many months, if not years; while the iterations need to be measured in days or weeks. Certainly, if the situation is not significantly worsened by LSI, we can use LSI. Within this context, then, what implementation procedures should we have for LSI? Figure 8 represents the rudiments of such a system, and although iterating loops can take place almost anywhere, the main iterating paths are as indicated. Notice that by implication, we are only considering minor changes. Hopefully, our detailed logic design simulation and unit logic modeling procedures will have proven the basic concepts and functional designs. Notice too, that the circuit design and partitioning are not part of the principal loops. (Conceptually, we could envision circuit design as an integral part of the layout design, adjusting the circuit parameters somewhat, due to the variations in capacitance loading, for example.) Generally the circuit (or circuit family) can be specified along with the necessary design and layout constraints and these constraints can serve as boundary conditions in the layout. Partitioning is not part of the iterative loop if changes are minor. If chips, susceptible to change, are designed leaving some extra chip area and I/O pins, then chips could be changed without impacting the basic partitioning. Of course, each case would need to be examined separately and, with some ingenuity, "fixes" might be made using some special adjunct circuitry.

The major iterating loops are indicated in Figure 8 by the heaviest lines. These loops really reflect that many of the changes may be due to errors in implementing the logic or they are minor enough that a larger loop can be by-passed. However, we should really have fast iterating loops including simulation for the more significant changes we will surely encounter. Note that the emphasis continues to be on iterations and iterations can often be shortened, particularly when using automated design methods.

Automated techniques

Obviously, if we are to experience the dramatic increases in the number and sizes of LSI systems, we must offset the enormously increased burden of labor and time on detailed logic designs, placements, layouts, mask generation, testing programs, and documentation, by highly automated means. In turn, if we are to develop costly automated means to reduce these handling times and costs, we obviously have to develop "master" programs and facilities which can serve many part numbers and system applications. However, it is difficult to see both fast turn-around and low cost in our part-number development cycle. We look to a facility, such as that in Figure 9, to provide us with the necessary flexibility and turn-around times, at least. Functionally, this is a part of most of the iteration loops and, in present-day circumstances, it is notoriously unpredictable, time consuming and costly. Does this mean a special high-speed development facility apart from the manufacturing facility? After all, the manufacturing facility may not be optimized for both fast response and low cost; but an added facility optimized for fast turn-around time may be an expensive addition to our part number costs, since such a facility may have a very low through-put. It could cost several hundred thousand dollars and require a comparable investment in people per year. Certainly, it is not difficult to see a few thousand dollars per part number, and perhaps much more, expended in this facility alone. Perhaps, then, if we are to truly have low-cost part number generation, we will have to envision this facility as a special subset of a larger complex—but we must be certain that this does not seriously jeopardize the turn-around time.

What is being done in the industry in some of these key problem areas? One of the more sophisticated approaches, and actually the earliest truly LSI approach

