R. L. Golden
P. A. Latus
P. Lowy

Design Automation and the Programmable Logic Array Macro

The Programmable Logic Array (PLA) macro is a physical structure which simplifies LSI chip design while yielding high density and good performance. In addition, the inherent order and regularity of this structure provide opportunities to speed design through automated logic documentation, design verification by computer simulation, and computer-automated physical design. In this paper a chip design methodology is described which is based on the use of PLA structures (or macros) within a chip. Logic functions in array form are specified in a compact notation that is automatically converted either to array personalization patterns or to conventional logic blocks for input to existing checking and testing software. Simulation of any logic array is performed by a single program subroutine operating on these patterns. In addition, the simple, regular nature of the logic array lends itself to automatic generation of the layout geometries necessary to actually build the array on a silicon chip. Programs developed for these purposes and the PLA macro design procedures are described in the context of an engineering design system.

Introduction

With Large Scale Integration (LSI) technology, a typical chip, such as a microprocessor, may contain thousands of transistors, and the chip may be fabricated from masks made up of hundreds of thousands of shapes. A great deal of engineering judgment, skill, and effort is required to meet design objectives; yet, to design such a chip without errors requires a high degree of automation. This paper describes a chip design methodology that permits a blend of manual engineering and automated design, and that is based on the use of programmable array logic structures (PLAs) within the chip.

Figure 1 illustrates the general design flow or methodology that is followed in designing a chip. In the preliminary design phase, logic and circuit designers determine the proper technology, *e.g.*, bipolar or field-effect transistor (FET), to meet the design specifications and the cost and schedule objectives. Preliminary designs are then undertaken to determine how large the chip must be. The preliminary design is accomplished as follows.

A design based on a particular data flow and clocking method is chosen. Preliminary circuit design is undertaken, and—on the basis of this early design—the circuit areas, their placement, and their interconnections are estimated. These steps proceed iteratively until a design that gives the minimum area and that satisfies objectives is achieved.

Once the preliminary design is chosen, logical and physical designs are begun. In Fig. 1, the logic design steps are illustrated on the left and the physical design steps are shown on the right. In some methodologies, logic and physical designs are accomplished in parallel as illustrated, while in others the physical design is delayed until logic verification is complete.

Logic design is generally comprised of the following steps (numbered items are keyed to those on the left in Fig. 1):

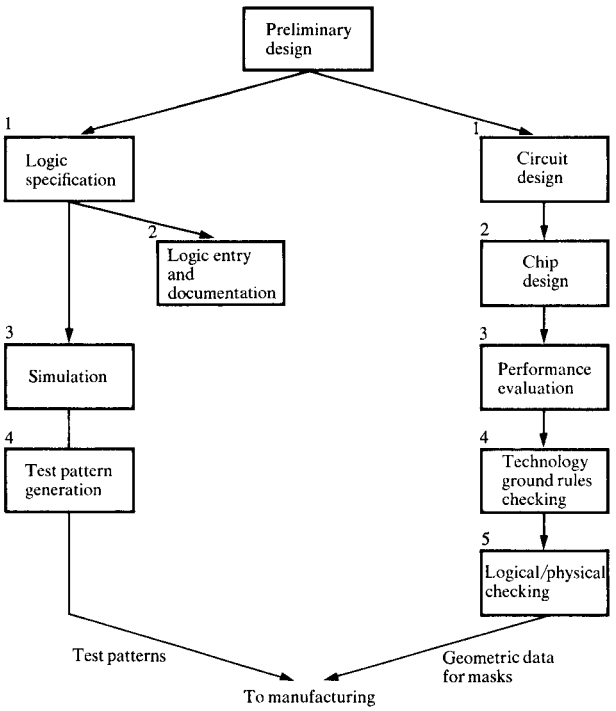

Figure 1 Methodology of chip design.

1. Transformation of the specification into logic primitives (AND, OR, INVERT, etc.) and storage elements (LATCHES, MEMORY, etc.); the elements and their interconnections are recorded on hand-drawn diagrams. The objective of this step is to obtain an accurate transformation using a minimum number of elements.
2. Entry of this logic into a computer data base. IBM has developed a standardized logic specification language for this purpose; this language will be described in some detail later in this paper. Logic descriptions are either encoded and entered into the computer system or are entered graphically. Computer-generated drawings are then produced from these.
3. Simulation of a chip model derived from this data base to verify logic function. The IBM Engineering Design System includes a software logic simulator that permits users to apply inputs to the software model and to obtain resulting outputs. This simulation is based on both function and timing data, and it can operate on primitive models or on higher-level models such as a memory or a logic macro. Logic may be simulated separately or in interconnected groups.
4. Creation of test patterns from the same model. These test patterns are used after manufacturing to test the chip. It is desirable to verify that every input, output, and circuit on the chip functions properly before the chip is placed in use. Testing of circuits not externally accessible is achieved by a combined hardware-software approach. First, every latch element is designed so that, in addition to its normal logic function, it is connected in series with similar elements to form continuous shift registers that are externally controllable for testing. This method, known as "Level Sensitive Scan Design" (LSSD) [1], allows test patterns to be shifted in and applied to combinatorial logic. The results are loaded into the shift registers and then are shifted out for analysis. Second, the IBM Engineering Design System includes programs [2] that generate the proper test patterns and the expected results for chips designed in this manner. In practice, for chips of this kind, over 98% of all circuit inputs and outputs are tested.

