THE TEACHING OF VLSI DESIGN

J. Allen

Rapporteurs: Mr. J.G.B. Heal
Mr. S. Mansi

Abstracts

1. Representational Issues in VLSI Design

In VLSI design, many different representation levels, e.g. architecture, logic, circuit, device, layout, must be manipulated and maintained in a consistent way. This leads us to consider algorithms competence/performance distinctions, cells represented as programs that generate them, equivalence of text and graphic representations, new MOS digital system and timing models, placement and routing constructs, and the role of modularity and hierarchy in these representations.

2. Compilation of VLSI Layouts

Many different target structures can be generated by progrms that operate on a high-level functional description. These programs can generate logic array (including PLAs), state machines, register files with localised ALU capability, and parallel co-operating processors. These techniques are reviewed, and parallels between hardware and software compilation are noted.

3. Performance Optimization of VLSI Architecture

We view computer architecture as the specification of computer performance. Techniques for exploring and manipulating space/time tradeoffs while maintaining algorithm competence are described. The realisation of fully parallel data dependence graphs with operators bound locally to relevant memory cells is discussed as well as large parallel systems, including the use of restructurable wafer-scale VLSI. Consideration is also given to the architecture of special hardware for VLSI design, including high-performance work stations.

LECTURE 1

REPRESENTATIONAL ISSUES IN VLSI DESIGN

The overall problem of VLSI design can be considered as the conversion of an algorithmic representation to a custom integrated circuit layout through a succession of different representational levels by means of a set of transformations. From this perspective, it can be seen that a major focus of research in integrated circuit design is the determination of the particular levels of representation, as well as the transformations that interrelate them. Modern design requires the careful characterization of many different aspects of the design. The initial specification is often provided in some functional form that specifies what the circuit is to achieve. At the next level, an algorithm provides not only what must be achieved, but how that result is to be obtained and at this level, of course, the question of architecture is raised since in general there will be many architectural alternatives appropriate to a particular functional requirement. Once the architecture is fixed, then it must be realized, in terms of a set of modular components, including memory structures and arithmetic logic units, generally realized as a set of gates. The interconnection of these structures is specified by means of a topological representation. Each of these constituent modules must, of course, be realized in terms of a particular circuit representation and these circuits will contain a variety of electronic devices. Finally, the devices and their interconnect are realized in terms of a geometrical layout specification sufficient to permit the generation of integrated circuit masks appropriate for manufacture. From

this brief overview, it can be seen that the variety of representations needed to specify a custom integrated circuit is extremely diverse, and that it is necessary to be able to transformationally pass from one representation to another representation level. Furthermore, it is important that information specified at one representational level be aligned with that at other levels and that this information be kept consistent across all representational levels. An additional difficulty is introduced due to the unsettled state of current techniques for representing architectures, algorithms, and functional specifications. Thus, a great deal of current research in integrated circuit design is devoted to the determination of appropriate representations and the means to manipulate them. This of course is nothing more than the usual scientific quest for appropriate abstractions that allow the designer to focus on those attributes of the overall design that are of interest at a given point.

Due to the history of computing technology, it is unfortunately the case that algorithms have been specified in terms of a particular architecture, namely the single sequence Von Neumann machine. Thus, it is common place to find the precise characterization of algorithms given in terms of a program intended for execution on such a single sequence machine. Because of this orientation, latent parallelism within the algorithm is often not exploited and the expression of the algorithm may take on an unnatural form. What we would like is a formal means to specify the algorithmic competence as distinct from the algorithmic performance. By competence we mean the functional nature of the algorithm or what it has to perform, no matter how the algorithm is implemented. Performance, however, refers to that which is optional in the algorithm and hence may be fixed by a variety of implementations. Unfortunately, there is at

present no well developed formalism that allows us to characterize algorithms in terms of separate but coordinated structures for competence and performance. It appears that some sort of declarative structure, related to the functional requirements for the algorithm, may be appropriate for this task. For example, simultaneous sets of linear equations, such as those found in linear circuit theory, characterize a set of constraints that must hold over this equation system without any specification for the way in which the independent variables are to be solved. It is possible to construct a network of these constraints for every such system and to erect on this network a variety of different solutions. In fact, it is easy to obtain from these networks several different sets of independent variables which may lead to equivalent but alternative forms of the equation system. While these constraint representations are attractive as a means to represent algorithmic competence, they have not been extended to a wide variety of algorithmic classes and it is not clear how that extension can be achieved. Nevertheless, they do provide an insightful static declarative specification of those conditions that must hold under any admissible solution, and they do indicate the possibility of a large variety of solution techniques or performance strategies based on this static structure. The proper specification of algorithmic competence and performance should be regarded as a basic problem for computer science, as well as integrated circuit design. This is a very difficult problem and it is appropriate to look for alternate techniques for achieving the means of varying performance strategies without the necessity of explicitly revealing a representation for algorithmic competence. We will return to this topic later in this series of lectures.