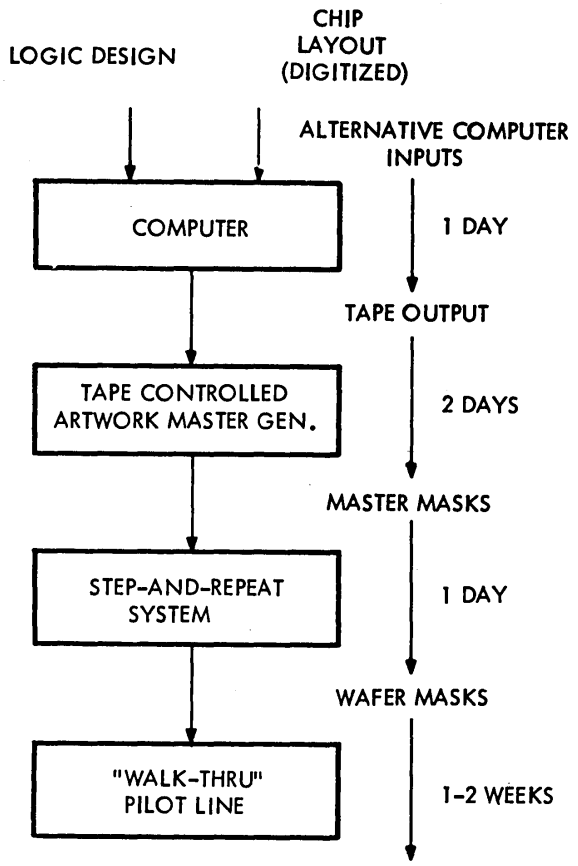

Figure 9—Fast turn-around development facility

is the programmed interconnection pattern (IBM)[7,8] or discretionary wiring (T.I.)[9,10] approach. Circuits are pre-tested and interconnected on a wafer. Interconnection patterns are determined by the computer which in turn guides a light beam directly on the wafer (IBM) or a cathode-ray tube beam to produce a film mask (T.I.). Extremely fast wiring programs have been generated, costing only a small fraction of a cent per circuit, although at some expense in the silicon area (the computer speed is greatly enhanced if the structure is tailored for the computer and if plenty of wiring space is available). Thus, while these schemes may have lower densities than the *ad hoc* fixed pattern approaches, they still may be competitive subject to the various yield and density factors, and they may be useful in lower volume cases. A key point here is that the rudiments of a system exist today to convert logic diagram into final mask, whether it be for the discretionary or fixed pattern approaches. When applied to fixed pattern approaches, which place a high premium upon maximizing the silicon area utilization, the problem is much tougher. (This accounts for some of the apparent discrepancies in the status reports concerning automated "wiring" approaches.)

Mask-making

What are some of the approaches being proposed for the standard fixed pattern approaches? A number of companies have reported efforts recently, including Fairchild,[11] Motorola,[12] and IBM.[13,14] While not all of the details are common to all the parties indicated, in general, each approach provides for both computer generated layouts and a manual input, at least for iterations. Some of these systems provide a graphic console for designer-computer iteration. The output of some of these systems is a rubylith master; at least one uses photographic plates. All of the systems reported are using reduction and step-and-repeat operation to generate the final masks.

An interesting variation recently reported at IBM,[13,14] combines computer and manual layout techniques to provide a very effective mask-making capability. The procedure begins with a sketch of the basic chip format which is also stored in the computer. The computer also stores frequently used structures or patterns of varying complexity which can be used without repeating the detailed coordinates. The designer sketches in his interconnections and devices and key coordinates are entered onto punched cards. From this sketchy information, the computer is able to generate the complete set of several masks, although the designer prepared only one sketch. The program also permits rapid alterations in the masks with the change of a few cards—and this scheme, if combined with a graphic terminal, would permit even faster alterations. It is clear these schemes give excellent flexibility and reduce the layout and mask-making costs significantly; but it does not replace the need for a totally automated system, which is both economical and efficient in the use of silicon area, particularly for low systems-volume situations. Excellent progress is being made in this area.

Testing

Testing continues to be a significant problem, not only because every part has to be tested, but because effective test generation programs are difficult to construct. A number of persons have attempted to illustrate the enormity of the problem in pointing out that at least 2^n variation is possible, where n is the number of inputs to the chip. For example, a 100-circuit chip with 50 inputs would mean 10^{15} tests, with many more possible if the chip has internal storage states or the sequence of testing must be varied. Fortunately, most of these tests are redundant, at least to some extent, or may otherwise be compromised. However, it is still a major problem to

define and develop an efficient and properly qualified set of tests. In principle, we may have a solution for the d.c. functional test problem,[15,16] even for sequential logic, but there has to be a serious question about the economics of the situation for the limited volume machine.

There are sophisticated test generation programs in operation today which were developed to test cards of conventional logic. Although these techniques are extendable to LSI, they were developed in a less stringent environment and impose constraints, as they presently exist, which limit our flexibility in LSI. For example, there are restrictions in the number of levels of sequential logic and if this is observed, the logic on the chip would have to be broken up at the expense of I/O pins and performance. Several companies have reported work here but it is not clear that adequate general-purpose test programs have been completed yet.

Simulation

Simulation of the potential LSI machine or components, either by software or unit logic hardware or some combination of both, is the accepted means of detecting and correcting design errors before production. However, there is at present no completely effective and comprehensive simulation system to detect all system errors, although there are techniques available aimed at most aspects of system simulation.[17,18]

There are a number of programs for higher level system simulation that can be applied down to a rather detailed level. Such programs can simulate the operation of a system quite accurately, but the problem of detecting design errors also requires that the system users as well as designers describe completely all the ways in which they expect to use the system. This is one of the main problems in system simulation. A system has many parts that are expected to operate concurrently, and the design should contain interlocks to prevent improper sequencing of operations that use several semi-autonomous machine parts. The designer cannot always anticipate all the combinations of applications of the machine parts or sequences that a user may employ. Sometimes the designer incorrectly assumes that certain combinations either don't make sense or result in don't-care conditions. This problem is compounded somewhat by the fact that a software system may be expected to take care of some undesirable conditions without full knowledge of the time and memory needed by the software system. Simulation of the architecture of a machine system is intended to encompass the hardware and software operation of the system but even in this level, the architect cannot foresee, in a complex system, all the anticipated combinations of conditions dependent upon both the programs and data that may occur in some practical application. To investigate exhaustively all possible system states or conditions in detail, even for fairly simple systems, is out of the question because of the years of time that would be required.