Physical design is basically comprised of the following (numbered items correspond to those on the right in Fig. 1):

1. Design of circuits for logic primitives and storage elements, and the creation of mask shapes in a graphic language for these circuits. Circuit designs are verified analytically to operate under worst-case conditions using the Advanced Statistical Analysis Program (ASTAP) [3]. Mask shapes are created using a system such as the IBM Interactive Graphics System IGS/370 [4].
2. Chip layout, *i.e.*, the interconnection of these circuits on the chip. This may be accomplished in a number of ways; for example, by using IGS/370, or a digitizing system such as an IBM 1130 system with a digitizer-plotter, or automatic wiring programs such as those described in [5].
3. Evaluation of circuit path delays to ensure that the required performance is achieved. One method for accomplishing this is to calculate delays for individual circuits as a function of load; compute the load automatically from the mask data; compute the delays for the circuits on the chip based on the preceding; and finally, repeat the simulation using these delays.
4. Checking the chip layout against technology ground rules for minimum line spacing, maximum parallel line lengths, etc. The IBM Unified Shapes Checker program [6] is used for this purpose. The program incorporates a high-level language that permits users to code checks for their applications.
5. Checking the chip layout against the logic design for correctness of circuit layout topology and circuit interconnections. Since both logical and physical data bases exist, it is possible to compare the two by automated means; this is known as logical/physical checking.

After both logical design and physical design are completed, the geometric data and test patterns are sent to the manufacturing facility, where the chip is built and tested.

Design approaches

The methodology just described can make use of design automation to varying degrees. Unfortunately, the degree to which automation can aid in the design process decreases when the chip and circuit layout must be optimized for maximum circuit density, and for performance at minimum power. This optimization may partially consist of the following:

1. Changing the layout of mask shapes for a particular circuit so that the circuit will fit into a smaller area and still retain its electrical characteristics.
2. Designing unique circuits for particular applications, *e.g.*, an internal driver circuit for heavy loads.
3. Changing the routing of circuit interconnections to reduce capacitance.

Clearly, these steps are manual and require expertise and time; as a result, they can only be justified for large-volume production. A chip designed in this way is said to be tailored or customized, and this first approach is known as *custom design*.

A second approach to LSI design that reduces development cost and time at the expense of increased silicon area is the *master slice* concept. In this approach, circuits are predesigned in an array; these basic circuits or cells are then interconnected with metal according to the logic design. The designer works in a constrained environment since the circuit design is fixed, as are the number of cells, fan-out limits, and other design parameters. These constraints, however, make it practical to automate the physical design process. Programs have been written for both the placement of cells and the wiring of the entire chip [5, 7]. The circuits on the master slice are of a fixed design; therefore, cell delays based on the worst placement can be calculated before the actual physical design. After the logic has been verified by simulation, a performance analysis is obtained by once again simulating the logic verification patterns, before physical design with these estimated delays. When physical design is complete, delays based on the actual interconnective wiring are used in simulation to verify performance.

A third approach that can significantly reduce design time is to utilize PLA chips of the type described by J. C. Logue *et al.* [8]. In this *PLA chip* approach, the designer must work within the bounds of a fixed number of product terms, inputs, outputs, and feedback elements, and with a fixed chip delay. Since the PLA chip layout is fixed, the remaining physical design effort consists only of adding or deleting devices and connecting the proper partitioning and output circuits as chosen by the logic designer. That these tasks are easily automated is evident from the fact that devices which form the crosspoints within the array are located in the same relative location as the logic designer's symbol for the device. As in the master slice approach, the silicon area is not as fully exploited as it is in custom design.

PLA macro approach

Unfortunately, the design constraints imposed by the master slice or PLA chip approaches are in many cases too severe to produce a cost-effective product. Conversely, the development cost for a custom design may be too great to warrant its production. To overcome this dilemma, we adopted a compromise approach known as *macro design*.

Macro design is based on the observation that circuits with a high degree of logical connectivity will fit closely together when physically implemented. These aggregations of circuits are called *macros*. Examples of these macros, shown in Fig. 2, are a multiplexer-register, an EXCLUSIVE-OR tree, and a most important macro—the PLA. The PLA macro discussed in this paper consists of input partitioning circuits, an AND array, and an OR array. It combines the PLA chip's ease of automation with the flexibility of custom design regarding the choice of the number of inputs, outputs, and product terms, as well as the control of input-to-output delay.

