Copyright © 1988, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

MODULE GENERATION SYSTEMS FOR PROGRAMMABLE LOGIC ARRAYS AND DATA PATHS

by

Shau-Lim Chow

Memorandum No. UCB/ERL M88/9

15 January 1988

MODULE GENERATION SYSTEMS FOR PROGRAMMABLE LOGIC ARRAYS AND DATA PATHS

by

Shau-Lim Chow

Memorandum No. UCB/ERL M88/9

15 January 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

MODULE GENERATION SYSTEMS FOR PROGRAMMABLE LOGIC ARRAYS AND DATA PATHS

by

Shau-Lim Chow

Memorandum No. UCB/ERL M88/9

15 January 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

YN 32/9

Module Generation Systems for

Programmable Logic Arrays and Data Paths

Shau-Lim Chow

Electronics Research Laboratory
Electrical Engineering and Computer Sciences Department
University of California
Berkeley, California 94720

January 14, 1987

Module Generation Systems for

Programmable Logic Arrays and Data Paths

Shau-Lim Chow

This report has been submitted to the Department of Electrical Engineering and Computer Science. University of California. Berkeley. in partial satisfaction of the requirements for the degree of Master of Sciences. Plan II.

Approval for the Report and Comprehensive Examination:

Committee: Richard Newton, Research Advisor

Jan 14, 1988, Date

Carlo H. Séquin, Second Reader

Jan 14. 88, Date

ABSTRACT

Module generators for structured arrays create automatically area-efficient custom circuit modules while maintaining compatibility with hand designs. Two module generation systems targeted toward building the two basic components of a microprocessor chip, namely a PLA-based control unit and a data path, have been developed. The algorithms and the optimization techniques used in these two module generation systems are described in this report.

ACKNOWLEDGEMENTS

I would like to thank my research advisor, Prof. Richard Newton, for his guidance, inspiration, patience, and support for my projects in both my undergraduate and graduate years. I would also like to acknowledge Prof. Carlo Sequin for generously spending his time on the evaluation of the report and the discussion with me.

This work was supported in part by the Defense Advanced Research Projects Agency under contract N00039-87-C-0182, the Digital Equipment Corporation, and the Raytheon. Their support is gratefully acknowledged.

There are a number of colleagues who contributed a lot to my research. I would like to express my gratitude to Daebum Lee for giving me invaluable advice. Chuck Kring offered a lot of help throughout all my projects. The support of CAD tools from Jeff Burns, Mark Beardslee, Andrea Casotto, Srinivas Devadas, Mitch Igusa, Bill Lin, and Russ Segal was also indispensable. The time Srinivas Devadas and Daebum Lee spent on proofreading this report is greatly acknowledged. Wayne Christopher, David Harrison, Tony Ma, Rick Rudell, Albert Wang, and other CAD group members taught me a lot about the CAD tools, the Berkeley CAD environment, and xtrek. My office-mates, Seung-Ho Hwang, Bill Lin, and Gregg Whitcomb, maintained a very nice working environment. My buddies, Ben Liu and Jacky Mak, made my graduate life at Berkeley more interesting.

I feel indebted to my family, including my late grandfather, and Alice. Without the financial support from my parents, I would not be studying at Berkeley. Constant supportive encouragement from all of them will never be forgotten.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

CHAPTER 2: OCTPLA: A FOLDED PLA DESIGN SYSTEM 3

2.1 Previous work 3

2.2 Features of OCTPLA 4

2.3 Structure of PLAs 5

2.4 Interaction of CAD tools in OCTPLA 13

2.4.1 Folding of PLA 14

2.4.2 Mapping of cells 15

2.4.3 Placement and Connection 16

2.4.4 Compaction 17

CHAPTER 3: DAPAGES: A DATA PATH GENERATION SYSTEM 19

3.1 Previous work 19

3.2 Overview of DAPAGES 21

3.3 Netlist and Logic Description 24

3.3.1 Netlist Requirement 25

3.3.2 Logic Optimization 26

3.4 Algorithm of DAPAGES 26

3.4.1 Graph Construction 29

3.4.2 Pitch Estimation 30

3.4.3 Cell Generation 31

3.4.3.1 Combinational Logic Gate Generation 32

3.4.3.2 Stretchable Cell Generation 34

3.4.4 Resizing 36

3.4.5 Global Connection 36

3.4.6 Modification for SOG style 37

CHAPTER 4: APPLICATION AND RESULTS 41

4.1 OCTPLA 41

4.2 DAPAGES 47

4.2.1 Instruction Unit Data Path 47

4.2.2 Lower Data Path 52

4.2.3 Sea-of-Gates Data Paths 54

CHAPTER 5: FUTURE RESEARCH 57

5.1 OCTPLA 57

5.2 DAPAGES 58

CHAPTER 6: CONCLUSION 59

REFERENCES 60

CHAPTER 1

INTRODUCTION

Recent development in Very Large Scale Integration (VLSI) design has made possible the realization of a microprocessor on a single chip. The complexity of a microprocessor chip can vary from 100,000 transistors to over 300,000 transistors. To design microprocessors of this complexity is a great challenge to chip designers and Computer Aided Design (CAD) tools are vital during the design process to ease the task of chip designers. The design of a canonical microprocessor can be divided into three major parts: *data path design*, *control unit design*, and *memory structure design*. A data path is a unit that performs logical operations on a set of data with a set of control signals from the control unit. A control unit is the "brain" of a microprocessor that controls the flow of the data. Memory structures, such as register files, store data in the chip. All of them can be implemented by structured arrays. The structural regularity can be exploited by CAD tools in order to obtain optimized layout and performance. In this report two module generation systems that implement data paths and help building Programmable Logic Array (PLA)-based control units are described.

The design of these systems was first motivated by projects aimed at implementing a Central Processing Unit (CPU) chip of a multiprocessor workstation, SPUR [hill85], being developed at UC Berkeley. This workstation is designed for supporting multiprocessing in LISP programming environments. In this report a description of the use of the module generation systems for the automatic layout of various parts of the SPUR CPU chip is presented.

In Chapter 2, a folded PLA design system, *OCTPLA*, is described. Previous PLA design systems are first analyzed, followed by a brief description of *OCTPLA*. Comparisons are drawn against other PLA design systems. The interaction of CAD tools in *OCTPLA* is then explained.

A data path generation system, *DAPAGES*, is presented in Chapter 3. Similar data path generation systems developed in the past are first analyzed. A description of how *DAPAGES* remedies the common problems occurring in data path generators and its overall optimization strategy is presented. Various phases of layout generation in *DAPAGES* are then examined.

In Chapter 4 the results obtained thus far by both module generation systems are presented. An evaluation of the two systems according to the results is provided. First, some general examples produced by *OCTPLA* are shown. The application of *OCTPLA* to the design of the Instruction Unit of the SPUR CPU chip is then described, and comparisons with hand design are drawn. Next, different components of the Instruction Unit data path and the Arithmetic Logic Unit of the SPUR CPU chip generated by *DAPAGES* are presented individually. This helps understand the local optimization achieved by *DAPAGES*.

Directions for future research in both module generation systems is discussed in Chapter 5. A discussion of how the systems can be improved with reference to the evaluation obtained in Chapter 4 follows.

This report is concluded in Chapter 6. Appendix A contains the manual page of *octopus*, a key element in *OCTPLA*. Appendix B contains the manual page of *OCTPLA*. The layout design rules used in both generators are listed in Appendix C.

CHAPTER 2

OCTPLA: A FOLDED PLA DESIGN SYSTEM

PLAs are very regular structures that are widely used in Very Large Scale Integration (VLSI) circuit design to implement control logic, decoder logic and random logic. They can have a very large regularity factor [lattin79], depending on their sizes. Besides, circuit designers can get fast and early estimation of area requirement, power consumption and the critical path delay in a PLA by just looking at an intermediate description of the PLA structure, e.g., a personality matrix. This makes it very attractive for CAD designers to develop PLA design systems. However, in order to obtain designs comparable to manually generated designs, optimization steps are crucially necessary. Optimizations in a PLA design system can be classified in three areas: logic optimization, topological optimization, and electrical optimization.

2.1. Previous work

Past PLA design systems developed in industry and universities have focused on different optimization areas, how easy the systems can be maintained, and the adaptability to other layout styles. The PLEASURE/PANDA system [micheli83] [mah84] developed at Berkeley has successfully generated multiply-folded PLAs. However, like other traditional PLA generators [mayo84], it uses a tiling approach to place the PLA cell templates, resulting in a very large cell library due to the large number of templates. Effort in maintaining a cell library of over 100 cell templates [mah84] is not easy. PLASCO [bart85], using procedural design for cell generation, attempts to avoid the time consuming re-design process for cells by

instantiating a cell library from a small set of parametrized cells. A classical approach [glas80] uses a prototype PLA and adapts it through a set of rules to other layout styles. Different and efficient PLA designs can be implemented, but with the specification of a large number of rules. Other PLA design systems focus on the performance of PLAs, rather than the design system. PLAOPT [hed85] improves the speed and power consumption by modifying transistor sizes, but it expends a large amount of CPU time. The Complementary PLA technique [powell84] also aims at low power consumption in PLAs, but the resulting PLAs occupy more area than standard PLAs due to the use of full complementary logic.

The symbolic design environment of *OCTPLA* not only eases the task of maintaining the cell library in the case of changes in design rules, but it also allows different layout styles to be easily implemented. *OCTPLA* also targets at improving the performance of the PLAs by allowing different sizes of pull-up or pull-down devices and different sizes of buffers. Buffers with different embedded logic can be used simultaneously in a single PLA. This can help optimize the performance of the whole chip, as well as the PLAs.

2.2. Features of OCTPLA

OCTPLA is a pipeline of CAD programs that begins from an unfolded PLA description and automatically generates PLA layouts using a symbolic design method. It can handle different layout styles, like static CMOS and domino CMOS. This flexibility is important for a general synthesis and layout generation system, since the choice of the layout style of PLA depends on the timing constraints, as well as power and area limitations of the chip.

OCTPLA allows the use of different types of input and output buffers. This is useful when some inputs or outputs have to be clocked, while the others are

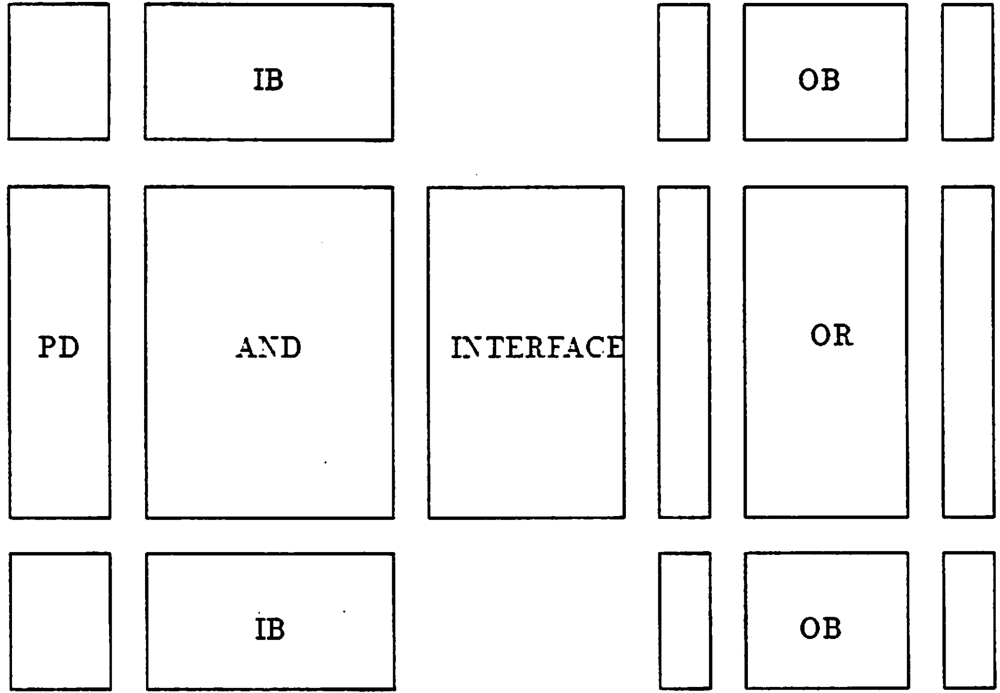
not. It is also useful when different logics are embedded in different output buffers to save area required for implementing logic outside the chip.

This system can handle singly-folded PLAs and unfolded PLAs. There are two common instances when users want to use unfolded PLAs. When the combinational logic of the PLA does not allow much folding and the PLA is not large, then the folded PLA may occupy a larger area than the unfolded one due to large buffers present on two sides rather than only one. In another case, users may want to minimize routing among different macro blocks in the chip due to changes in the logic of the PLA. A folded PLA usually causes more routing changes since the the locations of inputs and outputs may move to another side of the PLA due to folding.

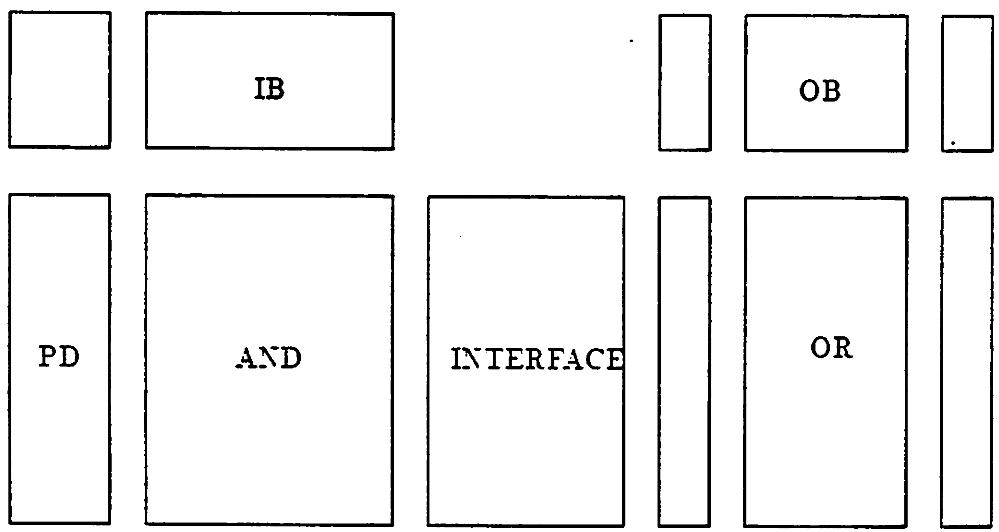
One other major feature of this system is the use of a hierarchical symbolic design method. This leads to much smaller mask layout description than one using only a single level physical design. The symbolic design environment at Berkeley also helps reduce the number of templates used for a PLA, as is explained in Section 2.4.2. The minimum number of templates used in *OCTPLA* is twelve.

2.3. Structure of PLAs

To facilitate discussion, the structures for a typical folded PLA and a typical unfolded PLA are shown in Fig. 2-1. The unnamed blocks are connectors, which serve the function of connecting other blocks. A PLA is composed of **AND** and **OR** planes, input and output buffers, and peripheral devices. The **AND** plane contains transistors that are connected to realize the product terms of the PLA, and the transistors in the **OR** plane are used to realize the sum terms of the PLA.



(a). a typical folded PLA



(b). a typical unfolded PLA

Fig. 2-1 Block diagrams of typical PLAs

Different layout styles require the use of different cell libraries and different connection rules. (Connection rules will be explained in Section 2.4.3). In a NOR-NOR pseudo NMOS layout style, both planes contain NMOS transistors in parallel, while the **PD** block is implemented by weak PMOS transistors. The **INTERFACE** block is then simply a column of connectors. The **OB** cells consist of output buffers and weak PMOS pull-up transistors used in the **OR** plane. The circuit level schematic of this type of PLA is shown in Fig. 2-2.

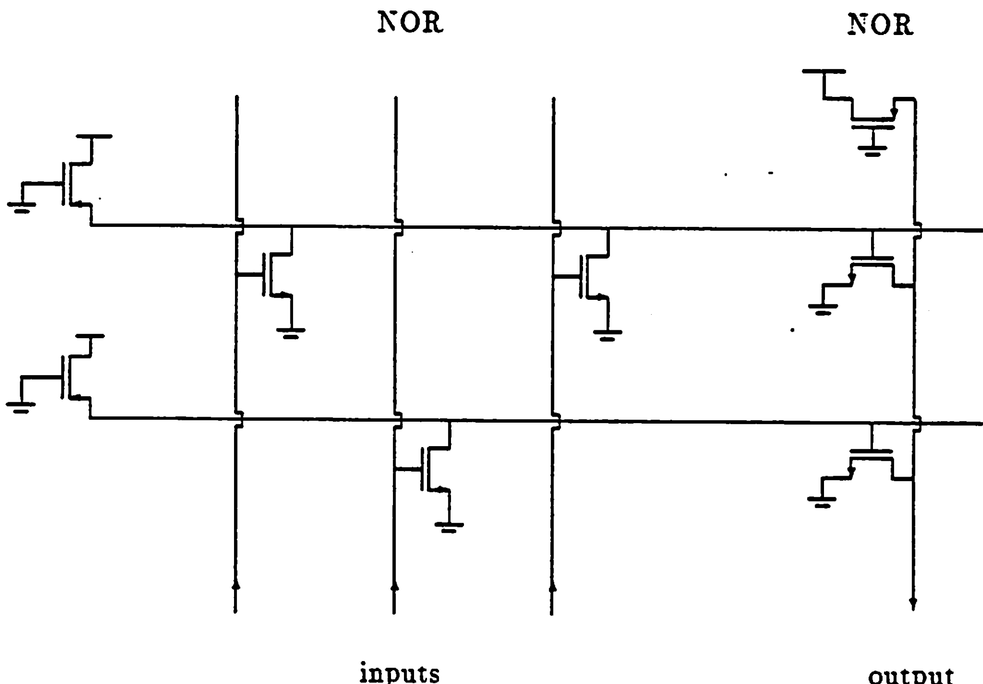


Fig. 2-2 Circuit schematic of a NOR-NOR pseudo NMOS PLA

It turns out that for a minimum area layout, diffusion is used as the vertical and horizontal ground line in both the **AND** and **OR** planes respectively. So horizontal and vertical metal ground lines are occasionally introduced in the **AND** and **OR** planes respectively to bring the diffusion ground line resistance down. The number of metal ground lines depends on the size of the PLA and the resistance of diffusion, which in turn depends on the process technology used.