The only reasonable approach to large-system simulation is to simulate at several different levels of detail, and at each level to simulate a limited number of units, each with limited complexity. At the higher levels of complete system simulation, one is primarily trying to check out broad timing properties of the system and insure that improper sequences of events do not occur. Simulation at an intermediate level will determine whether the functional units of the system will individually process data correctly. At the most detailed level of simulation, real circuit-to-circuit delays and the basic system synchronization techniques become important. If the system is basically asynchronous then the logical interlocks need to be carefully checked; but not detailed circuit delays. If the system is not basically asynchronous detailed wiring distance and circuit loading delays become important. As far as LSI is concerned, one problem is to insure that all timing paths on semiconductor chips stay within certain tolerances. If the design tolerances are too loose, the system will operate too slowly. If the tolerances are too tight, the yield of acceptable chips may be too low.

An alternative to software simulation in the past has been hardware simulation. This has usually taken the form of a prototype model built with pre-production circuits. For LSI, this hardware simulation may be done with unit logic circuits provided they can be designed and connected to realistically simulate LSI circuits and chip characteristics. This is still the path contemplated by many users.

SUMMARY

LSI use will be very much a function of component or system quantities, and even more, there will be a significant proliferation of logic circuitry within systems having large market quantities, e.g., terminals. The market volumes necessary for profitable LSI use will decrease as more economical solutions evolve but the "few-of-a-kind" components may never be justified by cost considerations alone.

The main concern of users in general is turn-around time (and costs) in the face of the many design changes they anticipate. Highly automated schemes and more effective communication between the user and the component manufacturer obviously hold the key. Excellent progress is being made in many areas, particularly in mask-making, partitioning and testing, but much remains to be done.

ACKNOWLEDGMENT

We would like to acknowledge support from P. Cook, D. Critchlow, H. Freitag, E. Schischa, J. L. Smith, and S. Triebwasser.

REFERENCES

1 R C MINNICK
Survey of microcellular research
Journal of the ACM April 1967

2 D L SLOTNICK
Achieving large computing capabilities through an array computer
AFIPS Conference Proc vol 30 April 1967

3 J GREGORY R McREYNOLDS
The SOLOMON computer
IEEE Trans on Electronic Computers vol EC-12 no 6 December 1963

4 J HOLLAND
Iterative circuit computers
Proc WJCC 1960

5 W COMFORT
Highly parallel machines
Proc of 1962 Workshop on Computer Organization Spartan Books Washington D.C. 1963

6 W NOTZ E SCHISCHA J L SMITH M G SMITH
LSI—The effect on systems design
Electronics February 20 1967

7 S TRIEBWASSER
Programmable interconnection techniques
ISSCC February 11 1966

8 H FREITAG
Design automation for large-scale integration
WESCON Los Angeles August 25 1966

9 J S KILBY
Device fabrication
ISSCC February 9 1966

10 M CANNING R DUNN G JEANSONNE
Active memory calls for discretion
Electronics February 20 1967

11 C H MAYS
Computer-aided design for large-scale integration
ISSCC February 16 1967

12 N L HAZLETT
Computer accelerates design and production of large arrays
Electronics February 20 1967

13 D L CRITCHLOW
Layout and mask generation for large-scale integration
IEEE international convention March 23 1967

14 P W COOK G A LEMKE A BRENNEMANN
An automatic integrated circuit mask artwork generating system
Microelectronics Symp St Louis Mo June 20 1967

15 E EICHELBERGER
Hazard detection in combinational and sequential switching circuits
IBM Journal March 1965

16 J ROTH W BOURICIUS P SCHNEIDER
Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits
Available from IEEE Computer Group Repository—received Dec 9 1966

17 D TEICHROEW J LUBIN
Computer simulation—discussion of the technique and comparison of languages
Communications of the ACM October 1966

18 R LARSEN M MANO
Modeling and simulation of digital networks
Communications of the ACM May 1965