A microprocessor was designed by our group using PLA macros. The chip consists of 35 macros, 19 of which are PLAs, and these account for half of the chip circuit area. The remaining 16 were assembled from smaller macros such as those shown in Figs. 2(a) and (b), and are referred to as *random logic* macros. Some of the custom design techniques described earlier were used to optimize these random logic macros. The choice of PLAs and random logic macros resulted in high circuit density and good performance. To illustrate, a study was made to compare this design with another design using the same technology but a different design approach. The devices in each design are totaled and the number of equivalent NOR circuits of fan-in 2.5 is calculated from the total. The results, illustrated in Table 1, demonstrate that the macro design approach is superior to the master slice approach by a factor of 1.8 in circuit area, and by a factor of 1.7 in power for the same or better performance.

Figure 3 illustrates how programs for PLA macro design, which will be elaborated on in more detail in the remainder of this paper, fit into the methodology of Fig. 1. The first step is the logical design of the PLA macros.

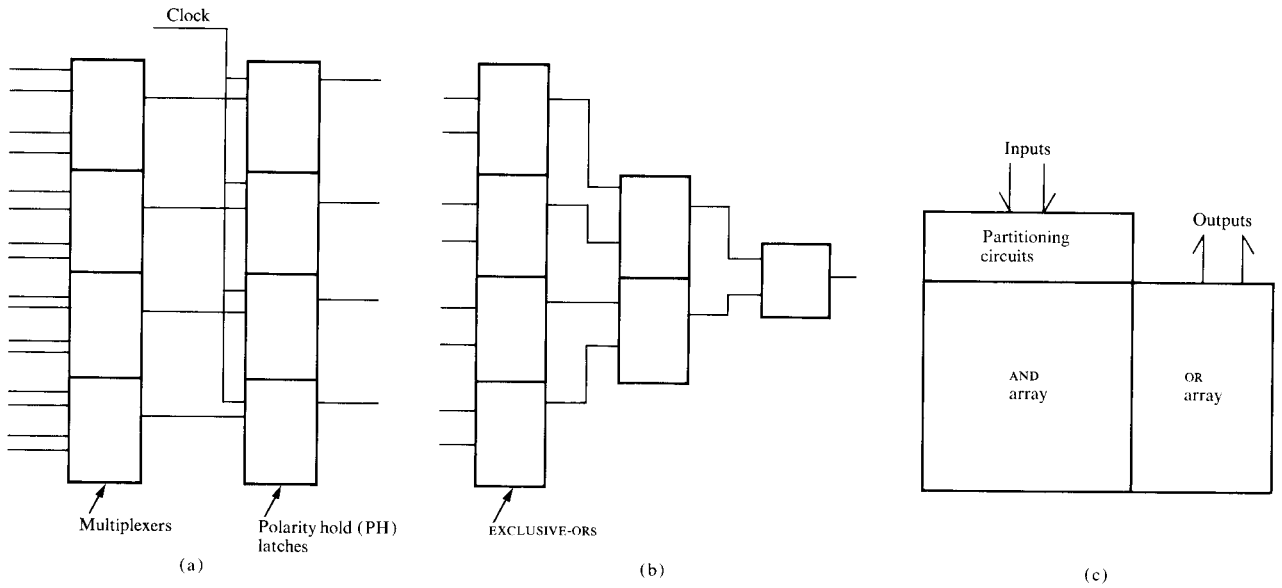

Figure 2 Examples of macros: (a) multiplexer-register macro; (b) EXCLUSIVE-OR macro; (c) PLA macro.

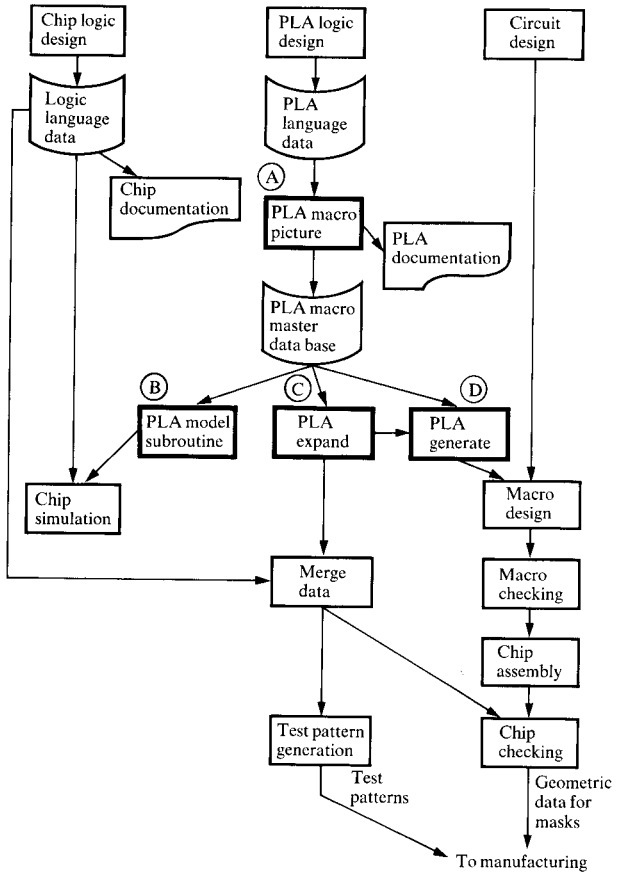

Figure 3 Detailed design flow.