At the architectural level, it is desirable to have a variety of modular functions available which can be juxtaposed to form a comprehensive set of both logical and memory structures that can be drawn upon in any particular custom design. The particular selection of these structures is highly dependent on the particular technology employed and in these lectures we will be focussing only on those structures appropriate for MOS design. In such a technology, the simplest logical forms can be derived by generalizing upon the simple inverter circuit replacing the active device by either two pass transistors in parallel or in series, yielding either a NOR gate or a NAND gate, respectively. For circuit reasons, the NOR gate is the preferable structure to use since the number of terms in the NOR gate can be changed without forcing any change in the specification of the load device of the circuit. It is important to realize that this fact reveals the modular nature of NOR circuits and the fact that they can be readily modified in the context of a larger canonical framework without forcing major revisions in that framework. Thus for example, NOR gates have been combined for many years into larger regular structures, such as the so-called Weinberger Image, which is a convenient way to juxtapose a collection of NOR gates into a regular geometric structure. This approach is particularly important, since it is easy to write a computer program to generate such arrays from an input logical specification and the ease of such programming is substantially enhanced by the modular nature of the NOR circuits that are employed. Generalizing a step further, NOR gates are frequently used as the basic elements of program logic arrays which are implemented as two planes of NOR gates, even though they are often referred to as performing the AND function, followed by the OR function, in order to realize any arbitrary logic

function as a sum of products. The modular nature of NOR-NOR PLAs is also responsible for the ease of PLA partitioning, segmentation, minimization and provision of input decoding, all of which are performance refinements to the basic logical capability. Just as it is easy to generate Weinberger image arrays, by means of a program, many codes have also been written for the generation of PLAs. These programs are exceedingly useful, since they provide the means for generating circuitry to compute any arbitrary class of logic functions in terms of well-defined regular structures, without the necessity for the designer to specify any of the circuit or layout details. Thus, the designer may use such a program interactively to estimate the size of the PLA structure, its speed, its power consumption and the precise location of all interconnect points to it. This capability should be regarded as an example of the desirable performance of all design tools in that the designer is able to ask questions that are important to the overall design of the system, without the need for instantiating all of the details. An important topic for further research in VLSI design is the search for additional canonical structures that will provide both memory and logic capability appropriate to a wide variety of tasks, but which can be generated automatically by means of design programs similar to those used for PLAs.

MOS technology has a large effect on the realization of logic in custom integrated circuits. Many designers are familiar with models appropriate to TTL design that utilize the so-called Boolean gate model. This model is unidirectional, in the sense that there are well-specified inputs and outputs in each model form and that all memory must be provided for explicitly in the design. This model is not, however, appropriate to MOS design, since past

transistors provide a bidirectional flow of information and circuit modes are often used for the storage of information without being explicitly represented as memory storage devices at any level of the representational structure for the design. For this reason, there has been a need for new representations that consistently and accurately model the full range of logic structures available to the MOS integrated circuit designer. An appropriate theory for this purpose has recently been devised wherein the MOS transistor is modelled as a switch that has three possible states. These states are either the closed state in which the switch is conducting, the open state in which there is no current flow, and the so-called x-state which specifies that the conductive state of the transistor is currently unknown. A very useful unit delay logic simulator has been built around this model that provides the designer with the means to accurately characterize both combinational and sequential logic circuits. A particularly important advantage of this simulator is the fact that its input representation can be derived algorithmically from the geometrical layout information of the circuit, resulting in a topological connection of idealized switches appropriate for simulator action. At any given time, the simulator establishes equivalence groups of nodes that are interconnected by conducting transistors and then sequences the circuit through its successive states, thus mirroring the flow of charge through conducting transistors to establish logical values by means of charge distribution at these nodes. An additional feature of the topological representation of interconnected transistors is that several tests for well-formedness of the circuit can be performed at this topological level. For example, the inadvertent connection of distinct power buses can be detected, as well as floating circuits and other ill-formed structures. A

particularly useful test is that which checks to see if every internal node can be varied between both binary logical values. This test has revealed many subtle difficulties in circuits that could not be readily discovered by other means.

Moving on from the logic level of representation, we encounter the circuit realization of the desired logic. It is important to realize that there is a many-one relationship between a given set of logic equations and the circuitry needed to realize that logic. For example, an exclusive OR gate can be realized by means of combining NAND structures, but it can be achieved much more simply by means of a simple pass transistor network in a generalized inverter structure. It is particularly important to realize that although designers frequently specify a circuit in terms of a so-called stick diagram that shows how the various levels of interconnect are related, there is a strong tendency to infer from this topological stick diagram circuit properties or geometrical relations that are not necessary. Thus, the designer must be constantly aware that not only can the desired logic be characterized in terms of several different topological structures, but that each of these topological structures can be realized by a variety of different circuits and that each of these circuits can in turn be implemented in terms of geometrical layout artifacts in a large number of different ways. Thus, there are many degrees of freedom which are often not exploited by the designer. We feel that while many designers are sensitive to architectural space/time tradeoffs, they are much less sensitive to these low level engineering choices that are equally important to a successful design. In fact, at the topological level many aspects of performance, dictated by the actual geometrical layout are abstracted away.