In a Domino CMOS layout style, the circuit level schematic is shown in Fig. 2-3. In this case, the **PD** block is implemented by clocked NMOS pull-down transistors, while the **INTERFACE** block consists of clocked PMOS pull-up transistors used in the **AND** plane and inverters between the two planes. The **OB** is modified to contain output buffers, and the pull-up and pull-down transistors used in the **OR** plane.

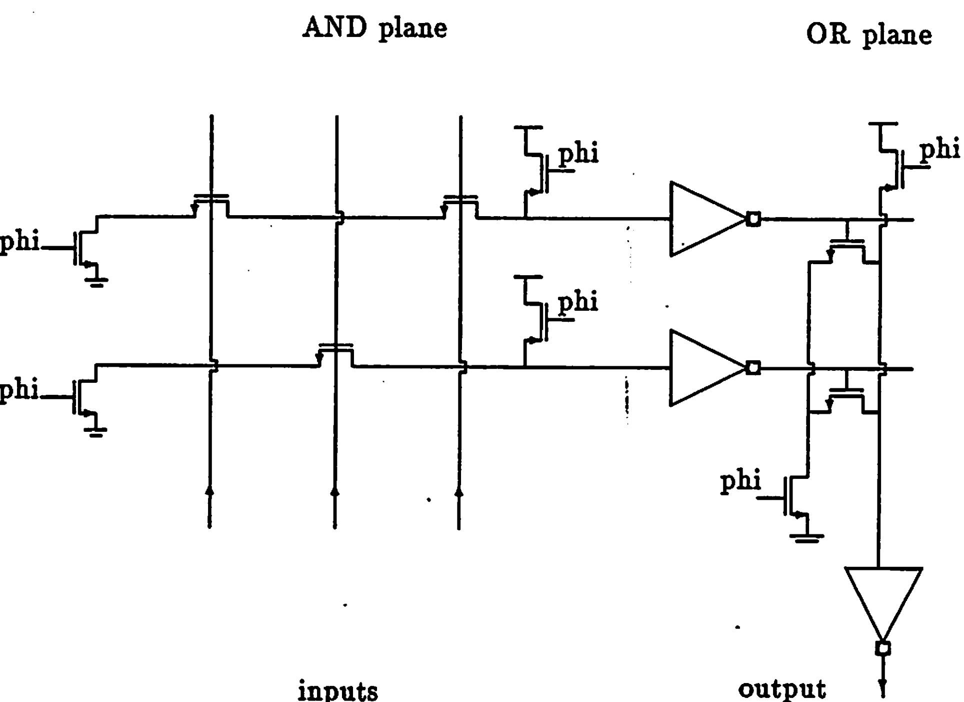


Fig. 2-3 Circuit schematic of a domino PLA

Different features can be included in a PLA by putting different components in the blocks in Fig. 2-1. Varying the size of pull-up or pull-down devices can optimize PLA performance. Sense amplifiers can be placed in the **INTERFACE** block to enhance the rise or fall time of the product term signal entering the **OR** plane. This is particularly useful for large PLAs, since the RC delay is large and a slight swing of voltage is then enough to operate the **OR** plane.

Different types of buffers have been designed for different purposes, again without any change in the algorithm used. The circuit level schematics and layouts[1] of these buffers are shown in Fig. 2-4 and Fig. 2-5 respectively. Buffers can be clocked and logic can be included in buffers. In Fig 2-4e the output voltage of the output buffer remains at logic '0' until the onset of the evaluation phase. Fancier logic can be included at the cost of slightly larger buffers.

---

1. Different masked layout layers and their corresponding layout patterns are shown:

N-diffusion

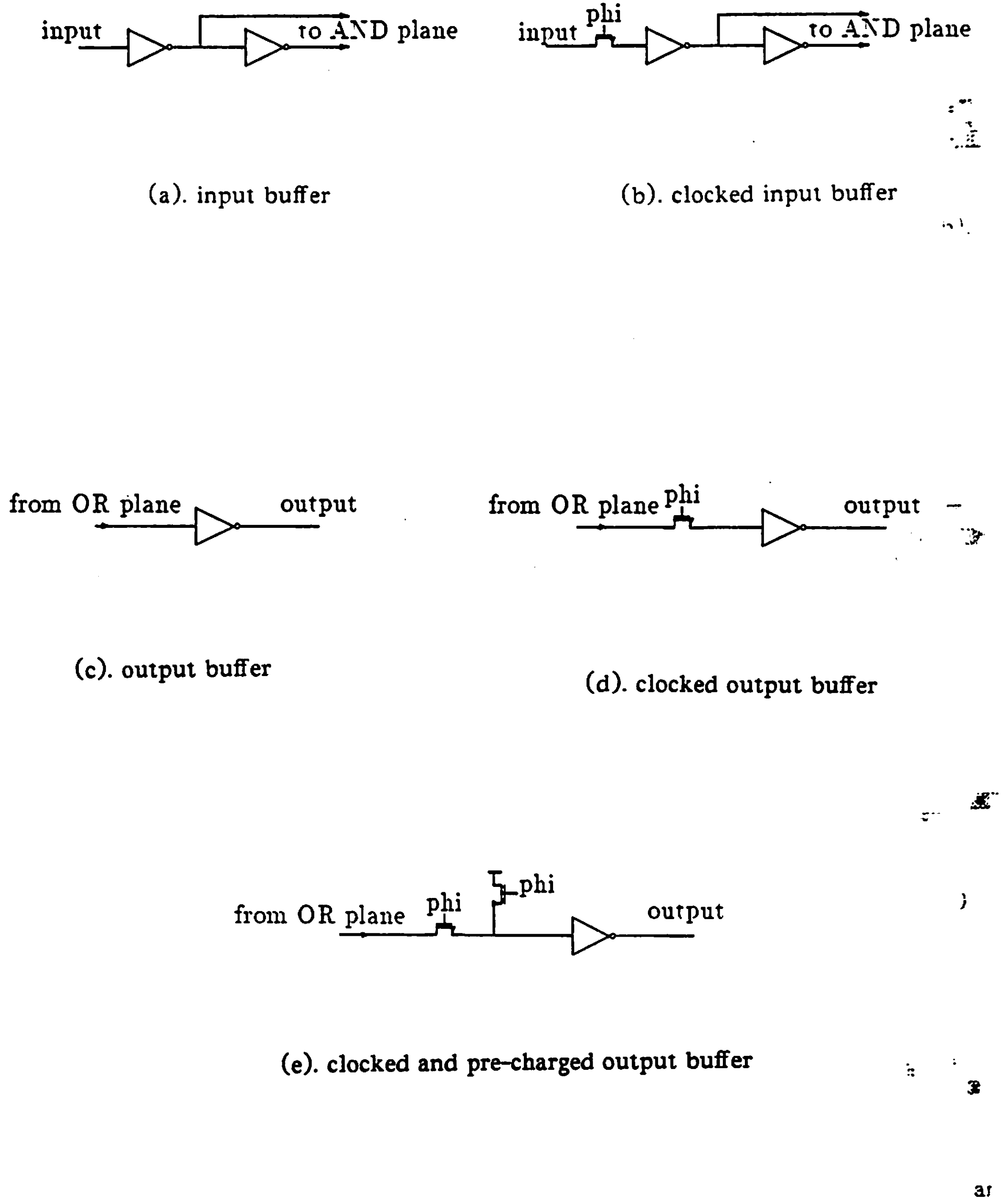
P-diffusion

polysilicon

first metal

second metal

contact



(a). input buffer

(b). clocked input buffer

(c). output buffer

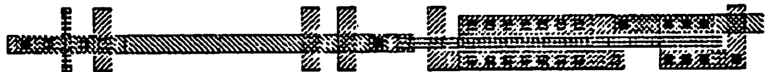
(d). clocked output buffer

(e). clocked and pre-charged output buffer

Fig. 2-4 Circuit schematics of buffers

(a). input buffer

(b). clocked input buffer



(c). output buffer

(d). clocked output buffer

(e). clocked and pre-charged output buffer

Fig. 2-5 Layouts of buffers:

2.4. Interaction of CAD tools in OCTPLA

Within the Berkeley synthesis framework, a number of tools exist that are integrated to form the *OCTPLA* system. These tools perform folding, mapping, placement and connection, and compaction. *Genie* [dev86] is a general array folder that produces a folded PLA personality matrix. *Octopus* is the key element of this system. It maps characters of a PLA personality matrix to transistors. It decides the appropriate peripheral devices and buffers based on the user's specifications. *MkArray* [kring87] forms an underlying framework for array composition. *Sparcs* [burns86] is a symbolic compactor that ensures minimum area and corrects design rule error. Fig. 2-6 illustrates the interaction of the CAD tools.

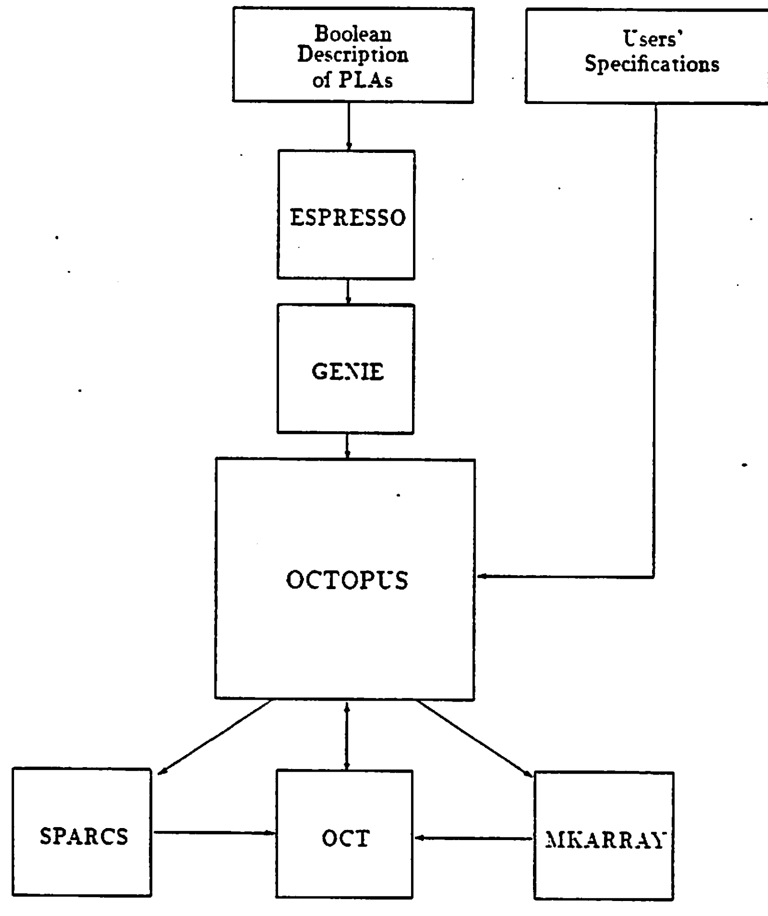


Fig. 2-6 Interaction of CAD tools in *OCTPLA*

2.4.1. Folding of PLA

The input to the folding program, *Genie*, is the output of a logic minimization tool *espresso* [rudell85]. *Genie* uses simulated annealing and performs single folding in *OCTPLA*. Input signals and their complements are constrained to stay together and on the same side of the PLA. A sample input and output file is shown in Fig. 2-7.

```
# espresso
.i 6
.o 2
.ilb 1 2 3 4 5 6
.ob 7 8
.p 5
-1---- 10
1---0- 01
1--0-- 01
1-0--- 01
-----1 01
.e
```

(a). input file

```
new 10 9

.c....c..
sc.ss.cs.
...s....c
........c
....s.c..
......c..
..c....s.
..c..cs..
.....c...
p..pp.pp.

( row 0 2* 0 1* 6 )
( row 1 2 0 1 3 )
( row 2 3* 0 )
( row 3 3 0 )
( row 4 4* 0 )
( row 5 4 0 )
( row 6 5* 0 )
( row 7 5 0 6 5 )
( row 8 6* 0 )
( row 9 7 0 8 3 )
( column 0 0 )
( column 1 0 )
( column 2 0 )
( column 3 0 )
( column 4 0 )
( column 5 0 )
( column 6 0 4 7 )
( column 7 0 )
( column 8 0 )
```

(b). output file

Fig. 2-7 Sample input and output files of *genie*

2.4.2. Mapping of Cells

Octopus reads the personality matrix generated by *Genie*, and then define the size of the array that holds cell instances. Using the pre-defined template structure of a PLA, *octopus* divides the array into a number of rows and columns. One cell instance may be put in one or more unit areas in the array. The placement of the instance depends on the topology of the PLA. If overlapping of instances is desired, then they share one or more unit areas in the array. An example is shown in Fig. 2-8. In this example, Transistors 1 and 2 have their diffusion layers overlapped, since they share the same ground line. They cannot be put in only one unit area; otherwise there is no way to differentiate the two different transistors. The input buffer shown supplies the signal lines connecting to Transistors 1 and 3. Thus the buffer should share the columns occupied by Transistors 1 and 3.

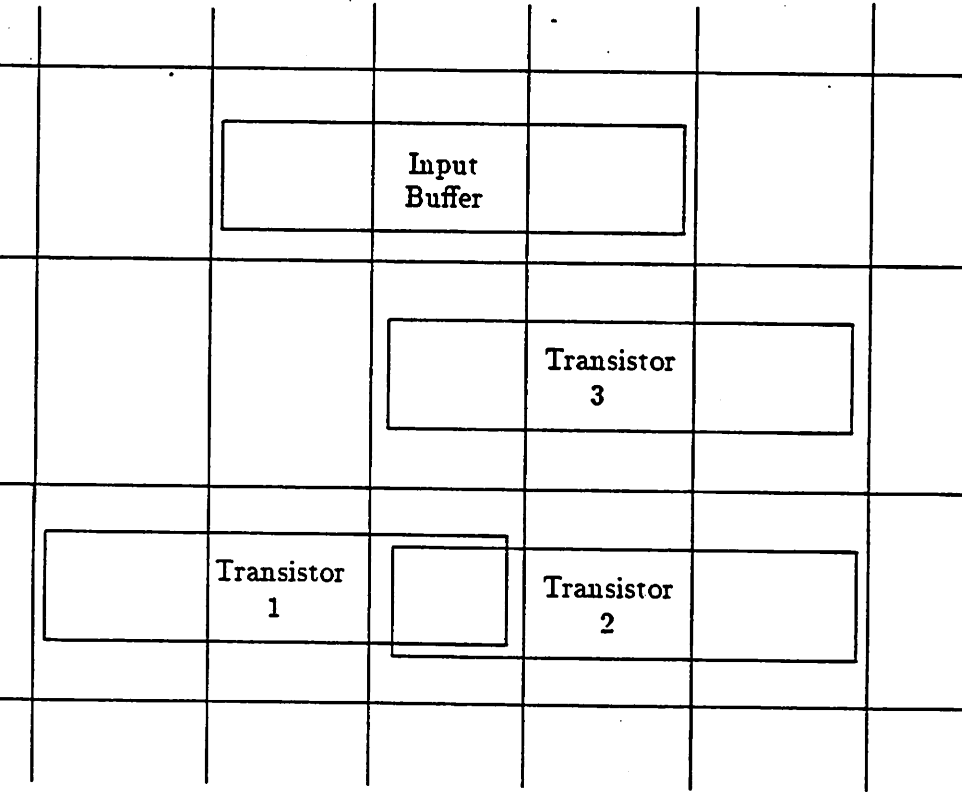


Fig. 2-8 Placement of cell instances in one part of an array

If a pure tiling technique is used in the above example, then the transistor cell and input buffer cell will have to be chopped off into several smaller cells. This is one of the reasons why *OCTPLA* can have a cell library of only twelve cells. Another reason is that the database of the layout design tools, *OCT* [harr86], allows all eight Manhattan transformations of a cell, including mirroring.