PLA macro logic specification and documentation

The regular structure of the PLA macro makes a tabular format convenient for specifying and recording the logic function. The format used is a matrix in which the columns are the AND-array input and the OR-array output lines of the macro, and the rows are the product terms. The logic functions of the PLA macro are defined by the following symbols:

1. In the AND array:
   - I -match on a logical *one*.
   - O -match on a logical *zero*.
   - · -match on either a logical *one* or *zero* (don't care state).
2. In the OR array:
   - I -set output to logical *one* if any input product term is selected; or to logical *zero* if no input product terms are selected.
   - · -ignore this product term.

The AND array symbols are for input partitioning circuits which are one-bit (or one-input) decoders. Additional symbols specify partitioning circuits which are two-to-four-bit (or two-input) decoders [9], but they will not be defined here. An example of PLA logic documented in this notation is shown in Fig. 4.

The matrix is numbered from left to right, and from top to bottom, as indicated on the top and left sides of the

figure. Consider the third row. Product term 3 will be selected when column 5 is a logical *zero* and column 6 is a logical *one*. In this event, output columns 13, 15, and 16 will each be logical *one* conditions.

The PLA macro logic specification describes the logical functions within the macro itself. However, the interconnections of the PLAs to the other macros on the chip, and the circuit connections within the other macros, must also be specified to fully describe the logical function of the chip. This global representation of the chip uses the IBM standardized logic specification language mentioned previously to form a data base for documentation, simulation, and test pattern generation.

In our use of the language, a primitive, a register element (latch), or a PLA macro is represented as one logic block. Each block is given a unique identification and is described by one statement. Each interconnecting net is given a symbolic name. A simplified example of this representation is shown in Fig. 5. The PLA macro block description includes a pointer (INTR) to the macro logic specification table (Fig. 4). Thus, the periphery of the PLA macro is stored with the chip's logic description, and the internal logic of the PLA macro is stored separately in a table. This approach permits changes to the logical function of the macro without changing the global representation, as long as the PLA macro input and output nets remain the same.

The one-block and one-table representation for the PLA reduces the documentation over an equivalent random logic approach. The 19 PLAs were shown as 19 blocks supplemented with 19 tables. The equivalent logic, assuming that one were to map the PLAs directly into primitives, *e.g.*, one product term = one NOR, would be over 1000 blocks.

A convenient language was also developed for the PLA logic designer. An abbreviated format specifies only the significant (I and O) table entries of the macro by column for each row (Fig. 6). For example, the third row in the array section of the data, corresponding to line 3 in Fig. 4, indicates that column 5 is O, followed by an I in column 6. The next column specified is 13, which contains an I, followed by a don't care term in 14, I in 15, and I in 16. This PLA macro design language also includes features for describing partitioning circuit types, *e.g.*, one-input or two-input partitioning, for assigning symbolic net names to the inputs and outputs, and for recording descriptive comments associated with the logic specification.

A program, Macro Picture (block A of Fig. 3), was developed to read this abbreviated format specification and

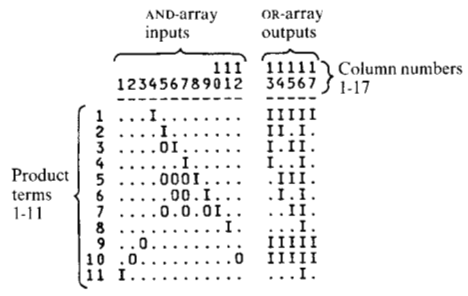

Figure 4 Example of PLA representation.

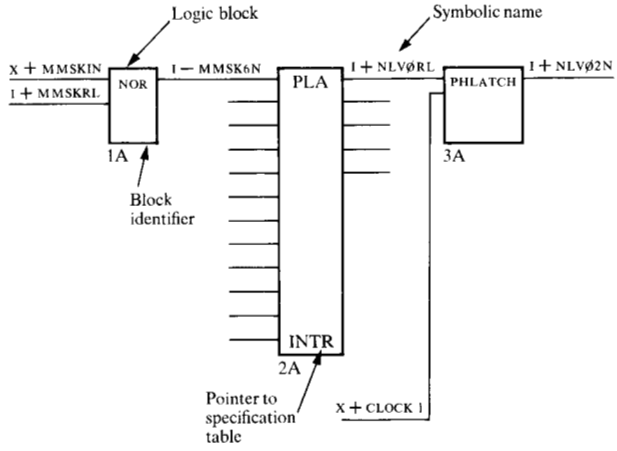

Figure 5 Example of standardized logic representation.