Fortunately, programs are available that provide the analytic techniques for going from a geometrical layout to an equivalent circuit that is in a form appropriate for a circuit simulator. Thus, an integrated circuit specification can be completely simulated at the circuit level in an accurate way that predicts performance of the final circuit. What is missing at present is the synthetic capability for generating a geometrical layout under the constraint of a particular performance specification. Instead, our current capability restricts us to generating alternative implementations and then using analytical techniques for characterizing their performance.

Finally, we turn to the specification of the detailed geometrical layout. Many of the means for specifying these artifacts have been derived through experience with practical devices used in integrated circuit manufacture that are not necessarily appropriate forms for use by the designer. Thus, some systems require the designer to specify rectangles and others allow the designer to indicate the opaqueness of the several mask layers with reference to a coarse geometrical grid. Designers have had to cope with a variety of limitations of such representations, such as the specification of rectangles in terms of not only their length and width, but also their center point and angle of rotation. This choice of representation was guided by the design of a particular device for generation of masks, but increasingly rotated rectangles are not allowed in designs since the use of orthogonal geometries provides for much easier layout generation as well as much more efficient design rule checking. The use of wires is another example of an awkward construct in many representational schemes. For example, in some cases wires have been specified as having round ends, but when the wires are of different width, this can cause substantial

difficulties at interconnect points. Furthermore, it is often necessary to represent these wires in terms of a juxtaposed set of rectangles whose connectivity must be maintained through a variety of operations on the mask data. Traditionally, the geometrical layout information has been represented as a set of static forms, represented in terms of some simple syntax. Nevertheless, recently modern computational techniques for the procedural representation of knowledge have been successfully used. Thus, it is possible to represent a detailed geometrical layout as a program which when executed will generate the final requisite static forms needed by the mask generation apparatus. This procedural form allows for easy parameterization of the design, as well as the symbolic naming of different parts of the design. For example, it is possible to relate one part of the geometrical layout to another in symbolic terms without having to specify the actual numerical distance between them, later binding these symbols to the particular geometry without requiring the designer to be cognizant of these details. The use of procedural techniques for representing layout can be regarded as an instance of delayed binding, a frequently used technique in computing that preserves degrees of freedom for the designer without the necessity to cope with a variety of low level implementation details.

An important aspect of the geometrical layout representation is the need to perform design rule checking, an exercise which consists of verifying that the widths of structures, as well as their mutual separation obey a set of constraints called design rules. Traditionally, these design rules have been expressed in geometric form, but there is some debate at present concerning the use of semantic constraints at the design rule level. For example, a so-called

twisting "snake" of diffusion wiring can be regarded either as a wire or as a resistor and when it is regarded as a wire it may be possible to disregard some of the spacing constraints, but this is not possible if it is to be regarded as a resistor. This raises a very large representationl issue since if semantic considerations are to be employed, the natural tendency would be to require the designer to indicate not only what structures are present, but what their functional intent is. Another example of such specification would be the requirement in some systems that designers indicate explicitly each and every intended transistor, so that if transistor structures are discovered by means of analytical computational techniques applied to the final detailed layout, then the structures would be regarded as spurious if they had not been explicitly called for by appropriate symbolic designation from the designer. Lastly, it is important to remark that design rule checking and indeed all procedures that operate directly on the geometrical layout are computationally intensive. Modern circuits can require the specification of several million rectangles on the various mask layers and since design rule checking requires computation of mutual overlap between distinct rectangles, enormous amounts of computer time can be consumed by naive versions of design rule checking algorithms. Two approaches have been taken to alleviate this problem. In the first approach, the design has either been naturally or arbitrarily broken up into a set of cells and the design rule checks are carried out for the most part within these cells, thus leaving only a small number of checks to be performed at the boundaries between cells. In another approach, each cell is designed in such a way that it is well-formed by itself, even though it may contain shared structures with adjacent cells. It is then possible to adapt the design rule

checking software, so that once each constituent module is checked for well-formedness only a small amount of boundary checking must be done when the modules are interconnected together. These techniques comprise so-called hierarchical design rule checking techniques and have currently been implemented in three or four different versions. A last approach to the complexity of artwork analysis programs is the use of special purpose hardware, which can utilize novel custom circuits to speed up the computations by at least two orders of magnitude.