If pull-up or pull-down devices with variable sizes are needed, *octopus* counts the number of product terms in the **AND** and **OR** plane. It then decides whether a larger or normal size pull-up is to be placed in that row or column.

2.4.3. Placement and Connection

After the cells have been put in the array, *MkArray* places the cell instances according to their sizes and the constraints of the spaces between instances. A placed PLA without connection is shown in Fig. 2-9a. It then connects the instances in a certain area, specified by *octopus*, according to the connection rules specified in a rule cell. The rule cell contains a number of **RULE** bags which consist of different rules. To implement the NOR-NOR PLA design, thirteen **RULE** bags are used. An example of a rule cell is shown below:

```
(FACET PLArule:symbolic
   (BAG RULE
      (PROP RULENAME "pull_up")
      (BAG CONNECT
         (PROP WIRE_MAXIMUM "")
         (TERM Vdd_term1_of_inst1)
         (TERM Vdd_term1_of_inst2)
         (TERM Vdd_term2_of_inst3) )
      (BAG CONNECT
         (TERM poly_term1_of_inst1)
         (TERM poly_term3_of_inst2)
         (TERM poly_term1_of_inst3) )
   )
)
```

RULENAME is used to specify the name of the rule. Terminals inside a **CONNECT** bag are connected together. The *OCT* property **WIRE_MAXIMUM** informs *MkArray* that the wires that connect the terminals should be as wide as possible. The example PLA is connected and shown in Fig. 2-9b.

2.4.4. Compaction

After *MkArray* places and connects the cell instances, the resulting layout may not be very compact, and the placement does not guarantee the absence of design rule error. So a symbolic compactor is called to ensure correct and minimum design rule spacing. The example PLA is compacted as shown in Fig. 2-9c.

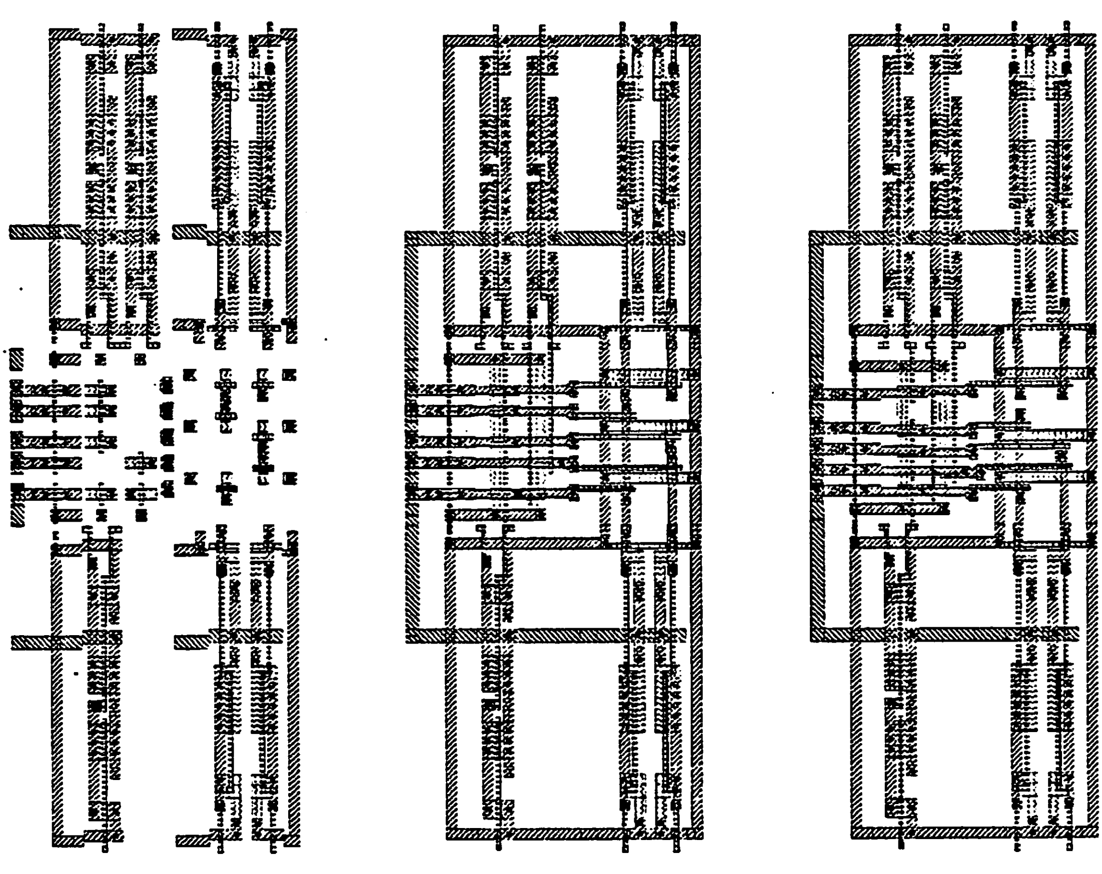


Fig. 2-9a. An unwired PLA　　Fig. 2-9b. a wired PLA　　Fig. 2-9c. a compacted PLA

CHAPTER 3

DAPAGES: A DATA PATH GENERATION SYSTEM

As more module generators are developed for regular structured modules, like PLAs and ROMs, techniques for other structured modules that are not as regular are being investigated. Data path is one type of modules that has been targeted. Data path is an indispensable structure in microprocessor chips and digital signal processing chips. Many other chips have their own data paths to suit their special needs. A data path is regular in the sense that each bit-slice is either similar or identical to another bit-slice of the data path. A data path, however, is more complex than PLA, ROM, and RAM. Therefore, although traditional module generator techniques, like tiling, can still be applied to data path generation, more advanced techniques are required to generate high performance data paths.

In the following sections, some previous works are discussed, followed by an overview of a new data path generation system, *DAPAGES*. The input format is explained first. The algorithm of *DAPAGES* is presented systematically in the last section. The results obtained by *DAPAGES* are presented in Chapter 4.

3.1. Previous work

In contrast to PLAs, data paths are less regular and generators for data paths are less common. Different approaches have been taken in past data path generators. Some of them rely more on experienced circuit and layout designers to design compact individual cells, while some others put more emphasis on the constraints imposed on the cells. In general, data path generators can be classified into

three categories by the cells they use to generate the entire data path: fixed cell library generators, modifiable cell library generators, and automatically generated cell library generators.

The first type of generator takes the simplest approach. The whole data path is partitioned into bit-slices, and each bit-slice is further decomposed into different elementary cells. Each elementary cell is laid out by hand to achieve optimal performance, while its area is minimized. These cells are then tiled to form the whole data path. Routing is often needed for the interconnection of cells, thus enlarging the entire data path. This approach suffers from the fact that changes in design rules or the data path architecture require a re-design of the cell library. However, this approach is simple and fast, so some new generators [ruetz86] [rowson87] are still using it.

The second type of generators use either parameterized cells or virtual grid symbolic layout methods. Both approaches can adapt to different constraints, thus a change of the data path architecture imposes less penalty to the final layout than the layout of the first type of generators. Generators [marsh86] that use the former approach have a cell library ready to be modified by input parameters. So the cells have some flexibility to re-configure. This flexibility, however, may not assure automatic alignment of terminals when there is a change in the architecture. Generators using the second approach [weste87, tan87] take advantage of virtual grid symbolic design to achieve pitch matching of cells and terminals, which is area efficient. Compactors are used to handle the pitch matching and to ensure design rule correctness. The performance, however, depends on how well the original layouts are constructed.

The third type of generators use a procedural design method [shrobe82, batali81]. Cells are defined procedurally and a set of parameters is passed to the

procedure to allow cells taking on different configurations. This approach does not rely on any existing layout and can adapt to any local constraints on pitch matching. It is, however, time consuming since all the cells have to be generated for each data path. Other procedural languages [lipton82] and procedural design environments [wood86] also exist to support this type of generator, and the constraints imposed by the use of a virtual grid.

The above review explains what advantages and disadvantages each type of data path generator has. The first type, although it is simple and fast, is not good enough for advanced data path generators. Thus *DAPAGES* uses a mixed approach and takes advantages of the other two types of generators. The modifiable cells that *DAPAGES* uses are memory and tri-state cells. These are termed stretchable cells. To generate this kind of cells automatically while keeping them compact is difficult. So they are partially laid out and can be stretched to pitch match with the neighboring cells. The other cells are combinational logic cells which are decomposed into logic gates. Each logic gate is generated automatically using a procedural approach. Parameters, including the locations of inputs and outputs, are passed in the generator. In this way, the advantages of both types of generators are exploited, while keeping *DAPAGES* faster than the pure Type 3 approach. Moreover, rather than only extracting local constraints, *DAPAGES* first extracts global constraints by examining all the cells in the whole data path. This gives *DAPAGES* better performance than the data path generators developed in the past.

3.2. Overview of DAPAGES

DAPAGES is a data path generation system for VLSI design. It takes a hierarchical netlist of logic blocks and their logic description as input, and generates a symbolic layout of the data path without a cell library. The whole pro-

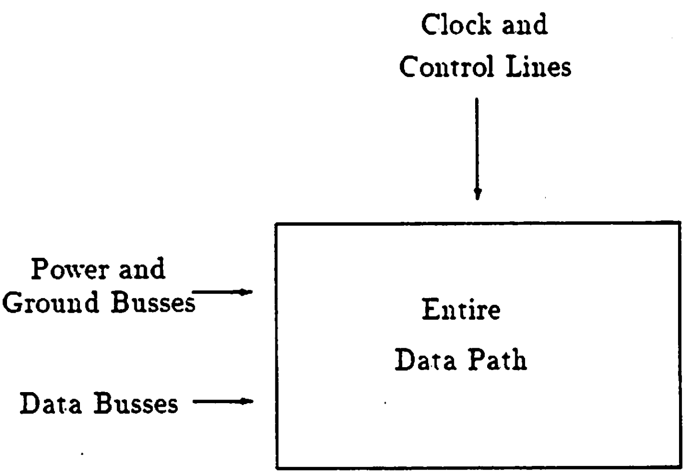
cess first produces a two-dimensional graph of cells. The next process uses *message-passing* to ensure pitch-matching of cells and input/output ports, thus greatly reducing the routing area. Each gate within a cell is individually configured to avoid excessive area consumption and unnecessary delay due to routing among gates.

The cells in the graph are treated as individual objects. They pass messages to neighboring cells to signal the beginning or the end of a process. These messages also contain the constraints of the terminals and the pitch of cells. The message-passing algorithm will further be explained in Section 3.4.3.

In the process of layout generation, technology and layout-style dependence is minimized so that different implementation technologies can be employed. Moreover, different layout styles can be implemented without a major change in the program. Presently, *DAPAGES* supports full custom and Sea-of-Gates (SOG) layout style.

Before going into the details, it is useful to know the floorplan of the entire data path. It is constructed in such a way that logic blocks are placed horizontally. Thus data busses, power and ground busses flow in the horizontal direction. Control lines, on the contrary, flow in the vertical direction. In other words, bit-slices containing registers and functional operators are stacked up to form the whole data path, starting with the least significant bit at the bottom. Each cell in a bit-slice is provided with one power bus and two ground busses, thus allowing two rows of NMOS transistors and two rows of PMOS transistors. The two ground busses bound the cell at the top and bottom. This layout structure brings the aspect ratio closer to unity. It also allows a bit-slice to share a ground bus with the upper bit-slice and another ground bus with the lower bit-slice. The structure of the data path is illustrated in Fig. 3-1.

DAPAGES is explained more thoroughly in the rest of this chapter, starting with a description of the design environment in which *DAPAGES* resides. The data-path generation algorithm is then decomposed into different parts and explained.



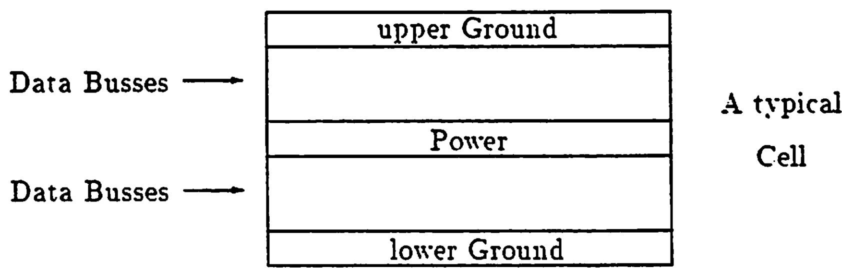


Fig. 3-1 Structure of a typical data path

3.3. Netlist and Logic Description

Users start from a behavioral description of the netlists of logic blocks and the description of the logic of a unit cell. Since data paths are generally large and complicated, the netlists are usually expressed hierarchically. The netlist description is then converted to netlists in the database, *OCT* [harr86], by a netlist translator, *bdnet* [segal87], and the logic description, written in BDS [segal87], is converted to the Berkeley Logic Interchange Format (BLIF) description by the logic description translator, *bdsyn* [segal87]. The logic is then minimized by a multi-level logic optimization program, *MIS* [rudell86]. The above process is illustrated by the diagram in Fig. 3-2.

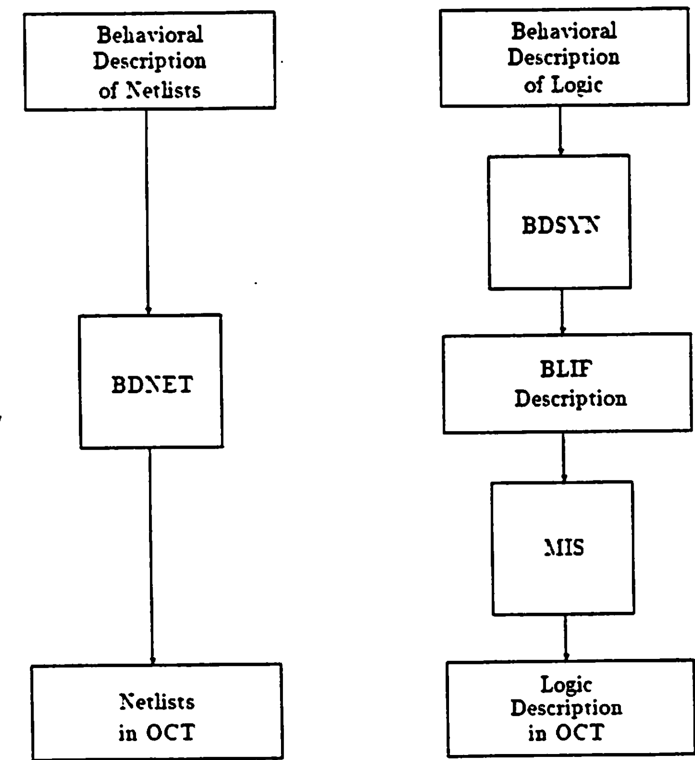


Fig. 3-2 Behavioral synthesis flow diagram

3.3.1. Netlist Requirement

Since the relative placement of logic blocks is a higher level synthesis problem, it is not performed by *DAPAGES*. Instead users provide coarse placement information of different logic blocks by specifying the relative co-ordinates of the logic blocks in the netlist description.

To differentiate different types of cells stored in the database, *OCT* allows properties of different values attached to cells. The **CELLCLASS** property is used to differentiate a single cell from a block of cells when a hierarchy of netlists is parsed. The possible values of this property are "MODULE", "UNITCELL", and "LEAF". The "LEAF" cell class includes all the basic gates that are used by the *MIS*. These gates will be discussed later. The "UNITCELL" cell class represents all cells that are the basic units in the two-dimensional graph. These basic units will be called unit cells from now on. "MODULE" cells contain one or more unit cells. Another requirement for a unit cell is the **CELLTYPE** property. This differentiates combinational logic cells from special stretchable cells. This is needed since a stretchable cell is chosen from a cell library and then modified, while a combinational logic cell is decomposed into gates and then generated automatically.

A simple example to illustrate the above properties is shown as follows:

```
(FACET datapath:symbolic
   (INSTANCE adder_A [0,0]
      (PROP CELLCLASS "MODULE"))
   (INSTANCE adder_B [0,20]
      (PROP CELLCLASS "MODULE"))
   (INSTANCE shifter [10,0]
      (PROP CELLCLASS "MODULE"))
   (INSTANCE latch.0 [30,0]
      (PROP CELLCLASS "UNITCELL")
      (PROP CELLTYPE "MEMORY"))
   (INSTANCE latch.1 [30,0]
      (PROP CELLCLASS "UNITCELL")
      (PROP CELLTYPE "MEMORY"))
)
```

In this example, the adder is made up of two different blocks, with "adder_B" placed on top of "adder_A". Each adder will further be broken down into unit cells. The shifter cells are identical and are represented by one block. The shifter is placed on the right side of the adders. Two identical latches placed on the far right side are represented by two latch instances. These two latches are stretchable library cells.

3.3.2. Logic Optimization

Gates are implemented in static CMOS and domino CMOS technology. Gates in static CMOS include NAND, NOR, and inverter. Gates in domino include AND, and OR. Two cell libraries are used to accomodate both technologies. *DAPAGES* recognizes the technology by reading the IMPL_TECH property attached to a gate cell. The two possible values of this property are "STATIC" and "DOMINO".

It should be noted that *MIS* does not handle pass transistor logic. This poses an area-efficiency problem to *DAPAGES* layout. For example, a multiplexer implemented by pass transistor logic is much smaller than if it is implemented by NAND gates and inverters.