Table 1 Results from design approach comparison study. (Results shown as ratios.)

	Macro chip / *Master slice chip*
Equivalent NORs/cm^2	1.8
Average delay/circuit	0.8
Average capacitance/circuit	0.5
Average power/circuit	0.6

to produce a printed picture of the PLA macro logic in a tabular format similar to that shown in Fig. 4. In addition to providing design documentation, this program also produces a data file to serve as a PLA macro master data base for design steps that follow, including simulation,

```
HEADER   INTR;
DIMENS   INPUTS =012,
         OUTPUTS=005,
         TERMS  =011,
         ANDOUT =000;
GENERAL  BP=(      ),
         TC=(005,006,007,009),
         I =(001,004,008,010,011),
         D =(002,003,012),
         SL=(    );
ARRAY    4I13IIIII,
         5I13II.I,
         50I13I.II,
         7I13I..I,
         5000I14III,
         600.I14I.I,
         50.0.0I15II,
         11I16I,
         3013IIIII,
         20120IIIII,
         1I16I;
NETNAME  1 ='I-MMSK6N',     2 ='I-MCPCON',     3 ='C-LCHKON',
         4 ='I+LVL0AN',     5 ='I+LVL1AN',     6 ='I+LVL2AN',
         7 ='I+LVL3AN',     8 ='I+LVL4AN',     9 ='I+LVL5AN',
         10='I+LVL6AN',     11='I+LVL7AN',     13='I+NLV0RL',
         14='I+NLV1RL',     15='I+NLV2RL',     16='I+WAITRL',
         17='I-REQ0RL', 12='C+RSETCN';
PICTURE  01,C=(1-12,S,S,13-17,S,S,),
           R=(1-11,S,*,5);
END      INTR;
```

Figure 6 Abbreviated format example.

Table 2 Simulation run time and data volume.

	All primitives, no modeling with subroutines	*Only latches modeled by subroutines*	*PLAs and latches modeled by subroutines*
Number of logic blocks	6363	1846	845
Chip model generation time, 370/168 (min)	27	11	4
All events trace data set size (13 000-byte disk tracks)	2040	197	122
Simulation time, 370/168 (min)	45	12	12

physical design, and test pattern generation. The programs used in these steps can easily read this data base because of its fixed tabular format, which minimizes the processing needed to read the data.

The Macro Picture program also provides syntax checks, such as verifying that only valid array symbols are used. The program guarantees that further processing can continue only when the logic specification has correct syntax. This is accomplished by recording a success or failure code in the data base produced.

PLA macro logic simulation

After the master data base has been successfully created for each PLA macro and the global chip logic has been correctly documented, the entire chip can be simulated to verify that it performs the intended function before hardware is designed and built.

The logic simulator described earlier can be utilized for chips containing both primitive logic functions (AND, OR, INVERT, etc.) and higher-level logic entities such as a memory or a macro. Nonprimitives are simulated by allowing macro modeling subroutines to be linked to the main program. The system is therefore well suited for simulating a design containing PLA macros.

A single subroutine (block B of Fig. 3) was developed to simulate the internal function of any PLA macro. The subroutine function is controlled by each individual PLA macro master data base, so that the PLA internal logic can be changed without altering the global logic or the subroutine. In addition, this subroutine produces diagnostic tracing of the simulation activity within each macro. The use of this PLA simulation program combined with the existing computer simulation system allows a PLA macro to be simulated by itself, or with any collection of logic blocks defined by the logic input language, including the global chip function. In addition, since the circuit design and the layout of the PLA are known, delays for the PLA macros can be calculated. The simulation subroutine and system can include these delays in order to identify timing problems.

When high-level models are used in lieu of primitives, larger designs can be simulated with less computation than for an all-primitive design, since fewer blocks and fewer transitions need be simulated. Table 2 shows the results from a comparative experiment. Our microprocessor was simulated for 320 instructions using three equivalent logic representations as detailed in the table. The entry "chip model generation" refers to the procedure for converting the logic language to a more efficient data base for simulation. The "all events trace" is a record of each net's transitions.

It can be seen from Table 2 that high-level modeling reduces computer run time and data volume. The advantage of the PLA macro-modeling subroutine is subtle but important. Suppose a logic error in a PLA is uncovered in simulation and is to be corrected. If the PLA were represented by primitives, the original data base would have to be updated, and chip model generation would be required before simulation could begin. On the other hand, using the subroutine, a change to a bit in a PLA is corrected by updating the table and running the picture program (which takes about five seconds of System 370/168 time). No chip model generation is required since the PLA subroutine is controlled by each individual master data base. This procedure saved several weeks during the design cycle of our chip.