In this lecture, we have sought to give a view of the diverse representational requirements of modern custom integrated circuit design. Much current research is focused on the discovery of appropriate structures that allow for both concise and insightful generation of integrated circuit layouts as well as the efficient determination of their well-formedness at all of the different representational levels. Perhaps the greatest difficulty has been encountered with the specification of functional representations and although a number of hardware design languages and other techniques have been suggested for this purpose, there is by no means consensus at present and we can expect that a substantial amount of continued research, as well as accumulated experience will be needed before a satisfactory solution is obtained.

Discussion

Professor Dijkstra was interested in how much the designer had to be aware of the variability of the processing. **Dr. Rideout** said that the small University user was in effect getting infrequent snapshots of the production line, and is particularly dependent on variations, so some form of characterisation of the process was essential.

Professor Randell speculated that perhaps foundries should be sharing wafers between chips of different users, spreading the variation - a form of time sharing instead of batch processing.

LECTURE 2

COMPILATION OF VLSI LAYOUTS

In the previous lecture, we discussed the various levels of representation involved in the design of custom integrated circuits, together with the transformations needed to go between these representations. In this lecture, we will focus more on the transformations than on the representations, and in particular on means for transforming automatically from a high level functional representation to the low level artwork details, characterizing the geometrical layout. In order to effect these transformations, it is important to be able to both precisely characterize the input specification in terms of a formal functional representation, but also to be able to utilize canonical target structures that are sufficiently restricted that programs can build particular structures of this type corresponding to the input specification. Over the years, a variety of target structures have been introduced and we will trace several of these leading up to a modern perspective on the current status of what is often called silicon compilation.

Perhaps one of the earliest attempts to provide a technique for building large circuits from small elements was the so-called standard cell or polycell approach. In this technique, a variety of small cells are designed, using a rectangular bounding box of uniform height, but varying width. In the earliest approaches, interconnect has been provided on only one side, but in more recent versions, interconnect was provided on two opposite sides. The idea was to provide a small, but growing set of structures, including gates and memory

elements that could be composed to form larger circuits by means of interconnect paths. Typically, the standard cells were arrayed in rows, side by side, since they were all of uniform height, and the interconnect was provided in channels between the rows. Generally, two rows were arranged back to back and although earlier approaches required so-called end-runs of routing along one row, around the end of the two back-to-back rows and along the opposite row, more general systems allow for feed-throughs between the back-to-back rows. Each of the cells of the standard cell system is completely characterized in that the cell has been designed, simulated, and all of its performance characteristics are well-known. In terms of binding time then, these designs are completely bound and no changes can be made to the cells in any regard. Channel routers have been provided for interconnecting the I/O points of these cells, so that a completely routed chip can be formed. It is fairly easy for the designer to specify his circuit in terms of these composite standard cells and to provide an interconnect list. The design software then arranges these cells and provides all the interconnect. In this way, a custom circuit can be generated very quickly. The main objection to this technique has been the large amount of space needed for the wiring channels, often amounting to as much as 50% of the area of the chip. Nevertheless, it is important to remember that the standard cell discipline is sufficiently constrained that placement and routing can take place automatically without any designer intervention, and this is a very large advantage which in large measure offsets the objections due to channel routing area. In high performance circuits, however, the delays associated with long wires may be unacceptable and a more flexible means of placing and interconnecting the cells may be required. It is important to remember,

however, that the standard cell approach is in no way obsolete, and for many applications where speed of design is more important than circuit performance considerations, these techniques will be readily accepted. They have even been applied for rather complex circuits, such as a 32-bit microprocessor. An additional advantage is the ability to place large amounts of memory on a standard cell chip, so that it can also be wired into the remaining set of standard cells.

There has been a continuing search for other ways to constrain the layout process so that a compilation strategy could be used. A variety of these have already been mentioned, including the so-called Weinberger Image of NOR gates and the PLA designs that are so frequently used. It is well to point out at this point that PLAs can also be generalized through the use of feedback to provide finite state machines, so that any techniques that lead to highly optimized PLA designs have great utilization, not only for logic but also for state machines. Recently, another regular technique used for CMOS arrays, has been proposed by Lopez and Law at Bell Laboratories, and this has led to the ability to create large circuits very quickly from a high level functional specification. We should not, of course, forget gate arrays as another example of a highly constrained architecture into which various designs can be compiled. Gate arrays, sometimes called master slice circuits, are available in a wide variety of technologies and provide another point on the scale of binding time of design. In the case of gate arrays, the gates are fabricated on a standard format, but the interconnect is bound later in accordance with the customer's desires. In this way, it is possible to provide very rapid turn around of gate array circuits and due to the exploitation of a number of aggressive

technologies, high performance circuits can be readily fabricated in this way.