3.4. Algorithm of DAPAGES

The whole process includes five major steps. The first step is input processing which creates a two-dimensional graph of cells from the input hierarchical netlists. Each cell corresponds to one bit of a logic block in the data path. Preliminary information, including bypassing busses, input connection, and output connection, are stored in each cell. The second step is the estimation of the vertical pitch of each cell. It determines the maximum height of each bit-slice. The third step generates individual cells and stretches specified individual cells stored in the

library. Constraints are passed among cells to make sure the input and output ports match so that no routing is required between adjacent cells. The fourth step involves resizing of each vertical slice. Each slice represents a column of unit cells. The resizing is to make sure that vertical signals can be connected without routing. Steps 2, 3, and 4 are distributed processes, performed by message passing among all cells. The last step extracts the connection information from each cell and then connects all the individual cells to form the entire data path. The process flow chart is shown in Fig. 3-3.

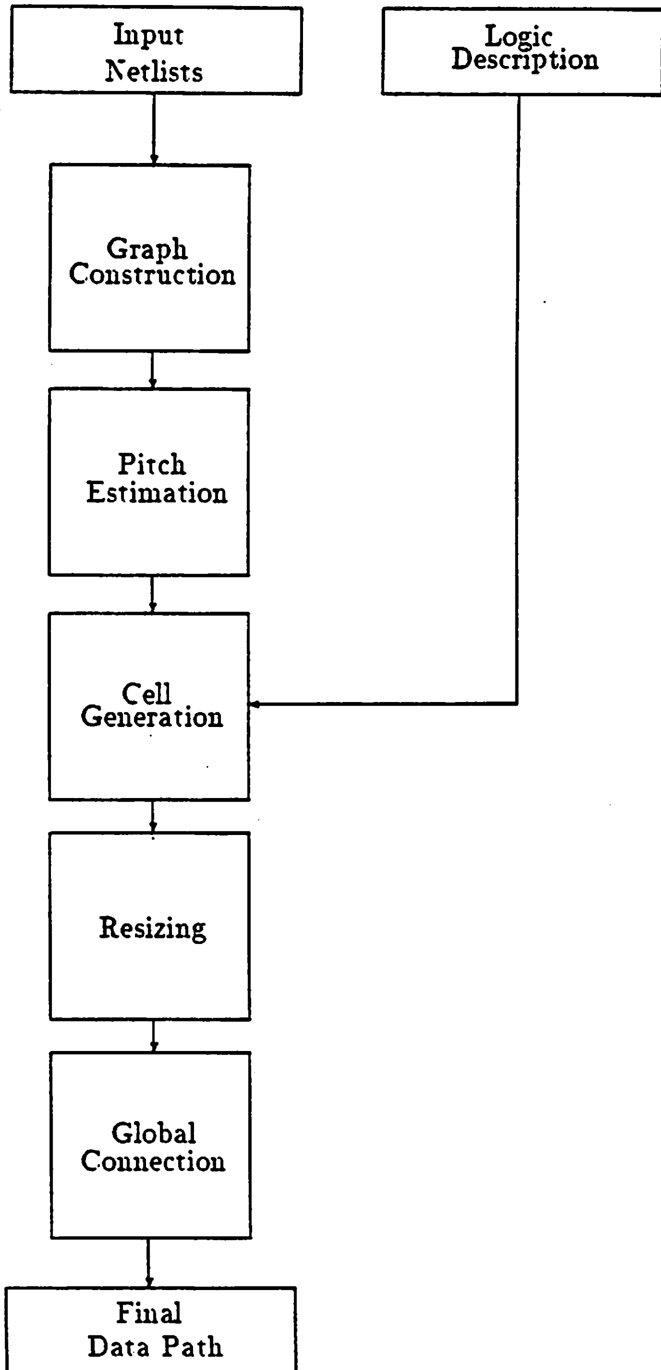


Fig. 3-3 Algorithm of *DAPAGES*

3.4.1. Graph Construction

A recursive function is called to extract all the cells in the hierarchical netlists. Cells are further classified to *base cells*, *block base cells*, and *normal cells*. A base cell is defined as a unit cell which is also the first cell instance of a new type of logic block. It is usually the least significant bit of a logic block. A block base cell is similar to a base cell, except that it has a **CELLCLASS** property "MODULE". A normal cell is any cell other than the base cell and the block base cell. Cells belonging to the same column are connected vertically according to their co-ordinates, while the base cells and the block base cells are connected in the order specified by their *oct* co-ordinates. Then each block base cell is extracted and replaced by its component cells. At this point, the whole graph contains unit cells. The links among all cells are then created, and information about bypassing busses is added to each cell.

The following is the pseudo codes of the graph building function:

```
Open input cell(&cell);
Init_octGenerator(&cell, &instance);

while ( generation not finished ) {
   if ( instance is a unit cell ) {
      allocate a unit cell for the instance;
      if ( instance is a base cell ) {
         add it to a list of base cells;
      } else {
         if ( the corresponding base cell exists )
            get the base cell from the base cell list;
         else
            create the corresponding base cell;
      }
      stack the unit cell on its base;
   } else {
      put the instance in a block list;
   }
}

while ( not all the block cells have been processed ) {
   call this function to process a block in the block list;
}
```

Arrange the bases and block bases according to the *oct* co-ordinates;
return();

It should be noted that after the whole graph has been created, each unit cell belongs to a unique row and column.

Each cell should "know" how it is connected to the neighboring cells. So each one stores the names of its formal terminals and the corresponding terminals that will be connected to its formal terminals. Cells that are placed on the edge of the datapath will then store the names of the formal terminals of the whole datapath for one or more sides.

Each cell should also "know" all the signals that pass through the cell passively. The above connection information is important in finding the path that signals go through. By looking at the relative locations of terminals in a net, the paths of bypass signals can be determined.

3.4.2. Pitch Estimation

A good estimation of pitch is important since resizing a cell to pitch match with other cells in the same row is costly. The process starts with estimating the height of each cell in a bit-slice. The maximum height of the cells defines the pitch of the bit-slice. The process is repeated for the other bit-slices. Like cell generation, this process is distributed. It is performed by message passing among the cells. It should be noted that this process is not necessary for SOG style since each cell has a pre-defined pitch. This will be explained in the next section.

Pitch estimation of a combinational logic cell is done after the logic gates inside the cell have been placed in one dimension by a placement tool, *octlatte* [lin87]. *Octlatte* uses simulated annealing to minimize the net density between adjacent gates. Then the maximum net density is determined and the pitch is thus

found. It should be noted that *octlatte* does not count the input and output nets. Therefore, when the net density is calculated, these nets have to be considered carefully since they may enter or leave the cell on any side.

Pitch estimation is not necessary for stretchable cells, since they are passive. Their pitches are defined by the other combinational logic cells in the same row. They are initially laid out with a minimum pitch, and then stretched vertically according to the maximum pitch defined above.

3.4.3. Cell Generation

Cells are generated starting from the lower left corner cell. The other cells in its bit-slice are generated with the constraints from the left. If any cell cannot meet the constraints, then they pass messages back to the leftmost cell and then re-start the process. Re-starting the process, however, is very unlikely since a good estimation has been made before the cell is generated. After one bit-slice has been generated, the leftmost cell asks its top neighbor to generate its bit-slice. The process continues until the whole data path has been generated.

When a cell is generated, constraints come from both the left neighbor and the bottom neighbor. The former set of constraints can easily be met by placing the input and output ports on the left side before any generation of gates is done. However, the constraints from the bottom are harder to meet, since the gate placement may not allow two vertical signal lines to be placed too close. This happens when the cell is different from its bottom neighbor. In this case, resizing is needed which is explained in Section 3.4.4.

3.4.3.1. Combinational Logic Gate Generation

Gates within a cell are already placed by *octlatte*. Each one of them is laid out individually with given input and output tracks. Each type of gate is handled by a dedicated procedural call with the above parameters. Since the input and output tracks are not fixed, standard cells cannot be used.

Recall that each cell is bounded by two ground rails on the top and bottom sides, while a power rail runs through the middle of each cell. So there can be at most four rows of transistors, including both NMOS and PMOS. There are four possible combinations for the placement of N-diffusion and P-diffusion blocks of a gate. The placement of transistors depend of the input and output tracks, and also the transistor placement of the last gate. For example, if the last gate is placed entirely in the top half of the cell, then the current gate should be put in the bottom half. The two gates can then share the empty spaces after compaction. Examples of two different transistor placements and different parameters, including input and output locations, for a 2-input NAND procedural call are shown in Fig. 3-4.

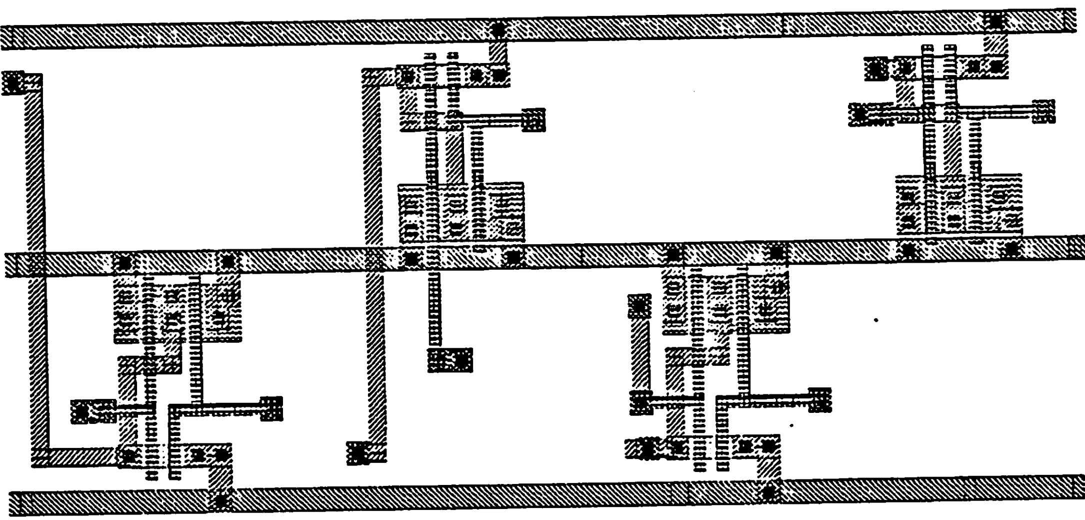


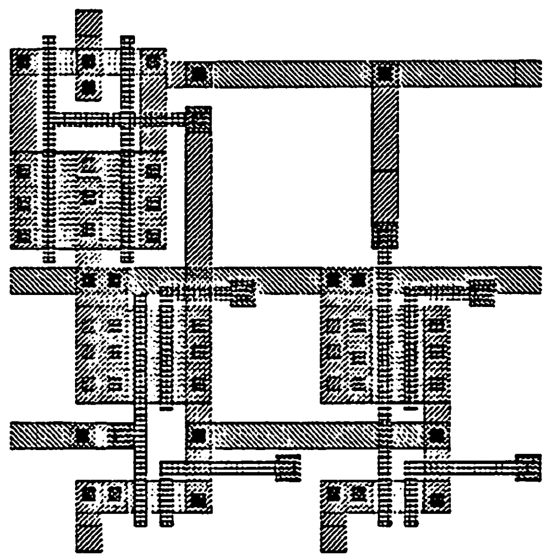
Fig. 3-4 Examples using different parameters for a 2-input NAND gate

3.4.3.2. Stretchable Cell Generation

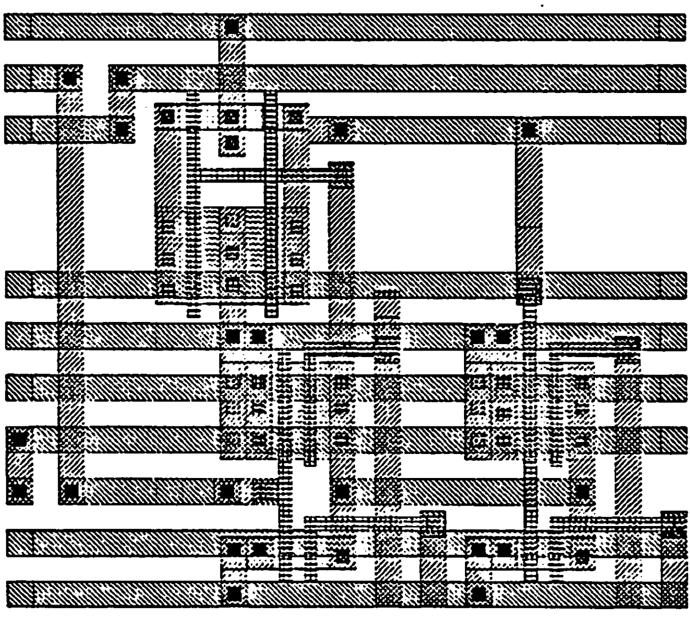
Stretchable cells are laid out with a pitch equal to the minimum pitch for all cells. The minimum pitch is defined as the number of allowable horizontal second metal tracks when all four rows of transistors are present in the cell. A latch cell is shown in Fig. 3-5a. The two ground rails are missing from the figure. They are added after the cell has been stretched.

To ease the modification process, some *OCT* bags and properties are required for stretchable cells. **TOP_TERM** and **BOTTOM_TERM** bags contain terminals of vertical signals on the top and bottom sides respectively. These terminals also include those to be connected to the ground rails. The **USED_M2_TRACKS** bag specifies the horizontal second metal track which has already been used by the cell. Contained by the bag is one or more **TRACK_NUMBER** properties which specifies the number of the used track. Another useful property, **DP_M1_TRACKS**, attached to the facet specifies the number of vertical first metal tracks occupied by the vertical signals and the vertical ground wires.

The cell is laid out with minimal use of second metal to allow more room for horizontal bypassing busses which are added during the modification process. When the cell is stretched, the above *OCT* information is first extracted. Then the ground rails are placed and connected. Routing is done to the inputs of the cell if necessary. An example for a stretched latch cell is shown in Fig. 3-5b.



(a). before modification



(b). after modification

Fig. 3-5 A latch cell before and after modification

3.4.4. Resizing

Signals running vertically, except clock and control signals, are placed on the left of the gates so that the spacing is regular and so the constraints can easily be met by the top neighbor cell. However, if all the vertical signals, including clock and control signals, are placed on the left, then the net density on the left may be too high and the fixed number of available tracks in the SOG cell template may not be enough. So clock and control signals, which run through the whole data path and are likely to be used by all the cells in a column, are placed next to the gates that use them.

If two cells in the same column have different gate netlists, then they may share two control signals but having different spacings for the signals. In this case, the most constrained cell determines the spacing of the two control signals. Iterations may be needed to make sure all the terminals can be pitch-matched.

This process also uses message passing to pass constraints to cells in a column. It is similar to the message-passing algorithm used in cell generation. However, resizing is performed for a column of cells first, and then the process propagates to the right to the other columns of cells.

3.4.5. Global Connection

After every cell has been laid out, they are connected using the connection information stored in each cell. This step is basically tiling, since all cells pitch match and all terminals align. Ground rails of cells are shared by their top and bottom neighbors.

3.4.6. Modification for SOG style

Modifying *DAPAGES* to adapt to different layout styles is not difficult, since most changes required only happen in the cell generation process. The sea-of-gates layout style has pre-defined locations for transistors and power and ground busses, thus having less flexibility than the custom layout style. The Mariner SOG cell template [layer87] also has four rows of transistors, but it is bounded by two power rails and the ground rail runs through the middle.

The major difference in the cell generation process is the use of partially laid out cell templates for logic gates. Since compaction is not appropriate for SOG style and gate placement is one dimensional, each gate always occupies a whole column of transistors within a cell. So a column of four transistors is taken as a cell template. Gates are then carefully laid out to allow input and output routing be done within each template. Transformation of the template and interchange of inputs for a multiple-input gate can also help the routing. three templates are shown in Fig. 3-6.