Expansion of the PLA to equivalent circuits

When the logic designer has gained confidence in the design through simulation, the symbolic array is expanded (block C of Fig. 3) to an equivalent logic primitive form. Expanding the array serves two purposes. First, it provides input for test pattern generation and logical-to-physical checking programs. Second, the array logic symbol matrix is translated into an array of binary *ones* and *zeros* representing the presence or absence of a field effect transistor. The pattern of *ones* and *zeros* forms a map from which the physical design of the PLA can be constructed. This information is of the form shown in Fig. 7(a) and is stored for later use by the program that actually performs the physical design.

The primitive logic representation of a PLA depends upon its circuit technology and design. The expansion discussed herein is for an n-channel MOSFET technology, where the basic circuit is a NOR; however, the program can easily be modified for any circuit family.

Each AND-array product term and OR-array output column becomes a NOR block, as illustrated in Fig. 7(b). The partitioning circuits are each represented as a single macro block, which is later expanded to one or more NORs according to an expansion definition for each partitioning circuit type. The same logic text language used for describing the global block connectivity is used to represent the internal PLA connections.

The PLA equivalent logic data are merged with the other macros to form a chip data base compatible with the test pattern generators, which create fault detection and isolation tests for logic in primitive form. In addition, since this merged data base contains all the connectivity information for the chip, it can be used to verify the connectivity in the physical design.

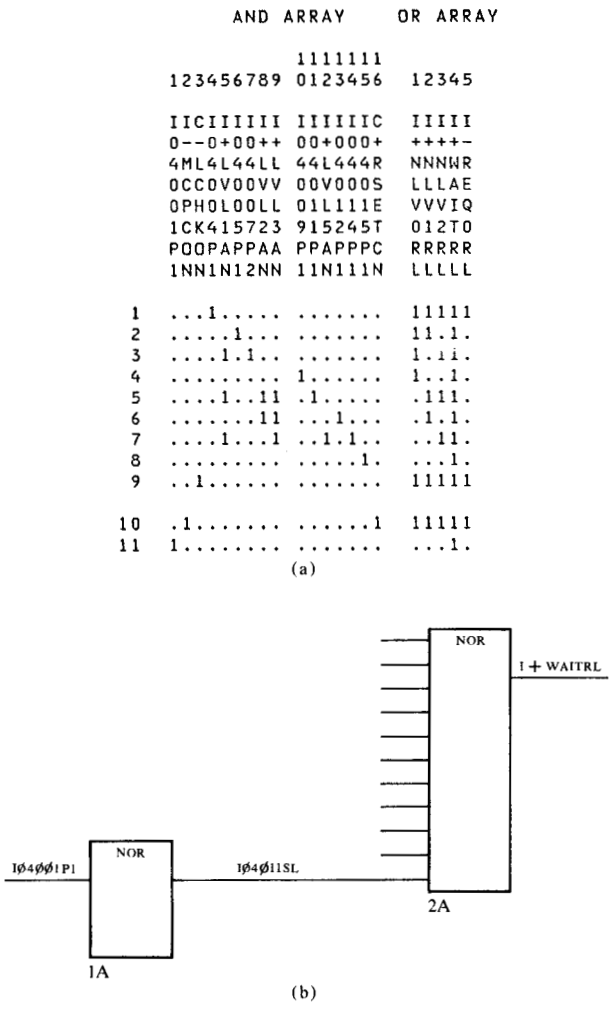

Figure 7 Example of PLA expansion: (a) bit map; (b) equivalent NORs for product term 11, output column 4.

Physical design of the PLA

To understand how the mask data for PLA macros can be automatically generated, consider how the data would be generated manually. The engineer would probably use an interactive graphic computer design system such as IGS [4] to create geometric data. Such a system consists of a central computer, disk storage, and a printer, as well as a graphic video display terminal and a plotter. The geometric data produced with this system consist of rectangles, lines, and polygons. These data are the input to an automatic mask generation machine called a "flasher," which exposes selected areas of a photosensitive emulsion to ultraviolet light, thus building up the desired image on the mask.

In the manual process, certain collections of shapes representing circuits or circuit elements that will be used repeatedly are designed first. These are stored as named graphic entities called "cells." When designing a particular PLA, shapes of unique dimensions are specified individually, while shapes common to many PLAs are added to the design by invoking predesigned cells at the desired locations. Such cells are used for the FET gate at the crosspoint, for the AND-array and OR-array load devices, and for the various partitioning circuits. The invocation of a cell is called a "cell transform." An examination of this task suggested that automation might be practical. We therefore wrote a PLA generation program (block D of Fig. 3) that creates the geometric data for a PLA in the same fashion as the manual design process. The geometric elements unique to a particular PLA, in this case rec-