Recent practice in the compilation of circuits from a functional specification has sought to further delay the binding time of various design parameters in accordance with the input specification. Furthermore, these approaches have sought to design entire systems on a chip, rather than individual modules. A well-known example of this approach is the so-called bristle blocks technique which allows the designer to place a series of arithmetic logic and memory elements along a two-bus structure. This data path structure is then controlled through a set of signals which are generated from a set of decoders which in turn act upon an input instruction word. The bristle block system automatically generates the register for the instruction word and the decoders which operate on the various fields of the instruction to provide the controlled signals for the data path. Test access circuitry is also provided by the bristle block system. It is important to remember that the bristle block system as it is currently conceived provides only a single architecture and that systems requiring different kinds of architecture would have to use an entirely different compilation approach. Some designers have speculated that a small number, perhaps less than 10, of canonical architectures would suffice and that a compiler could be built for each of these. In this way, specialized compilation strategies might be employed to generate the different kinds of canonical architectures. While it is not clear just how many specialized target architectures would be needed, this approach has so far not been realized and there is only a very limited initial experience with the original bristle blocks architecture.

When a microprocessor is being designed, it may be possible to characterize the action of the microprocessor in terms of a set of microcode specifications. These specifications may then be used by a compiler to generate not only the control structure, but the data path to achieve the functionality specified. Such techniques have been applied recently in the design of a special processor for the language LISP, called SCHEME, which has utilized regular arrays of memory and PLAs for control and computation, with a minimal amount of random logic. Unfortunately, it has still been necessary to generate much of the interconnect by hand. Such an approach, however, is indicative of the kinds of software-oriented techniques that can be effectively utilized to design custom architectures very quickly. To the extent that these architectures utilize very regular components, such as memory arrays and PLAs, they can be quite efficient and there is an undoubtable tendency for designs of even commercial microprocessors to utilize these regular structures to cut down design time by relieving the amount of complexity that has to be specified by the designer. The use of these regular structures is also useful in delaying the binding time of various design decisions, including the instruction sets of modern microprocessors. For example, it is possible to design a microprocessor completely except for specifying the details of microcode decoding as executed by a PLA. These details may be specified quite late in the design, thereby providing the designer with a substantial amount of flexibility and allowing the circuit design to proceed in parallel with low level architectural decisions.

Modern compilation strategies can also provide for a great deal of parallelism in the design. For example, if the functionality to be implemented is best represented in terms of a multiple set of processes, then the control

for each of these processes can be dedicated to a separate finite state machine, separately implemented on the chip. This is a relatively easy and straightforward thing to do in custom IC design. These separate control processes can easily, however, manipulate the same data path structure, which can also employ a great deal of parallelism. The data path of a modern central processing unit can be thought of as comprising a register array, together with an arithmetic logic unit. Typically, operands are fetched from the register array and then transformed to produce one or more results, which are then inserted back into the register array. In the case of a custom design, however, it may be possible to provide special arithmetic logic unit capability for particular sets of registers within the register file. For example, if it is desired to increment a particular register in the register file, then an incrementer can be provided locally, directly adjacent to that register with its own local busing arrangements, independent of the overall busing provided for the register file. Similarly, if one wishes to compare two registers in the file, then a comparator can be inserted between the two registers, together with local busing so that this comparison can proceed independently of other operations involving the register file. Techniques are currently available for generation of such distributed arithmetic logic unit register files that can provide a large degree of parallelism. This is, of course, another example of delayed binding in the design of processing units, since the provision of this specific capability is task dependent and is only feasible once the detailed nature of the processes to be performed is available. Such techniques are only valuable, of course, if the designs can be created quickly in response to a particular functional requirement and this is the great advantage of modern

silicon compilation techniques. There is, however, a penalty to be paid by these techniques, since the area consumed by contemporary compiled circuits is approximately 2-3 times the area consumed by a good manual design. Nevertheless, this penalty may be acceptable in a wide variety of cases and there are a substantial number of research efforts currently being devoted to improving the area utilization, speed of performance, and power consumption of compiled designs.

Another necessary component for modern compiled systems is the ability to both place and interconnect the components of a system efficiently and rapidly. The canonical problem which contemporary research faces is the need, given a set of rectangles of arbitrary aspect ratio with interconnect on all four sides, to place and interconnect these modules in minimal area. This is a difficult combinatoric problem and initial solutions are currently becoming available. As experience grows, we can expect by the middle of this decade to have capable systems that will allow us to conveniently juxtapose a substantial number of orthogonally related rectangular modules on a circuit very quickly and in minimal space. To the extent that these techniques are incorporated within compilation techniques, greater efficiency can be expected, although the packing strategies used are not necessarily compatible with the canonical architectures typically employed by compilation procedures.