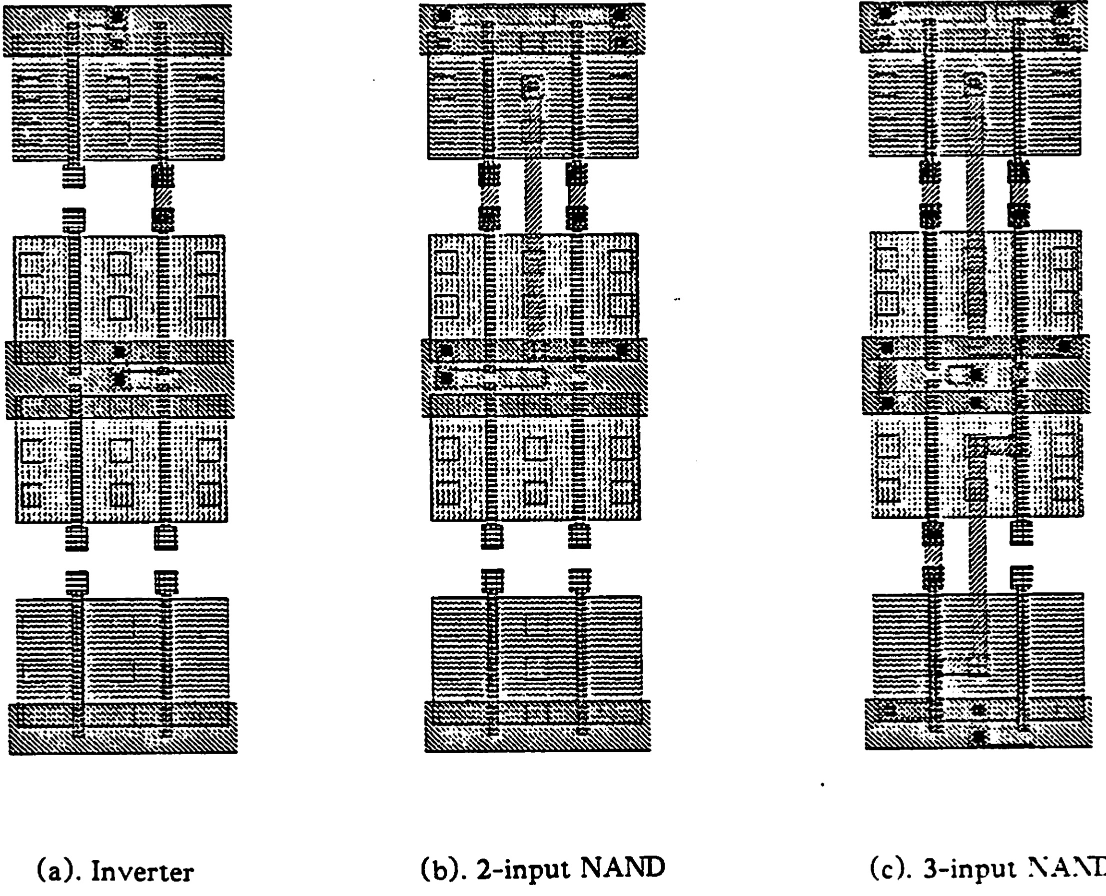
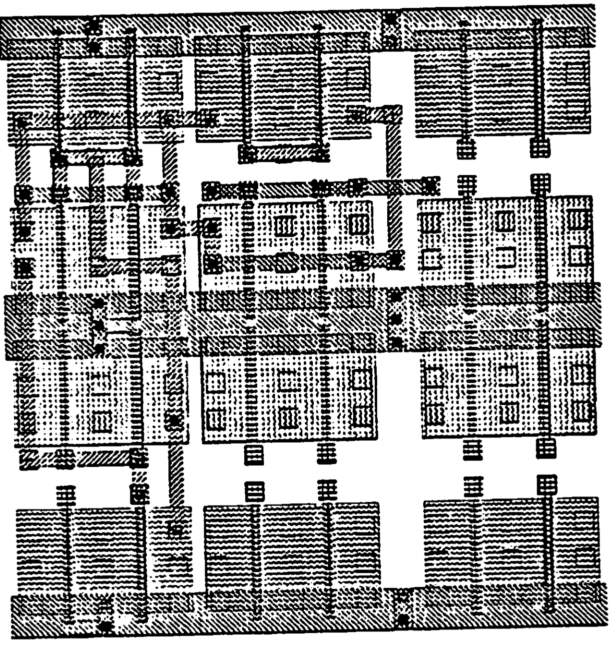


Fig. 3-6 Combinational gate templates in SOG style

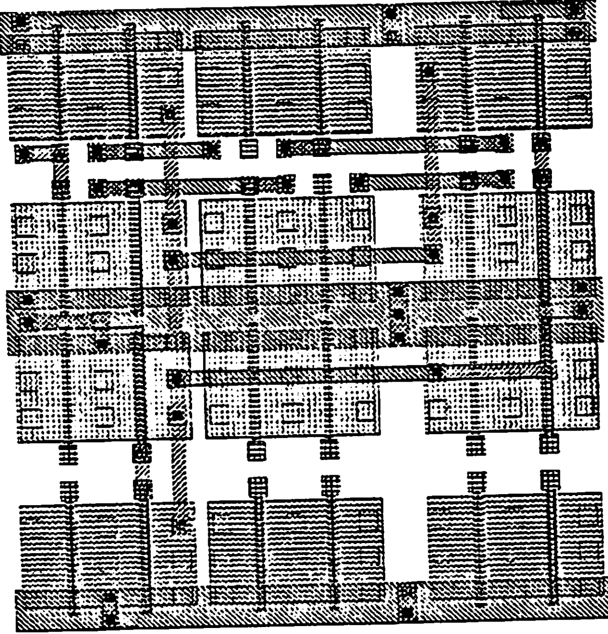
Memory and tri-state cells are no longer stretchable due to fixed template. However, the Mariner SOG cell template allows twelve horizontal second metal tracks to cross, so the pitch of a cell should be large enough for any data path. A tri-state buffer template and a latch template are shown in Fig. 3-7.

Another difference is that routing space is always the area of one or more templates. This quantum increase is very area inefficient. This is why much effort is paid to localize routing within the template of a gate.

The last difference is that the Mariner SOG cell template uses first metal for the power and ground. So after all the cells have been connected, large vias are added to bring the second metal power and ground to first metal terminals for external connection.



(a) Tri-state buffer template



(b) Latch template

Fig. 3-7 Tri-state buffer and latch templates in SOG style

CHAPTER 4

APPLICATION AND RESULTS

Both *OCTPLA* and *DAPAGES* were run using numerous examples obtained from the SPUR CPU chip. The results are analyzed and compared with the hand design. How these experiments reflect the effectiveness and the drawbacks of both tools is also described.

4.1. OCTPLA

The Instruction Unit (IU) Controller of the SPUR CPU chip contains 6 PLAs and some random logic. These PLAs are generally small. The largest one has 30 product terms. When the IU controller was designed, *espresso* was employed, and then a tiling based unfolded PLA generator, *Mpla* [mayo84], put together the cells in the cell library to form the PLAs.

Since the hand design approach also includes *espresso* for logic minimization, comparison between *OCTPLA* and *Mpla* shows only the difference due to folding. Both folded and unfolded versions of the above PLAs are generated by *OCTPLA* and *Mpla* using the same buffers. The areas of the above PLAs and two other PLAs, "trap" and "opcode", are summarized in Table 4-1.

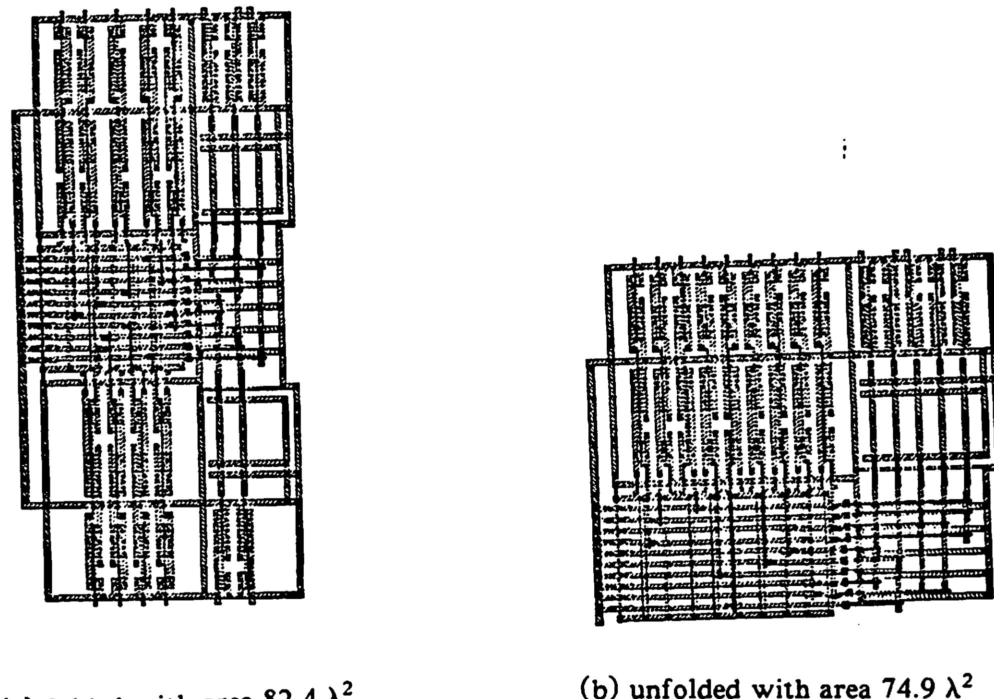
Three observations can be derived from this experiment. First the unfolded PLAs generated by *OCTPLA* and *Mpla* have approximately the same size. Since tiling approach can always generate very compact layout with prior careful planning on the tiles, the approach that *OCTPLA* employs compares favorably with *Mpla*. This is partly due to the tightly coupled structure of a PLA. The use of an efficient

compactor also contributes to this result.

The second observation deals with the use of folded PLAs. The output buffers used in these examples are planned to drive large loads, in the range of 5pf to 20pf. They are, thus, very large. So if a PLA is not that large or if not much folding can be achieved, an unfolded PLA may be smaller than a folded one. The largest PLA, "opcode", shown in Table 4-1 has a 27% reduction in area when the PLA is folded. Shown in Fig. 4-1 are both versions of PLA "p2" with 10 product terms. The area gained in folding almost offsets the area wasted in placing buffers on the other side of the PLA. The PLA "pf" with 14 product terms shown in Fig. 4-2 is worse. Not much folding can be derived in this PLA. So the unfolded one is actually smaller. Therefore, if buffers are large and the PLA cannot have much folding, a folded PLA is not rewarding.

PLA	Number of product terms	Unfolded (MPLA)	Unfolded (OCTPLA)	Folded (OCTPLA)
fet	14	89.8	80.5	121.4
pf	14	89.8	82.5	117.6
p1	6	59.7	55.4	77.6
p2	10	85.1	74.9	82.4
p3	30	192.9	173.2	195.6
p4	21	168.9	152.4	194.9
trap	35	722.0	648.1	475.2
opcode	69	720.1	648.1	475.2

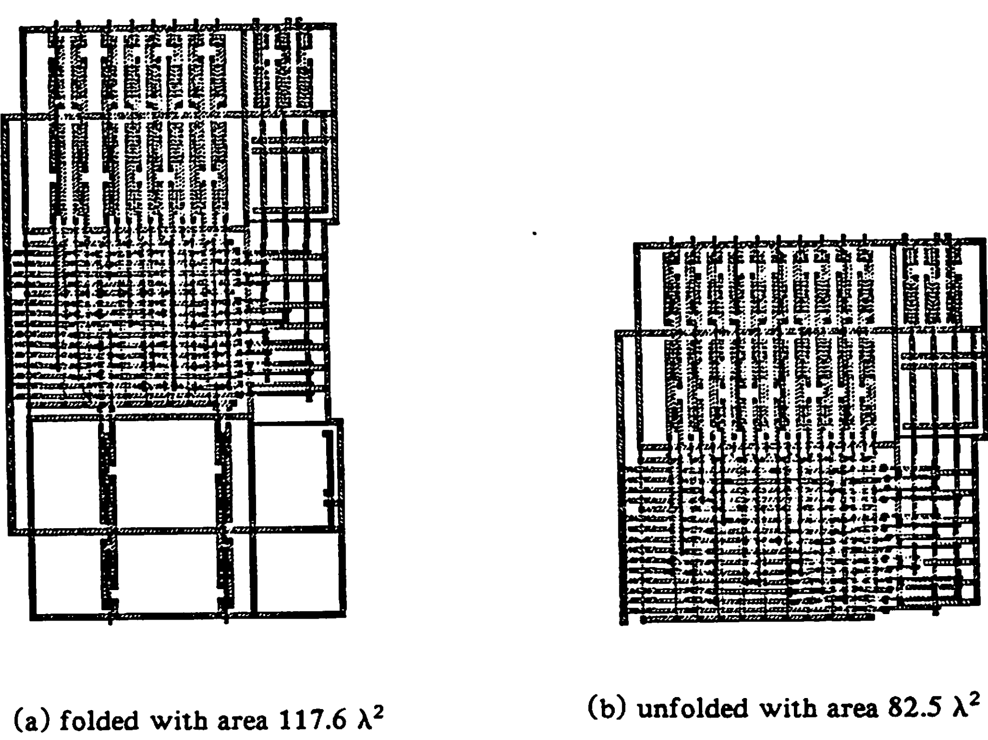
Table 4-1 Areas (in thousands of λ^2) of PLAs in the IU Controller



(a) folded with area 82.4 λ^2

(b) unfolded with area 74.9 λ^2

Fig. 4-1 Folded and unfolded versions of PLA "p2"



(a) folded with area 117.6 λ^2

(b) unfolded with area 82.5 λ^2

Fig. 4-2 Folded and unfolded versions of PLA "pf"

The timing performance of folded PLAs are, however, much better than that of unfolded PLAs. This can easily be understood by comparing the sizes of the cores of folded and unfolded PLAs. The cores of folded PLAs are smaller, thus reducing the capacitances of the wires. The reduction leads to shorter charging and discharging time. Therefore, folded PLAs are superior to unfolded PLAs in terms of timing performance.

The IU Controller has been implemented using the PLAs generated by *OCT-PLA*. The random logic is also implemented using a small PLA. Two of the PLAs are unfolded due to the reasons mentioned above. A Berkeley placement and routing system, *mosaico* [igusa87], has been used to place the PLAs and latches, and then route them. Both this IU Controller and the hand design one are shown in Fig. 4-3 and Fig. 4-4 respectively. The area ratio of this IU Controller and the hand design one is 1.2:1. The main reason for this difference is due to the placement of the input and output pins and the related routing. Ideally, the floorplanner specifies the constraints of the pin locations. This in turn puts constraints on the pins of the PLAs which can be met by the folding program, *Genie*. However, when the example is run, the PLAs are generated first and no constraint on the pin locations of the IU Controller is given. The pad placement program thus randomly places the input and output pins of the IU Controller. This unoptimized placement creates a lot of unnecessary routing. When doing a hand design, one has a pre-conceived floorplan for the whole chip, so the input and output pins are grouped together on both sides. Routing is thus more regular.