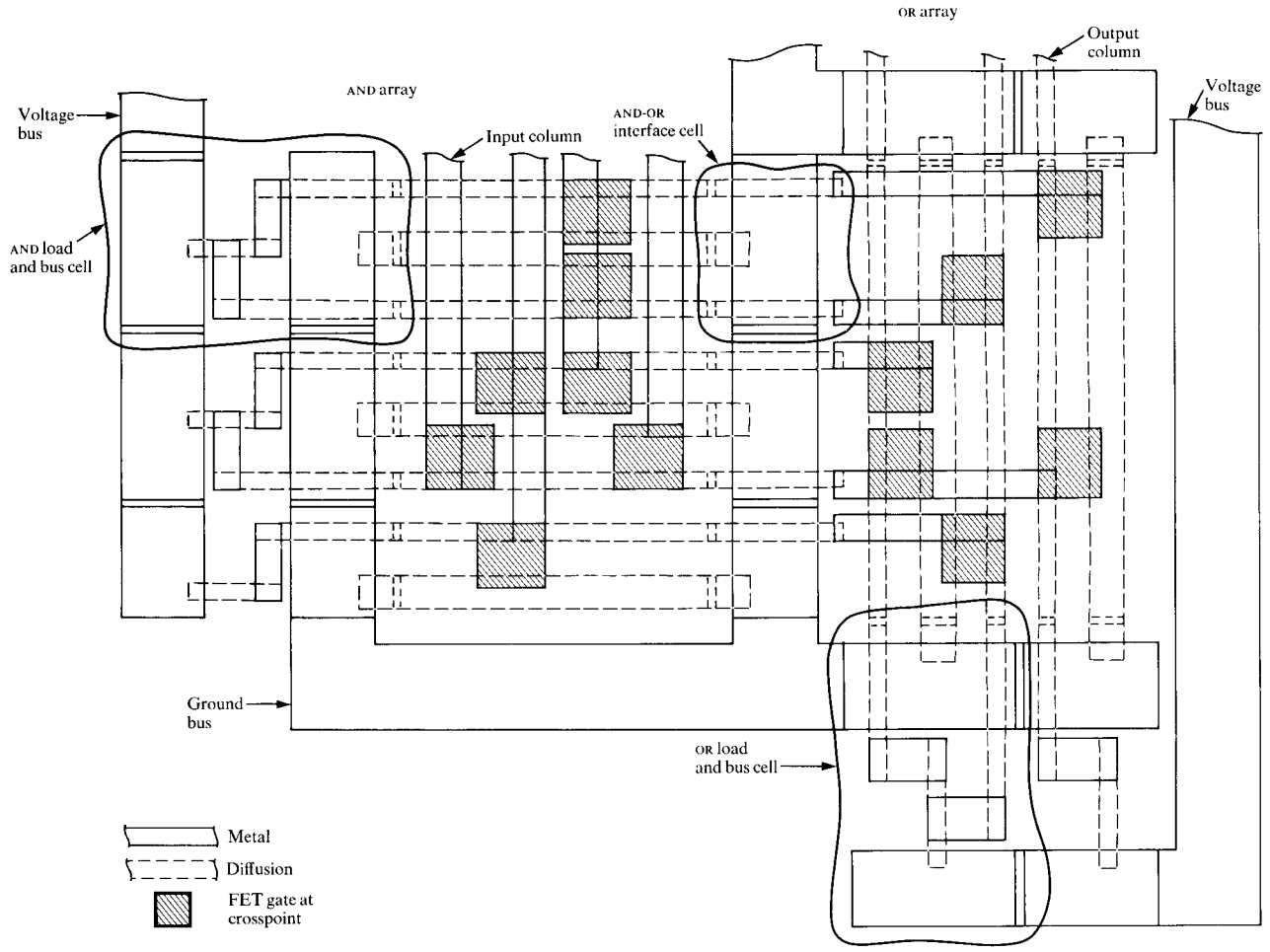

Figure 8 Example of AND-OR layout. Note: Figure not drawn to scale; diffusion-metal connects (contacts) omitted.

tangles representing AND input lines, product terms, and OR output lines, are specified using program-calculated dimensions and locations. These are based on the number and desired spacing of array rows and columns. Identifying labels are applied to rectangular row and column shapes for later automatic cross-checking against the logic interconnection list, which uses the same labels as symbolic net names. In this way, proper interconnection of the physical design can be verified against that specified in the logic design. The program calculates the position to which a needed cell, for example an FET gate or a product term FET load cell, should be transformed, and codes the transform operation to that cell. (The physical design of these fixed cells is left to the designer because automation of this aspect was judged not to be cost effective.) The bit map previously stored by the logic block expansion program is used to calculate the positions where the FET gates are needed to create the required logic function in the PLA. Figure 8 is a sketch of a PLA macro layout as it would be automatically generated. The resulting graphic data are transferred to the graphic design computer as a new cell. The PLA cell is then manually positioned and added to the desired location in the global chip data base for later interconnection with other circuits. This merges the automatically generated graphic data for the PLA with the graphic data for the other macros used on the chip. After the global connections are made, the entire chip data base is checked for technology rule violations and is also checked against its logical counterpart for interconnection errors. After these automatic checks are successfully performed, the geometry data set is ready for mask generation.