There has been a large amount of borrowing of software compilation techniques into the VLSI design silicon compilation practices. Thus, the recognition of common subexpressions, the elimination of dead code and a variety of other techniques familiar to the software compiler designer are readily employed in the hardware area as well. It is well to remember, however, that

unlike the software compilation approach where the target architecture is fixed, in the case of silicon compilation, it may be possible to generate a substantial variety of target architectures providing a number of different space/time tradeoffs. The analogy between software compilation and hardware compilation is thus by no means perfect, but we may expect a continuing tendency to unify systems through simultaneous design of hardware and software. Frequently, the decision as to what capability to provide in hardware vs. software can be made on performance grounds and system design of the future should certainly contemplate such a unified approach. Certainly, contemporary VLSI design software does not contemplate such a unified design practice, but within the next few years we can expect to see such an approach emerging.

It is well to remark at this point, that the techniques which we have been describing that transform a high level specification into low level mask specifications avoid the usual steps required in custom integrated circuit design devoted to ascertaining the well-formedness of the circuit. For example, topology tests, logic simulation, design rule checking and timing simulation are all procedures that do not necessarily have to be applied to a design generated through silicon compilation. That is to say, if the procedural techniques for generating the low level mask information are correct, then there is no need to ascertain the well-formedness of the resulting structures. Since, however, we can expect this software to introduce its own errors, conservative practice will require the use of these well-formedness checks for some time to come. Thus, the so-called "correctness by construction" approach can be taken seriously only by those who believe that it is possible to generate a program that will also be correct by construction.

In this lecture, we have given a view of the various techniques for achieving rapid circuit design through the use of program generation of circuit elements together with interconnect procedures. While these techniques can often lead to rapid design time, they are not very well optimized for performance at present. In the next lecture, we will deal with these performance conditions as well as the structure of overall design systems.

Discussion

The discussion concentrated on the appropriateness of using design rule checkers in addition to higher level design tools. **Professor Randell** thought it the wrong level of abstraction - rather like checking the output of a compiler for program bugs. **Professor's Allen and Sequin** held that it was important in practice, both because the design tools themselves might have bugs, and because their output might be mixed with hand-produced layouts. The use of hierarchical design rule testing in which boxes known to be correct were not re-tested was approaching, but is still a few years off.

LECTURE 3

PERFORMANCE OPTIMIZATION OF VLSI ARCHITECTURES

In the previous lectures of this series, we have discussed the many factors affecting the choice of representations at various levels in integrated circuit design, as well as the transformational process affording connectivity between these representations. We also discussed how the use of standard canonical forms could be used to maximize the degree of regularity of designs and hence minimize the number of individual distinct circuits that have to be designed manually by the designer. This led naturally to a consideration of programming techniques for the conversion of input functional specifications to output mask layout representations. These procedural techniques can be readily parameterized and hence used for a wide variety of structures that are regular, but nevertheless customized for the individual application. While these software techniques are highly valuable and can generate correct layouts very quickly, the resulting circuits often consume more area and operate in more time than is desirable. We may regard as one of the major challenges of contemporary integrated circuit design practice, the need for retaining the efficiency provided by procedural techniques, while simultaneously providing the means to optimize these designs along the traditional performance dimensions of space, time and power. At the outset, it well to remark that perhaps the most effective means for performing this optimization is through process innovation, a technique that the custom integrated circuit designer usually does not have at his disposal. Nevertheless, the change from a one-level-metal technology to a

two-level-metal technology can result in area savings of a factor of two, and the use of novel self-aligned contacts that can be used over active device area can add yet another factor of two in area savings. The scaling down in overall dimensions of modern IC processes not only implements a given circuit in less area, but also provides for a substantial increase in operating speed. Currently, there is a trend toward a larger number of CMOS circuits, generally implemented in some sort of bulk technology, and we are approaching a period when these circuits give acceptable performance in time and space and of course provide very low power dissipation. These remarks are meant to indicate that the integrated circuit designer should be aware of the tremendous influence that the choice of technology has on design practices. For example, the researcher who is designing routing programs for the automatic provision of interconnect in large scale circuits should realize that an additional level of metal interconnect will have profound influence on the routing algorithms under development. Thus, it is important to not necessarily regard the technology as a given, but to understand where technology improvements can improve the design process in ways that could not be equalled by innovative design practice in a less aggressive technology.