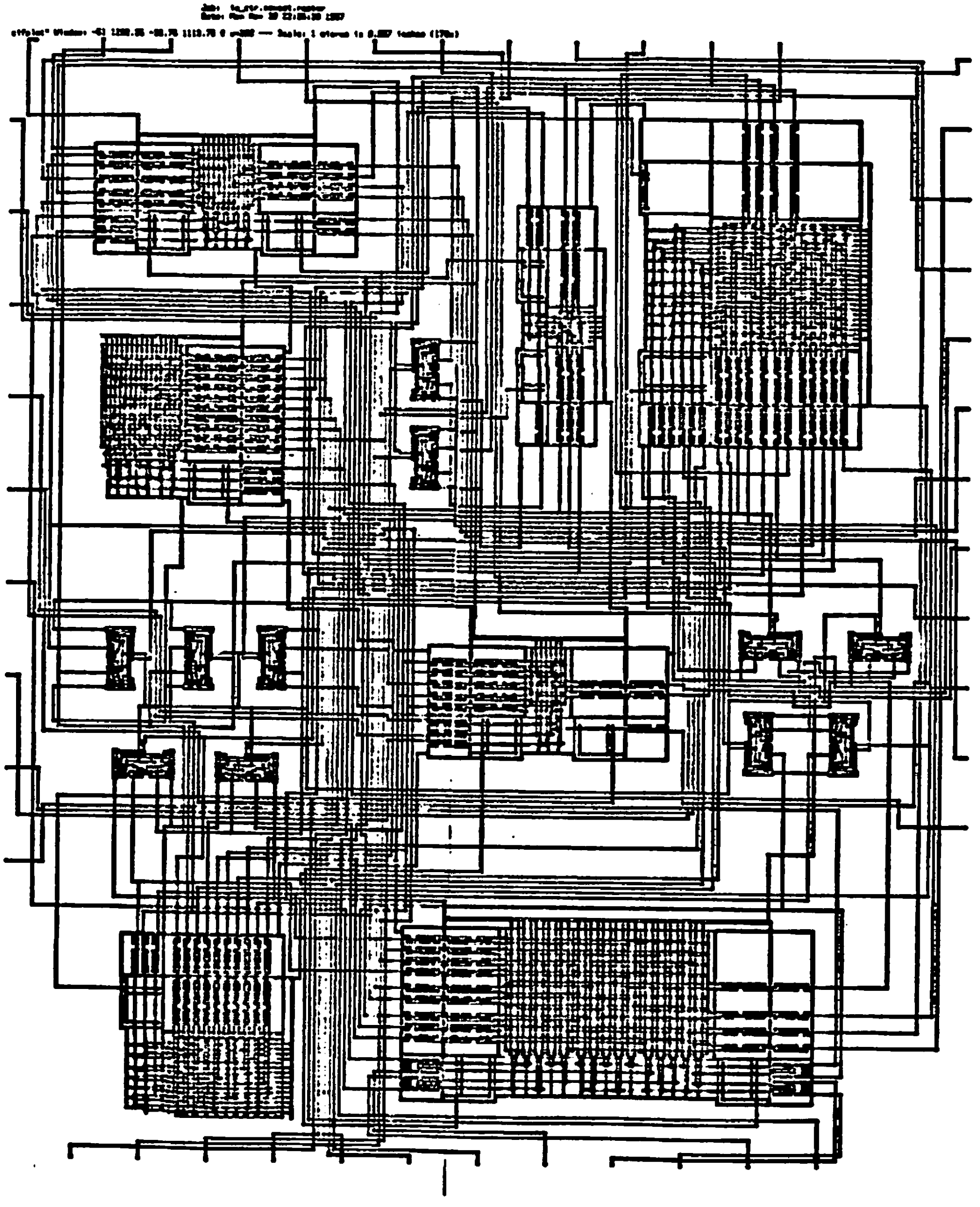


Fig. 4-3 IU Controller generated by *OCTPLA* and *mosaico* with area 2100λx1900λ

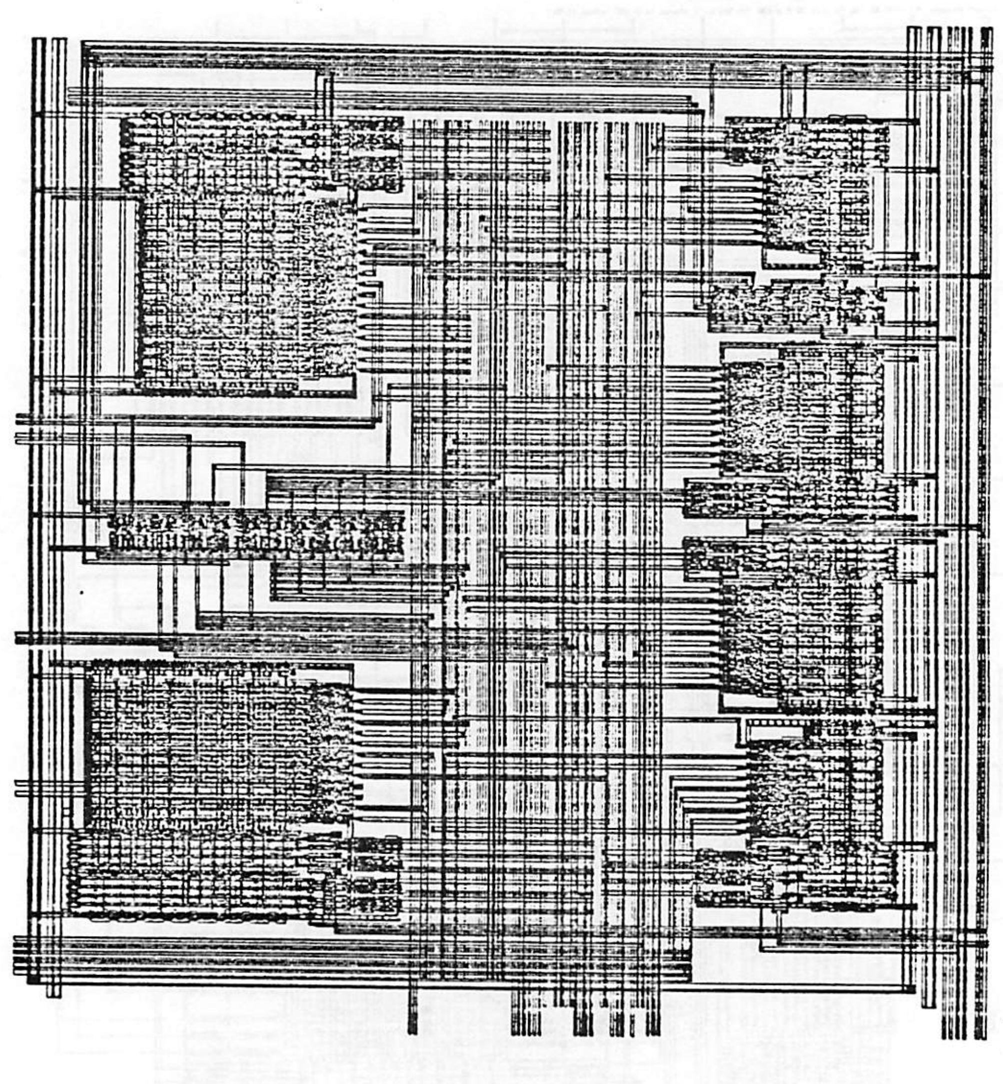


Fig. 4-4 Hand designed IU Controller with area 1775λx1879λ

4.2. DAPAGES

Major components of two data paths in the SPUR CPU chip are taken as examples. The first one is the 32-bit data path for the Instruction Unit (IU). The second one is the the Arithmetic Logic Unit (ALU) of the Lower Data Path. The SOG layouts of these cells are also shown.

4.2.1. Instruction Unit Data Path

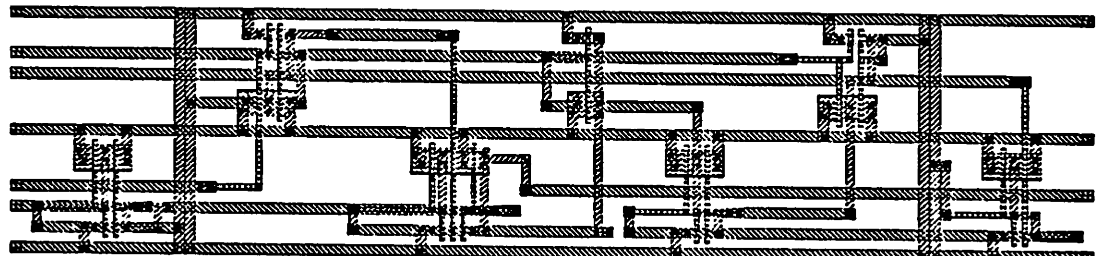
The IU data path is chosen since it is a very simple data path. It consists of a set of latches, followed by multiplexers (MUXes), and then tri-state buffers. All bit-slices of this data path are identical. So this is a good example to show some basic functions of *DAPAGES*.

The 4-to-1 multiplexer cell is the only combinational logic cell in this data path. Three of the four input busses are external, while the last input bus is the output of the latch. The other cells in the data path are stretchable cells in the library. Since this data path has only five local busses and the minimum pitch of a library cell allows eight busses, the library cells do not need stretching and the pitch of the MUXes are set to the minimum.

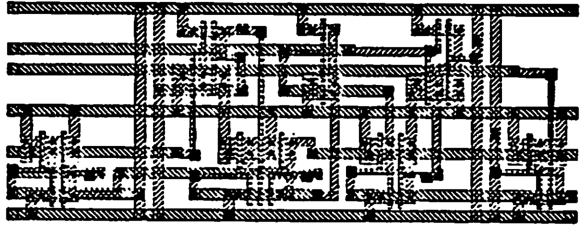
An unspaced MUX cell is shown in Fig. 4-5a. The area is 415 λ x 96 λ. Transistors are placed alternatively in the top half and the bottom half, hoping that they can share the space after compaction. Although most of the gates in this cell are 2-input NAND gate, they have different configurations to adapt to different locations of inputs and outputs. After compaction the cell with an area of 217 λ x 84 λ is shown in Fig. 4-5b. The area reduction after compaction is 46%.

The ratio of area of this cell to that of the hand design cell is 2.0 : 1. The big difference in area is mainly due to the use of different circuit design techniques. A

hand design cell can use pass transistors to realize the multiplexing function. However, since *MIS* cannot handle pass transistor logic, *DAPAGES* can only use standard logic gates, like 2-input and 3-input NAND gates. These standard gates are certainly larger than pass transistors. However, *DAPAGES* has a performance advantage since it uses fully-restored logic, instead of pass transistor logic which may have charge sharing problems. It should be noted that if *MIS* allowed pass transistor logic, *DAPAGES* would be able to generate the MUX using pass transistors.



(a). before compaction



(b). after compaction

Fig. 4-5 A MUX cell before and after compaction

A single bit-slice of the IU data path is shown in Fig. 4-6. The ratio of area of this bit-slice to that of the hand design bit-slice is 1.8 : 1. The large difference is, again, due to the use of different circuit design techniques, as explained before. An 8-bit IU data path is shown in Fig. 4-7.

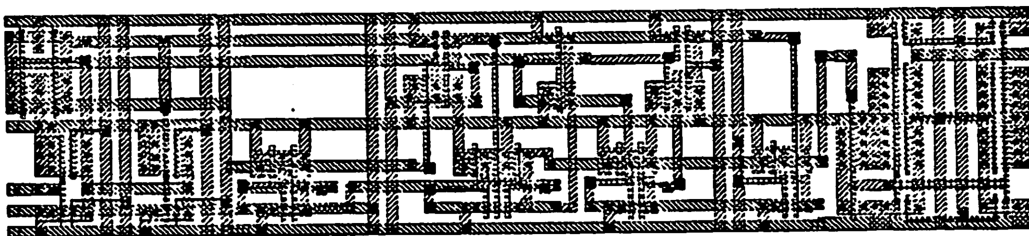


Fig. 4-6 A single bit-slice of the IU data path

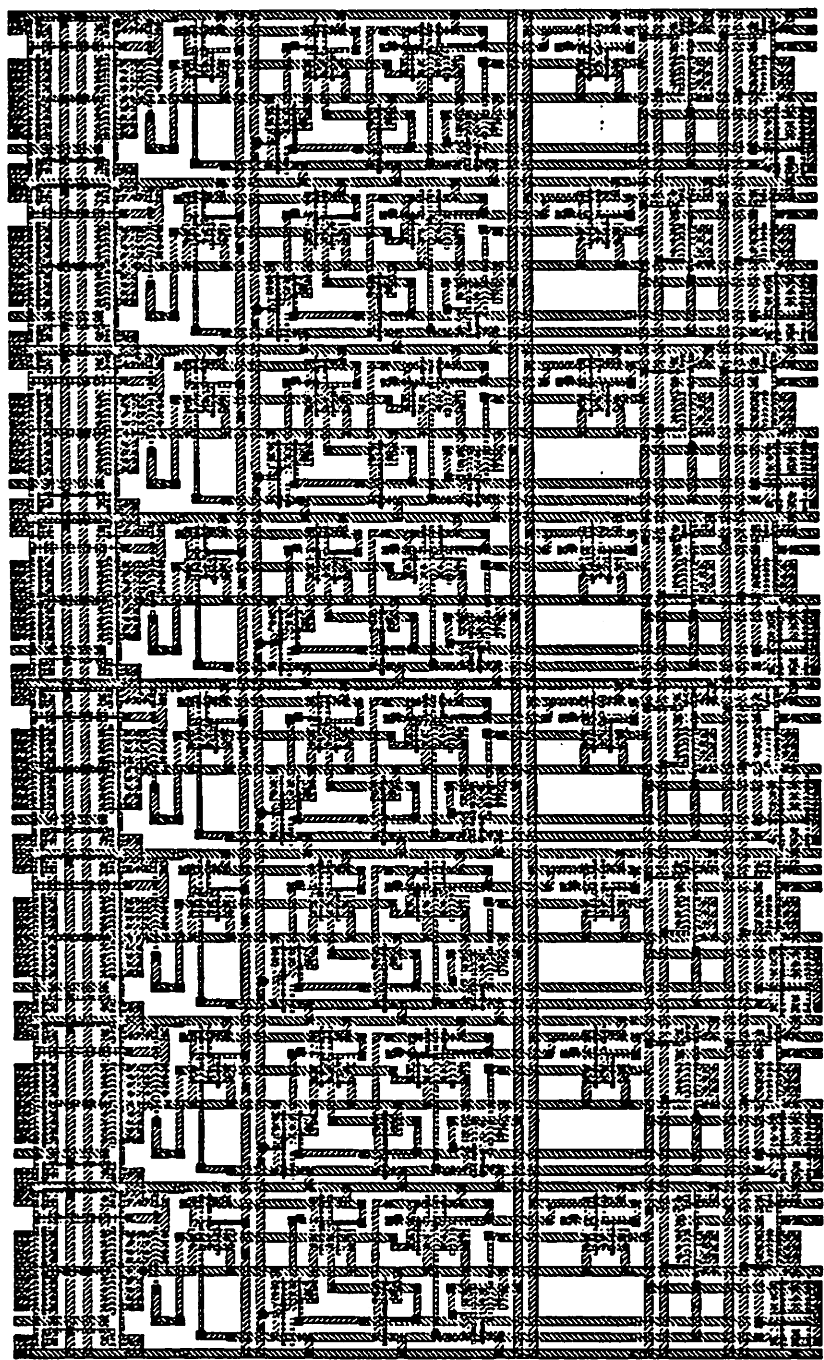


Fig. 4-7 An 8-bit IU data path

4.2.2. Lower Data Path

The Lower Data Path is the major data path in the SPUR CPU chip. It includes an adder, a shifter, latches, and some random logic. Of all these components, the shifter is the best one that can demonstrate how constraints of vertical signals are handled, since it has a lot of vertical signals that cross the top and bottom boundaries.

This shifter cell is able to shift left by 1, 2, or 3 bits, shift right by 1 bit, and perform no operation. So one bit signal may travel through at most three cells upward or at most one cell downward. In a typical shifter cell, there are five input signals, three coming from the bottom, one coming from the top, and one coming horizontally from the left of the bit-slice. There are also five control signals that choose the output from these five input signals. It is important to make sure that the input signal and the corresponding control signal are connected to the same gate. This is done by getting the exact locations of the input signals from the bottom cell.

A shifter cell with no constraints from the bottom is shown in Fig. 4-8b, and a shifter cell that is placed on top of the previous one is shown in Fig. 4-8a. The vertical signals are not placed beside the corresponding gates, otherwise routing must be required for the vertical signals between the two cells. So all the routing is done within each cell on the left as shown.

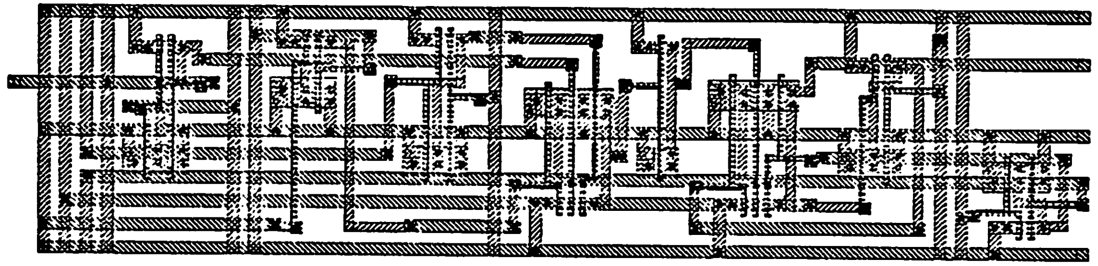


Fig. 4-8a. Top shifter cell with constraints from bottom

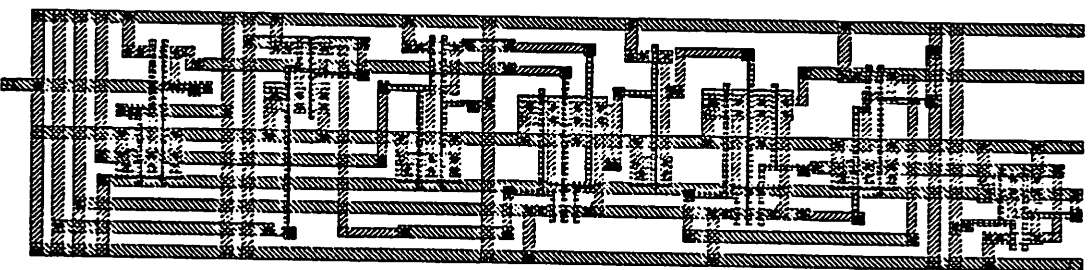


Fig. 4-8b. Bottom shifter cell with no constraint from bottom

The area of any one of the shifter cells is 350 λ x 84 λ. The ratio of this area to that of a hand design shifter cell is 1.5 : 1. The difference is due to the use of pass transistor logic in hand designed cells. However, this ratio is much smaller than the area ratio of the MUXes determined in section 4.2.1. The reason is that *DAPAGES* routed the vertical signals more efficiently than the hand designed cells in this case.

4.2.3. Sea-of-Gates Data Paths

Some terminology is defined before the results for SOG layout are analyzed. A transistor template is a vertical column of eight N- and P- transistors bounded by two power rails. A substrate contact template is a column of well and substrate plugs.

Since the SOG layout style has fixed transistor templates, one can expect the SOG cells to be larger than the custom style cells. A MUX cell, a shifter cell, and a latch cell are shown in Fig. 4-9. Unused transistor templates have been removed to show the input and output signals better. The horizontal bypassing signals in the latch cell are made short. This makes the program simpler, while spaces are guaranteed for the signals to bypass the whole cell.

The SOG cell generation routines are modified so that control signals can be grouped together if possible to save area. Both the MUX cell and the shifter cell demonstrate this point. If the control signals were not grouped together, each one would occupy a transistor template.