Efficiency of PLA macro physical design

How much design time is saved by this automated approach? For the microprocessor we designed, the PLA macros contained an average of 16 inputs, 12 outputs, 28 product terms, and 238 crosspoints. As calculated in Table 3, over 400 operations would be required for each PLA if it were implemented manually on the graphic sys-

tem. Since any manual process is prone to error, at least two design passes would be required, and these would take an experienced designer two to three days to accomplish. The automatic method requires one minute of computer time on a System 370/168. Further, a change to the design requiring the addition of an input, an output, or a product term would be accomplished by re-executing the program. Of the 19 PLAs, 17 were implemented using the PLA Generate program, saving seven to ten man-weeks of effort. Graphic system usage was also reduced.

The design time savings of an automated PLA macro approach *versus* implementing the functions as in a custom design are significant. The 17 PLAs contain 4800 devices, equivalent to almost 1400 NOR circuits of fan-in 2.5. It would take an experienced designer four to six months to lay out, interconnect, and check this many circuits manually, as compared to less than a week using the automatic PLA macro generator.

Conclusions

The programs described in this paper are part of a methodology for designing chips containing PLA macros. Documentation, design verification, and physical design for the PLA portion of the chip are automated; at the same time, however, a considerable degree of design freedom for the chip as a whole is retained. An NMOS FET microprocessor chip containing PLA macros and random logic macros has been successfully designed with this methodology. The automated design of 50% of the circuits on the chip saved one-third to one-half of the design effort required for a totally manual custom design. A comparison of the circuit density and performance of this chip with those of a master slice demonstrated that the macro design methodology results in higher density and less power than the fully automated master slice approach.

Acknowledgments

The input language and the documentation and simulation model programs for the PLA macros were based on prior work at IBM Kingston by D. J. Donaghy, T. G. Foote, and W. Snyders. The authors thank D. A. Conrad for the concept of the PLA generator and for his help in the initial design of the program. The effort of J. Dedrick, who wrote the PLA Macro Picture program, is also gratefully acknowledged.

Table 3 Calculation of manual graphic operations to implement a PLA.

For the average PLA macro (Fig. 8), there were:

16 inputs
12 outputs
28 product terms
238 crosspoint devices

Number of shapes

AND array input columns = 2 × number of logical columns = 2 × 16 =	32
AND array diffusion rails = 3 shapes for every 2 product terms = 28 × 3/2 =	42
OR array diffusion rails = 3 shapes for every 2 output columns = 12 × 3/2 =	18
	92

Number of cell transforms

Crosspoint device gates =	238
AND load and bus cells (1 for every 2 product terms) = 1/2 × 28 =	14
OR load and bus cells (1 for every 2 output columns) = 1/2 × 12 =	6
AND-OR interface cells (1 for every 2 product terms) = 1/2 × 28 =	14
	272

Number of labels

AND input columns =	32
AND product terms =	28
OR output columns =	12
	72

Total operations = 436

References

1. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, June 1977, pp. 462–468.
2. P. S. Bottorff, R. E. France, N. H. Garges, and E. J. Orosz, "Test Generation for Large Logic Networks," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, June 1977, pp. 479–485.
3. *Advanced Statistical Analysis Program (ASTAP)*, Program Reference Manual, Program Number 5796-PBH, IBM Data Processing Division, White Plains, NY 10604.
4. P. Carmody, A. Barone, J. Morrell, and C. Lovejoy, "An Interactive Graphics System for Large Scale Integration Design," *Proceedings of the International Conference on Interactive Techniques in Computer Aided Design*, Bologna, Italy, September 1978, pp. 281–294.
5. K. A. Chen, M. Feuer, K. H. Khokhani, N. Nan, and S. Schmidt, "The Chip Layout Problem: An Automatic Wiring Procedure," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, June 1977, pp. 298–302.
6. C. R. McCaw, "Unified Shapes Checker—A Checking Tool for LSI," *Proceedings of the 16th Design Automation Conference*, San Diego, CA, June 1979, pp. 81–87.
7. K. H. Khokhani and A. M. Patel, "The Chip Layout Problem: A Placement Procedure for LSI," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, June 1977, pp. 291–297.
8. J. C. Logue, N. F. Brickman, F. Howley, J. W. Jones, and W. W. Wu, "Hardware Implementation of a Small System in Programmable Logic Arrays," *IBM J. Res. Develop.* **19,** 110 (1975).
9. H. Fleisher and L. I. Maissel, "An Introduction to Array Logic," *IBM J. Res. Develop.* **19,** 98 (1975).

Received April 16, 1979; revised June 20, 1979

The authors are located at the IBM System Communications Division laboratory, Kingston, New York 12401.