When discussing integrated circuit performance, it is well to distinguish between architectural performance and circuit performance. We have discussed in the first lecture of this series, the limited success obtained through the use of constraint representations for algorithmic competence. We remarked at that time that it is currently not possible to formally separate algorithmic competence from algorithmic performance. Nevertheless, it is possible to manipulate a given algorithmic formulation along various performance

dimensions in a way that is guaranteed not to perturb the underlying competence of the algorithm, even though that competence is not explicitly exhibited at any stage of the process. A recent theoretical study has shown that the means for preserving transformations is critically dependent upon the notion of conflict. Thus, if we start with an initial description of the algorithm in some sort of source level hardware description language, then we can examine the statements in that language to see whether or not there is sequential conflict between them. That is if in a sequence of n such statements, none of them depend on any of the others, then they may be done in parallel. On the other hand, if a sequential dependence is observed, then the effect of this dependence must be preserved by the implementation. The on-line detection of such sequential conflict has been implemented in a few high performance computer architectures, but general techniques for exploring architectural tradeoffs prior to fabrication are not in general use at present. We can expect, however, that in the near future, techniques for transforming between various classes of statements in a high level hardware design language, as well as the manipulation of iterative constructs in these languages, can be reliably manipulated to exhibit various performance tradeoffs. A good practical example of these techniques has recently been demonstrated for the special case of masterslice or gate array implementation, where the target architecture is somewhat constrained by the need to implement all logic in terms of an interconnected set of NAND gates. The basic idea of this approach has been to generate a naive implementation automatically from the given input functional specification which consists of a set of logic equations. The second step in the process is to incrementally improve the naive design which was obtained automatically, by

applying local transformations selected by the designer to vary the performance along dimensions that are important to the practical implementation. These transformations are provided to the designer for his possible use and are guaranteed not to perturb the correctness of the design from the functional point of view. That is to say, a design system is being provided for the implementer that constrains his degrees of freedom in manipulating the design, such that the original functionality is always preserved. These transformations provide for explicit control of technology-specific factors. These include timing, fan-in and fan-out, rules for testability, and constraints on the numbers of gates and pins on the resulting circuit. The transformations embody local techniques that are very familiar to compiler designers, such as the identification of common subexpressions, and dead code elimination. In fact, the originators of these techniques all come from a software, rather than a hardware, background. Not only is it possible to perform these transformations under designer control, but it can be expected that any given design can be automatically inspected by programmatic means for satisfaction of various performance constraints and then a set of heuristics used to select the set of transformations needed at the next iteration of the design. Finally, of course, by whatever means are chosen, a satisfactory design is obtained and the interconnect between the set of NAND gates on the chip is bound according to this final satisfactory specification and fabrication proceeds. This is to be regarded as a good example of the use of performance transformations within a constrained design environment. We can expect to see many such additional instances in the future, perhaps in the context of less constrained target architectures.

A frequently used technique for implementing finite state machines is to utilize program logic arrays for the combinational logic part, while providing feedback between the output and the input for the representation of state information. Nevertheless, these PLA based finite state machines are often quite large and it is desirable to try to minimize the area of the logic arrays. There are a number of formal techniques for such minimization, based on classical techniques from logic, but the designer can often utilize a well-developed set of heuristics for this purpose. For example, it may be observed that one output column in the PLA is the complement of another, so that one of those columns can be derived through an inverter from the other. Similarly, we often encounter the presence of state-dependent inputs, such that some inputs are only required in one set of states, while other inputs are required during other states. When this is the case, the inputs can be multiplexed into the input AND array in a state dependent way. Finally, it is often possible to choose the encoding of the states in such a way that rows in the PLA can be eliminated and this technique can have dramatic savings. It is not uncommon for the result of applying these various heuristic techniques to yield a 20% saving in the overall area of the design. It is important to realize, however, that there is an inherent tension between two tendencies displayed in these techniques. On the one hand, we generally prefer to have regular structures that do not have ad hoc appendages attached to them, since the latter imply additional design time and perhaps difficulty in ascertaining well-formedness. On the other hand, as we have just seen, sometimes the addition of a small amount of external circuitry, in addition to the basic regular structure such as a PLA, can provide dramatic savings in area, as well

as timing. The designer is thus faced with the need to decide how much regularity should be preserved as a function of the savings obtained by departing from that regularity. For this reason, designers have tried to group these ad hoc heuristic changes into well-formed canonical classes so that, while they do perturb the original basic regular design, they do so in constrained ways that preserve many of the desirable properties of the circuit. Thus, there is a constant battle between the various optimizing factors in the design, including the desire for regularity, modularity, hierarchy, minimal area, maximum speed, and mimimum power. The design of custom integrated circuits, even with the advent of a wide variety of helpful computer-aided design software, is still very much an engineering enterprise and it requires a great deal of judgment on the part of the designer, as to the proper selection of circuit elements. It may be that in time we will be able to capture some of this engineering expertise in expert programs, utilizing techniques from artificial intelligence. For the present, it seems best to provide a set of techniques, constrained in such a way that the original functionality is preserved, but providing freedom to the designer to explore the design space and discover the consequences of various design decisions quickly and economically. This approach to custom integrated circuit design seeks to relieve the designer of the need to specify a great deal of low level artwork detail, but instead provide the designer with the need to specify only those decisions that are best done (under current understanding) by a human designer.