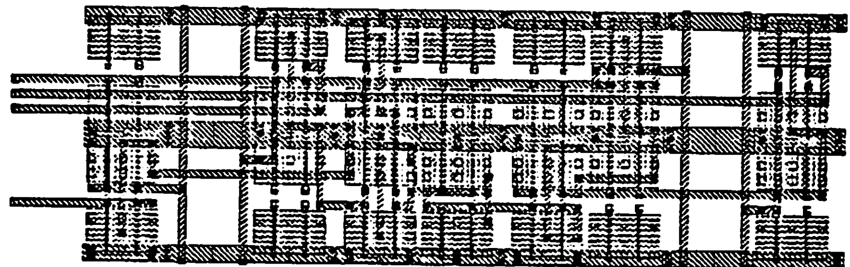


Fig 4-9a. A MUX cell in SOG style

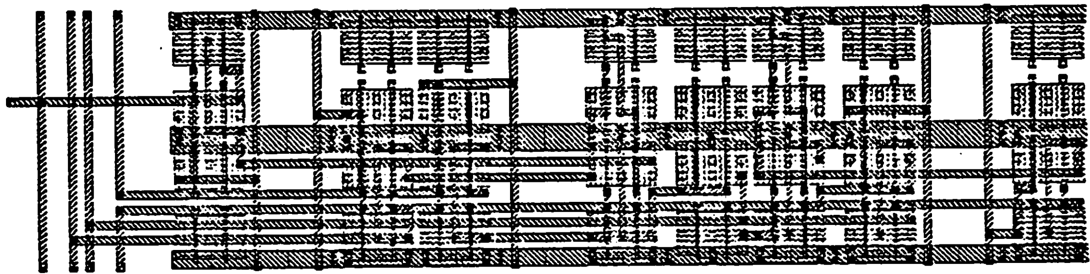


Fig 4-9b. A shifter cell in SOG style

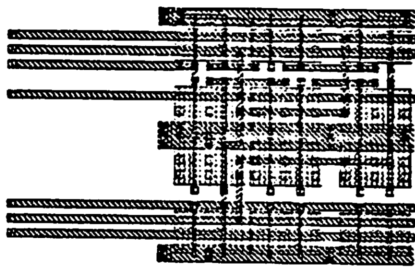


Fig 4-9c. A latch cell in SOG style

Since the Mariner SOG layout style has two transistor templates followed by one substrate contact template, sometimes dummy transistor templates have to be added to a fixed library cell to maintain this pattern. The latch cell shows this weakness. This is one of the reasons why a fixed cell template style is not preferred. A tri-state buffer cell can be generated similarly. The layout is very similar to the latch cell and therefore has not been shown.

The total number of transistor templates and the transistor templates used by gates or memory cells are given in Table 4-2. This is to show how much area is really used by the logic and how much is for routing. The results show that area utilization is between 60% to 70% which is very good.

CELL	Total # of transistor templates	Transistors Templates Used	Utilization Ratio
Shifter	13	8	61.5%
MUX	10	7	70%
Latch	5	3	60%
Tri-state	5	3	60%

Table 4-2. Area utilization of cells

The area ratios of cells generated by *DAPAGES* in both layout styles when compared with hand design cells are summarized in Table 4-3.

CELL	Custom Style by DAPAGES	SOG Style by DAPAGES
Shifter	1.5	4.1
MUX	2.0	6.9
Latch	2.1	8.9
TriBuf	1.2	5.5

Table 4-3. Area Ratios of cells generated by *DAPAGES* when compared with hand design cells

CHAPTER 5

FUTURE RESEARCH

The previous chapters explain the approaches taken by both *OCTPLA* and *DAPAGES* to optimize their outputs, with the illustration of some results using the SPUR CPU chip as an example. Drawbacks of each system mentioned in the last chapter lead to the following discussion on possible future work to enhance and further optimize the systems.

5.1. OCTPLA

OCTPLA can be easily modified to handle multiply folded PLAs. Since *genie* has the capability to do multiple folding, the only change in *OCTPLA* is then the repetition of the mapping of transistors and buffers of the multiple **AND** and **OR** planes. The cell library remains the same.

One way to further optimize the area of the PLAs is to merge output buffers in adjacent columns. This requires terminal merging done by compaction. Careful layout and terminal implementation are needed.

The cell library can be expanded to handle different layout styles and to include testable circuits. Dynamic PLAs can be realized as described in Chapter 2. This only requires three more cell templates. Output buffers and the circuitry in the interface of the **AND** and **OR** planes can be modified so that users can monitor the outputs of the PLA and the product terms in the **AND** plane.

5.2. DAPAGES

DAPAGES is in an experimental stage. It is less mature, compared to *OCT-PLA*, thus more future work on *DAPAGES* is expected.

More gate generation modules should be written to enlarge the static CMOS gate library for *MIS*. This can further reduce the area of the data path. Up to now, an inverter, a 2-input NAND, and a 3-input NAND modules have been successfully implemented. A 4-input NAND, some NORs, and a 2-input exclusive-OR generation modules are desirable. Transmission-gate generation modules should be introduced if *MIS* is modified to allow the use of transmission gates.

A change in the data structure can simplify the netlist requirements. The current version requires that a unit cell belongs to only one row and one column in the graph. It is sometimes difficult for users to separate a logic block into exactly **n** bit-slices, where **n** is the number of bits of the whole data path. It is more convenient for them to describe a unit cell, like an adder cell, that belongs to more than one row or one column. This requires a change in the data structure of the graph and a corresponding change in the message passing algorithm. In addition to these changes, a two-dimensional placement program for the gates of a cell is also needed in case the cell belongs to more than one row.

CHAPTER 6

CONCLUSION

Two module generation systems targeted at building PLA based control units and data paths of microprocessor chips are described in this report. The PLA design system, *OCTPLA*, generates singly folded PLAs for a number of layout styles using hierarchical symbolic design. The whole system uses only twelve templates in the cell library. The CAD tools within *OCTPLA* perform logic minimization, topological optimization, and electrical optimization. The data path generation system, *DAPAGES*, generates the entire data path using a mixture of automatically generated cells and modifiable cells. It handles both global constraints and local constraints by message passing in an object-oriented environment to achieve pitch matching of cells and terminals. It handles both custom layout style and SOG layout styles.

Numerous examples from the SPUR CPU chip were run on these systems to illustrate the characteristics of the systems and determine the drawbacks. Both *OCTPLA* and *DAPAGES* were successful in generating important modules in the SPUR CPU chip.

Possible future work on *OCTPLA* includes extending its capability to handle multiply folded PLAs, further optimizing the area by merging cell instances, and expanding the cell library for different layout styles and testable circuits. More gate generation modules to optimize both area and performance of the resulting data paths, and simplifying the input requirement by changing the data structure are recommended for *DAPAGES*.

REFERENCES

[bart85]
M. Bartholomeus et. al., "PLASCO: A Procedural Silicon Compiler for PLA based Systems," *Proceedings of the Custom Integrated Circuits Conference*, May 1985.

[batali81]
J. Batali et. al., "The DPL/Daedalus Design Environment," *Proceedings of VLSI 81*, Edinburgh, 1981.

[burns86]
J. Burns and A. R. Newton, "SPARCS: A New Constraint-Based IC Symbolic Layout Spacer," *Proceedings of the Custom Integrated Circuits Conference*, May 1986.

[chris88]
"Mariner: A Sea of Gates Layout System," *Master Report, University of California*, 1988, *To be published.*

[dev86]
S. Devadas and A. R. Newton,"GENIE: A Generalized Array Optimizer for VLSI Synthesis," *Proceedings of the 23rd Design Automation Conference*, 1986.

[glas80]
L. Glasser and P. Penfield, Jr., "An Interactive PLA generator as an archetype for a new VLSI design methodology," *International Symposium on Circuits and Systems*, 1980.

[harr86]
D. Harrison et. al., "Data Management and Graphics Editing in the Berkeley Design Environment," *Proceedings of the 23rd Design Automation Conference*, 1986.

[hed85]
K. Hedlund, "Electrical Optimization of PLAs," *Proceedings of the 22nd Design Automation Conference*, 1985.

[hill85]
M. Hill et. al., "SPUR: A VLSI Multiprocessor Workstation," Report No. UCB/CSD 86/273, Computer Science Division (EECS), U.C. Berkeley, December 1985.

[igusa87]
J. Burns et. al., "Mosaico: An Integrated Macro-cell Layout System," *Proceedings of VLSI 87*, Vancouver, 1987.

[kring87]
C. Kring, "Mkarray: an Assembler for Arbitrary Arrays GEM: a Gate-Matrix Module Generator," *Master Report, University of California*, 1988, *to be published*.

[lattin79]
B. Lattin, "VLSI Design Methodology: The Problem of the 80's for Microprocessor Design," *Proc. of the Caltech Conference on Very Large Scale Integration*, C.L. Seitz, Pasadena, Calif., 248-252, 1979

[layer87]
L. Layer, "Analysis of Sea-of-Gates Template and Cell Library Design Issues," *Master Report, University of California*, 1987.

[lin87]
B. Lin, "OCTLATTE — A general purpose net list partitioning and one dimensional placement tool for oct," *Berkeley CAD Tools User's Manual*, 1987.

[lipton82]
R. Lipton et. al., "ALI: a Procedural Language to Describe VLSI Layouts," *Proceedings of the 19th Design Automation Conference*, June 1982.

[mah84]
G. H. Mah and A. R. Newton, "PANDA: A PLA generator for multiply folded PLAS," *Proc Int. Conf. on Computer-Aided Design 1984, pp 122-124*.

[marsh86]
T. Marshburn et. al., "DATAPATH: A CMOS Data Path Silicon Assembler," *Proceedings of the 23rd Design Automation Conference*, June 1986.

[mayo84]
R. Mayo, "Combining Graphics and Procedures in a VLSI Layout Tool: The Tpack System," *Master's Report, University of California*, 1984.

[micheli83]
G. De Micheli, "Computer-Aided Synthesis of PLA-Based Systems", *PhD Dissertation, University of California*, 1983.

[powell84]
S. Powell et. al., "An Automated, Low Power, High Speed Complementary PLA Design System for VLSI Applications," *Proceedings of the International Conference On Computer Design*, Oct. 1984.

[rowson87]
J. Rowson et. al., "A Datapath Compiler for Standard Cells and Gate Arrays," *Proceedings of the Custom Integrated Circuits Conference*, May 1987.

[rudell86]
R. Rudell et. al., "Multiple-Level Logic Optimization System," *Proceedings of Int. Conf. of Computer Aided Design*, 1986.

[rudell85]
R. Rudell and A. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for Multiple Valued Boolean Minimization," *Proceedings of the Custom Integrated Circuits Conference*, May 1985.

[ruetz86]
P. Ruetz et. al., "Automatic Layout Generation of Real-Time Digital Image Processing Circuits," *Proceedings of the Custom Integrated Circuits Conference*, May 1986.

[segal87]
R. Segal, "BDSYN: Logic Description Translator BDSIM: Switch-Level Simulator," *Master Report, University of California*, 1987.

[shrobe82]
H. Shrobe, "The Data Path Generator," *MIT Conference on Advanced Research in VLSI*, Jan. 1982.

[tan87]
D. Tan and N. Weste, "Virtual Grid Symbolic Layout 1987," *Proceedings of the International Conference On Computer Design*, Oct. 1987.

[weste87]
N. Weste et. al., "The Symbolics Ivory Design and Verification Strategy," *Proceedings of the International Conference On Computer Design*, Oct. 1987.

[wood86]
G. Wood and H. Law, "SKILL — An Interactive Procedural Design Environment," *Proceedings of the Custom Integrated Circuits Conference*, May 1986.

APPENDIX A

Manual Page of OCTOPUS

NAME

octopus - A Programmable Logic Array (PLA) module generator

SYNOPSIS

octopus [options] cell_out

DESCRIPTION

OCTOPUS is a layout generator that takes the output of an array folder GENIE and generates an unspaced layout of the PLA.

INPUT / OUTPUT

The input to OCTOPUS is a text file that includes a personality matrix of a folded or unfolded PLA. The view of the output OCT cell is optional and defaults to "unspaced". Users have to run SPARCS to get the PLA laid out according to the design rules.

The input file can be generated by GENIE and a modified version of GENIE, GENIE.DIST, as follow:

```
genie.dist -pla espresso_output_file | genie -pmr
```

OPTIONS

-i specifies the name of the input personality matrix file. The default is ./final.

-F generates an unfolded PLA. All the input and output buffers are put on the top of the PLA.

-s sets the spacing of the horizontal metal-1 ground line in the AND plane and that of the vertical metal-1 ground line in the OR plane in terms of rows and columns respectively. Metal-1 ground lines are introduced to bring the resistance of diffusion ground line down.

-v specifies the view of the output OCT cell.

-t specifies the technology of the cell.

-clib specifies the names of the library cells in a text file. This will be described later in this manual.

-ru tells where the RULE cell is placed. This rule cell contains all the connection rules of the PLA.

-cib uses clocked input buffer.

-cob uses clocked output buffer.

-cp uses clocked output buffer with the output set to low while the clock is not asserted.

-d turns on the debugging mode, thus allowing OCTOPUS to print the debugging information about the cell instances.

-do uses different output buffers in the same PLA. When this option is used, the option 'clib' must also be used. The signals that use different output buffers and their corresponding buffer names are listed in the same text file that the option 'clib' specifies.

-g tells OCTOPUS to print the constraint graphs used by MKARRAY in ./mkarray.x and ./mkarray.y.

-p specifies the threshold number of transistor for the AND plane pull up devices. When this number is exceeded for a row in the AND plane, a larger pull up is used to minimize the delay in this row. The default is 10.

-W tells OCTOPUS to place the PLA cells without wiring. This is useful for debugging when users change the cell templates that cause non-Manhatten wires.

CELL LIBRARY

Users can use their own cells, in which case another input file should be provided to indicate the names of the cells in the cell library. Users should change the second entry in the appropriate line in the following file:

```
LSIDEBUF /cad/lib/technology/scmos/octopus/Lside_buf
```

RSIDEBUF /cad/lib/technology/scmos/octopus/Rside_buf
ANDM1GND /cad/lib/technology/scmos/octopus/and_GND
ORM1GND /cad/lib/technology/scmos/octopus/or_GND
CONNECT /cad/lib/technology/scmos/octopus/connect
INBUF /cad/lib/technology/scmos/octopus/in_buf
MIDBUF /cad/lib/technology/scmos/octopus/mid_buf
LEFTORGND /cad/lib/technology/scmos/octopus/or_GND_L
RIGHTORGND /cad/lib/technology/scmos/octopus/or_GND_R
OUTBUF /cad/lib/technology/scmos/octopus/out_buf
PULLUP /cad/lib/technology/scmos/octopus/pull_up
PULLUPGND /cad/lib/technology/scmos/octopus/pull_up_GND
ORGNDBUF /cad/lib/technology/scmos/octopus/GND_or_buf
ANDXSTER /cad/lib/technology/scmos/octopus/and_xster
ORXSTER /cad/lib/technology/scmos/octopus/or_xster
LARGEPULLUP /cad/lib/technology/scmos/octopus/large_pull_up

Each cell has its own function which is described as follows:

-LSIDEBUF
Connector placed on the left of the input buffer.

-RSIDEBUF
Connector placed on the right of the output buffer.

-ANDM1GND
Connector for the horizontal metal-1 ground line in the AND plane.

-ORM1GND
Connector for the vertical metal-1 ground line in the OR plane.

-CONNECT
The interface of the AND plane and OR plane.

-INBUF
Input buffer.

-MIDBUF
Connector between the input and output buffers.

-LEFTORGND
Connector on the left of the OR plane for the horizontal ground diffusion.

-RIGHTORGND
Connector on the right of the OR plane for the horizontal ground diffusion.

-OUTBUF
Output buffer.

-PULLUP
Pull up device for the AND plane.

-PULLUPGND
Connector replacing the pull up device for the AND plane when a row is used for metal-1 ground line only.

-ORGNDBUF
Connector replacing the output buffer when a column is used for vertical metal-1 ground line only.

-ANDXSTER
Transistor in the AND plane.

-ORXSTER
Transistor in the OR plane.

-LARGEPULLUP
Large pull up device for the AND plane. This is used when the number of transistors in a row exceeds a given or default value.

Sometimes different output buffers may be used in the same PLA to ensure correct timing and save some external logic. In this case, the signals that use different output buffers are listed in the same file. This information may follow the previous texts by a line of one or more asterisks. No separator is needed if default cells are used. An example is shown as follows:

```
****
output2 ~/lib/out_buf_clk_pre
output5 ~/lib/out_buf_clk
```

In this example, signals 'output2' and 'output5' use different buffers, while the rest of the outputs use the default or given buffers.

After the cells have been changed, users should also update the rule cell. The following is the set of rules embedded in the rule cell:

-pull_up
For the column of AND plane pull-up devices.

-andOrIntRule
For the column of connectors between the AND and OR plane.

-andInRule
Connect the gate of the AND plane transistors to the non-inverted output signal of the input buffer.

-andInBarRule
Connect the gate of the AND plane transistors to the inverted output signal of the input buffer.

-andDiffRule
Connect the diffusion of the AND plane transistors.

-orColRule
Connect the drain of the OR plane transistors to the output buffer.

-orSigRule
Connect the gate of the OR plane transistors to the product terms in the AND plane.

-lastColRule
For the last column of PLA.

-bufRule
For the row of buffers and connectors.

-rowCoreRule
For the row in the AND plane.

-orDiffRule
Connect the diffusion of the OR plane transistors.