There are sometimes surprising relationships between logical formulations and their consequent utilization of area. An example of such a relationship is the use of so-called two-bit decoding with PLAs. In a standard

PLA, each logical input is supplied to the array in both its true and complemented form. Thus, two input variables account for four logic values that must be bussed across the input AND array. It is possible, however, to decode these two input values, using standard decoding techniques, such that the functional logical load of the four wires previously mentioned is substantially increased. If we examine the sixteen possible variations of the logical values on these four wires, we find that if the two inputs are not decoded and merely supplied in their true and complemented forms to the four wires, then there are seven false conditions and one don't care that are contained in this set. This means that of the sixteen possible logical configurations that could be enforced on these four wires, one-half of them are not possible and hence, we may think that the space consumed by these four wires is not doing as much logical work as it might. A way to improve this utilization factor is to employ the two input decoding. Under such a regime, there is only one false condition out of the sixteen possible cases and still one don't care. A number of new logical forms are generated automatically in this way, including the provision of the exclusive OR between the two input variables. This is a very interesting design example since it shows that by external manipulation of the input variables to a logic array, the area utilization in terms of logic functionality carried by a particular part of the circuit can be increased dramatically. In the design of central processing unit circuits, where matching is often needed and hence, exclusive ORs required, such techniques can be expected to provide a large gain in effective area utilization. Furthermore, since the two input decoding can be readily adapted within the driving buffers for the logic wires in the array, the cost in increased area outside of the array is minimized.

Lastly, we should remark on the growing tendency to build large systems on an entire wafer, rather than by means of interconnecting a number of discrete integrated circuits. When one attempts to build an entire system from a wafer, however, the yield of the entire wafer can be expected to be zero, unless redundancy and petitioning techniques are imaginatively utilized. Research in this area is very new, but already it is clear that a variety of new design considerations will have to be introduced, including the study of optimum partition size as a function of the defect distribution in a given technology, as well as the need to provide flexible routing schemes for defect avoidance. These considerations can be expected to have substantial impact on the class of algorithms that attempts to exploit locality, such as systolic systems. On a wafer based system, it is not possible to guarantee physical proximity of logically adjacent elements due to the random occurrence of defects. For this reason, it may be necessary to substantially revise the characterization of these locally organized systems to reflect the system level technological constraints.

Not only is it important to provide for the manipulation of circuit and architectural performance on the circuit itself, but it is also desirable to provide for the designer high performance computational environments for not only the synthetic task of generating the modules of a given system, but also for the more analytic techniques used to ascertain the well-formedness of the constituent modules and their hierarchical relation in the resulting system. In the early part of this decade, we are witnessing a rapid growth of the design of work stations for custom integrated circuit design. Typically, these installations are personal computers, providing mainframe CPU capability and

large address space in order to deal with the large number of artifacts in integrated circuit design. Desktop 32-bit architectures of this kind are rapidly appearing and seem well suited to this need. They should also provide for high resolution black and white and color graphics and a large amount of bulk storage capability (several hundred megabytes) directly attached to each individual work station. These stations can be connected together by a local network to a large file server, that can serve as the main library for all designs, as well as a controller for input/output services, including plotting and listing. Thus, the custom integrated circuit design environment of the future will include a collection of such work stations, interconnected so that individual designers are part of a larger community that can share a wide variety of computational resources, as well as previously designed circuits. One can thus imagine a large circuit being designed by a team of workers who can readily communicate their progress to each other. It will be important, however, to enhance these systems with the provision of special hardware for certain computationally intensive tasks, such as various kinds of artwork analysis, including design rule checking. Currently under study are a variety of techniques for providing special hardware to speed up such checking by as much as two orders of magnitude. Such techniques can be expected to allow the designer to interactively generate and analyze a large variety of structures without undesirable delay. By the middle of this decade, then, we can expect the integrated circuit design community to have at its disposal a set of techniques for rapidly, correctly, and economically generating very impressive integrated circuits in a way that allows the designers to focus on those representational issues that are important to the initially given

specifications. In this way, the huge potential provided by the ever improving modern integrated circuit technology can be exploited and the fears of excessive design time expressed in the context of previous design styles can be confidently set aside.

Discussion

Professor Dijkstra queried the need for logic simulator, but **Professor Allen** was emphatic as to its usefulness. There was some discussion with **Professor's Tanenbaum** and **Michaelson** about how far one could take the analogy with software, as in software one re-uses resources in time, but in hardware one must replicate in space. A system must handle both if the designer is to be able to choose solutions appropriately.

Professor Allen confirmed that the placement programs he had mentioned were all written in LISP, and emphasised the importance of having all the design tools mounted on one machine. At Berkeley this was eased because of the standardisation on VAX machines running UNIX, but at MIT they had to contend with a DEC20 and five different programming languages, besides the problems of an 18-bit address space.