-coreGndRule
Connect the metal-1 ground lines in both planes.

-orXsterRule
Connect the gates of OR plane transistors.

Some OCT properties, like DIRECTION and TERMTYPE, are required in some cells. Users can look at the "addProp" file in the cell library. Run bdnet on this file to attach the OCT properties.

DIAGNOSTICS

If users manually change the number of rows or columns in the personality matrix without modifying the related information in the file, the program will die and may not give enough error messages to point out the incoherence in the file.

BUGS

There is a bug in GENIE that may not constraint an input and its complement to stay together. This may occurs when there is an empty column in the PLA and the signals are folded. This is not detected by OCTOPUS now. So users should make sure the input file is correct before running OCTOPUS on it. The following is an example that shows the error in the file:

```
( row 3 s1* 0 )
( row 4 s1 0 s2* 5 )
( row 5 s2 0 s3* 9 )
( row 6 s3 0 )
```

Since "s2" and "s2*" should stay together, users can swap "s2" and "s3" in row5 or "s1" and "s2*" in row4, and change the starting position for the signals. If signal "s3*" is empty, then starting place of "s2" can be set to 0:

```
( row 3 s1* 0 )
( row 4 s1 0 s2* 5 )
( row 5 s3* 0 s2 0 )
( row 6 s3 0 )
```

Up to now most of the PLA's generated can be compacted by SPARCS. It is possible that a PLA cannot be compacted. This problem is being investigated.

FILES

/cad/lib/technology/scmos/octopus - the default cell library
/cad/lib/technology/scmos/octopus/PLArules:symbolic - the default rule cell
/cad/lib/technology/scmos/octopus/addProp

SEE ALSO

genie(1) mkarray(1) sparcs(1) espresso(1) bdnet(1)

AUTHOR

Shau-Lim Chow
schow@ic.Berkeley.EDU
(415) 642-5322

APPENDIX B

Manual Page of OCTPLA

NAME

octpla - A Programmable Logic Array (PLA) module generation pipeline

SYNOPSIS

octpla [options] input_file

DESCRIPTION

OCTPLA is a shell script that calls ESPRESSO, GENIE, OCTOPUS, SPARCS, and VULCAN to generate a PLA. The input to this pipeline is a text file. By default, it is an input file for ESPRESSO. If ESPRESSO is not run, then it will be an input file for GENIE. If GENIE is not run, then it will be a personality matrix input file for OCTOPUS. The output of this pipeline is an OCT cell that contains the PLA layout. A more detailed description of the layout can be found in the manual page of OCTOPUS.

OPTIONS

noEsp	Skip ESPRESSO and start from GENIE.
noGenie	Skip GENIE and start from OCTOPUS.
noSparcs	Skip SPARCS. The final output view is "unspaced".
output	Specify the output OCT cell.
unfolded	Generate an unfolded PLA.
debug	Turn on debug mode.
clib	specify cell library for OCTOPUS.
cob	Use clocked output buffer.
cib	Use clocked input buffer.
cp	Use clocked precharged output buffer.
do	Use different output buffers in the same PLA.
p	Specify the threshold transistor number for AND plane pull up devices.
ru	Specify the rule cell for OCTOPUS.
s	Specify metal-1 ground line separation.
W	Make an unwired PLA.

BUGS

Please report to schow@ic.

SEE ALSO

espresso(CAD1)
genie(CAD1)
octopus(CAD1)
sparcs(CAD1)
vulcan(CAD1)

AUTHOR

Shau-Lim Chow
schow@ic.Berkeley.EDU
(415) 642-5322

APPENDIX C

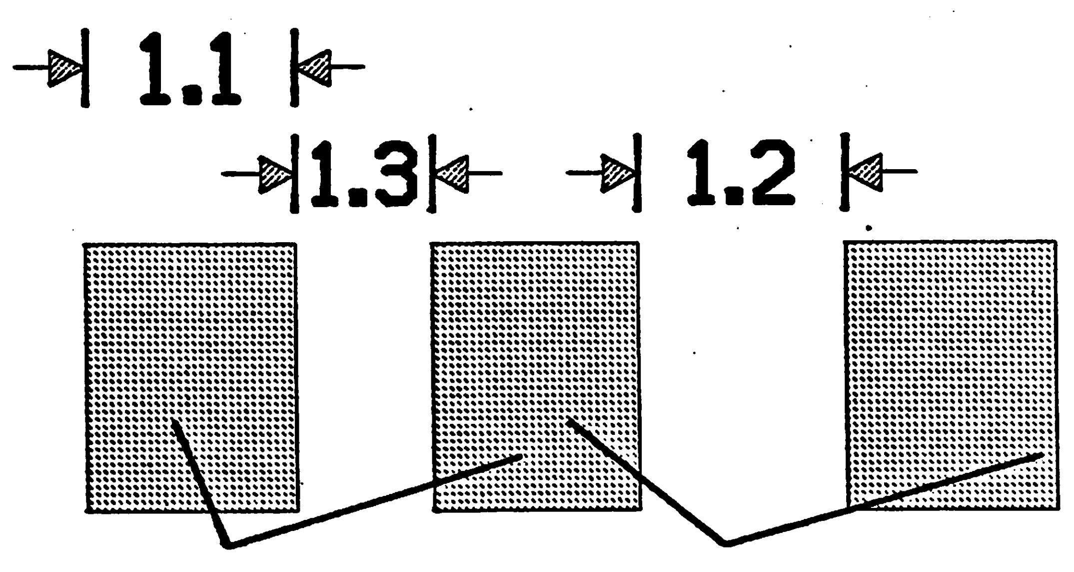
Design Rules

LAYER NAMES AND COLORS

LAYER	CIF	CALMA #	COLOR
WELL	CWG	53	
PWELL	CWP	41	
NWELL	CWN	42	
ACTIVE	CAA	43	
SELECT	CSG	54	
PSELECT	CSP	44	
NSELECT	CSN	45	
POLY	CPG	46	
CONT to POLY	CCP	47	
CONT to ACT	CCA	48	
METAL1	CMF	49	
VIA	CVA	50	
METAL2	CMS	51	
OVERGLASS	COG	52	

1. WELL (NWELL,PWELL)

		LAMBDAS
1.1	WIDTH	9
1.2	SPACE DIFF. POT.	9
1.3	SPACE SAME POT.	0 OR 4



NOTE: IF BOTH P AND N WELLS SUBMITTED, THEY MAY NOT OVERLAP BUT THEY MAY BE COINCIDENT.

2. ACTIVE

		LAMBDAS
2.1	WIDTH	2
2.2	SPACE	3
2.3	SOURCE/DRAIN ACTIVE TO WELL EDGE	5
2.4	SUBS./WELL CONTACT, ACTIVE TO WELL EDGE	3

2.2
P- REGION
N+ N+ P+
2.1 2.3 2.4
2.3 2.4
P+ P+ N+
N- REGION
2.2
2.1

NOTE: MIN XTOR WIDTH 3 LAMBDA

3. POLY

		LAMBDAS
3.1	WIDTH	2
3.2	SPACE	2
3.3	GATE OVERLAP OF ACTIVE	2
3.4	ACTIVE OVERLAP OF GATE	2
3.5	FIELD POLY TO ACTIVE	1

3.5

3.1

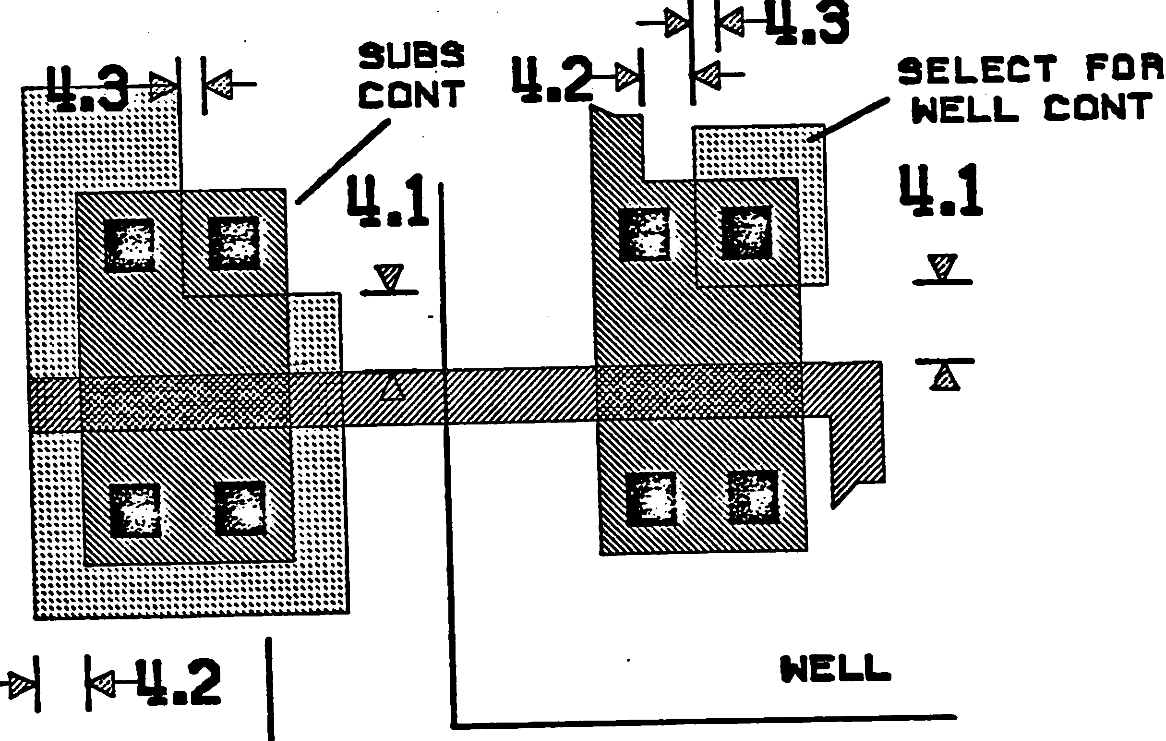
3.2

3.4

3.3

4. SELECT (PSELECT,NSELECT)

		LAMBDAS
4.1	SELECT SPACE (OVERLAP) TO (OF) CHANNEL TO ENSURE ADEQUATE SOURCE/DRAIN WIDTH	3
4.2	SELECT SPACE (OVERLAP) TO (OF) ACTIVE	2
4.3	SELECT SPACE (OVERLAP) TO (OF) CONTACT TO WELL OR SUBSTRATE	1
4.4	MIN WIDTH AND SPACE	2

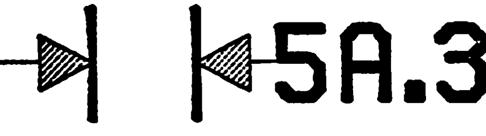


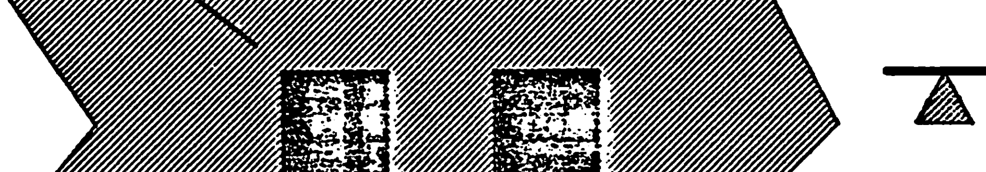
NOTE: IF BOTH PSELECT AND NSELECT SUBMITTED, THEY MAY BE COINCIDENT BUT MUST NOT OVERLAP

5A. SIMPLER CONTACT TO POLY

		LAMBDAS
5A.1	CONTACT SIZE EXACTLY	2x2
5A.2	POLY OVERLAP	2
5A.3	SPACING	2

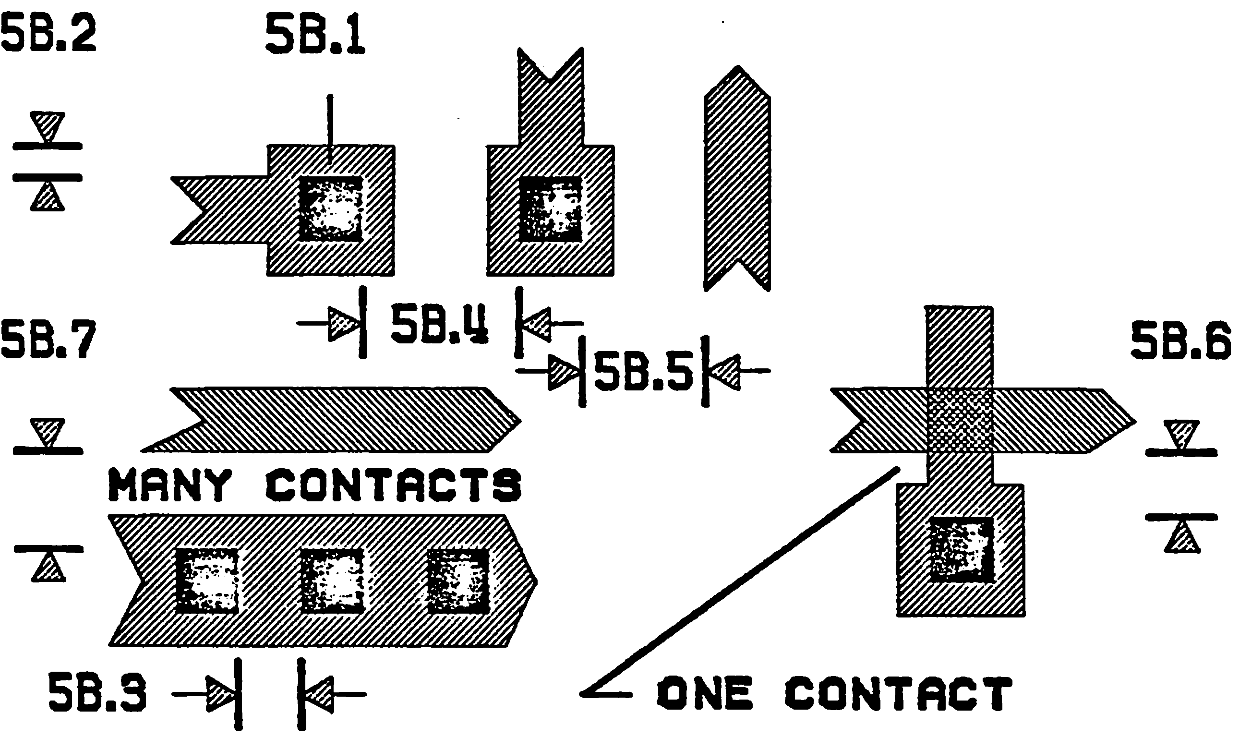
5A.1 5A.2





5B. DENSER CONTACT TO POLY

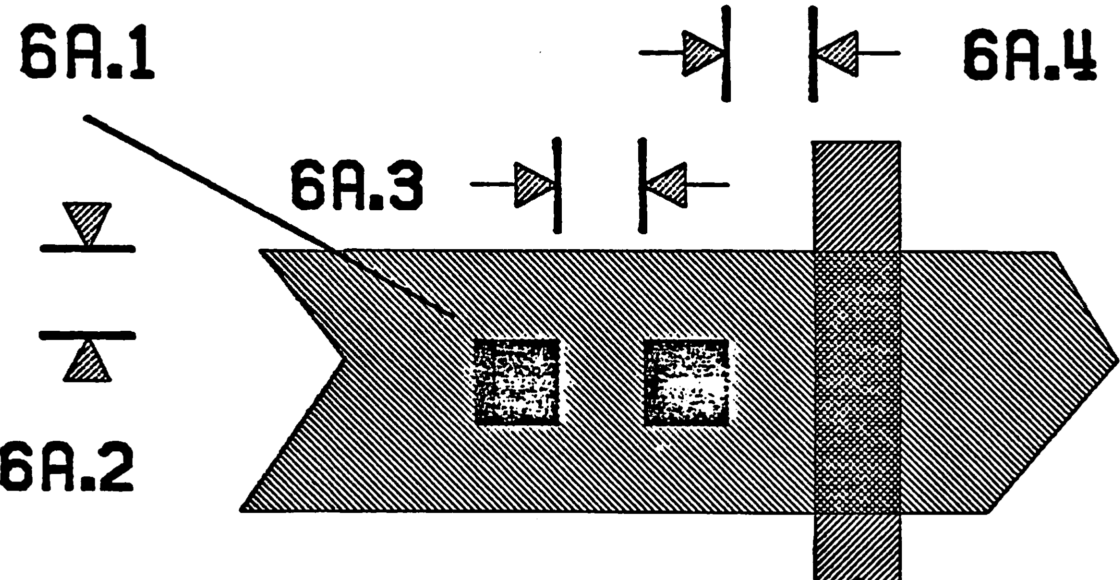
		LAMBDAS
5B.1	CONTACT SIZE, EXACTLY	2x2
5B.2	POLY OVERLAP OF CONTACT	1
5B.3	SPACING ON SAME POLY	2
5B.4	SPACING ON DIFF POLY	5
5B.5	SPACE TO OTHER POLY	4
5B.6	SPACE TO ACT, ONE CONTACT	2
5B.7	SPACE TO ACT, MANY CONTACTS	3

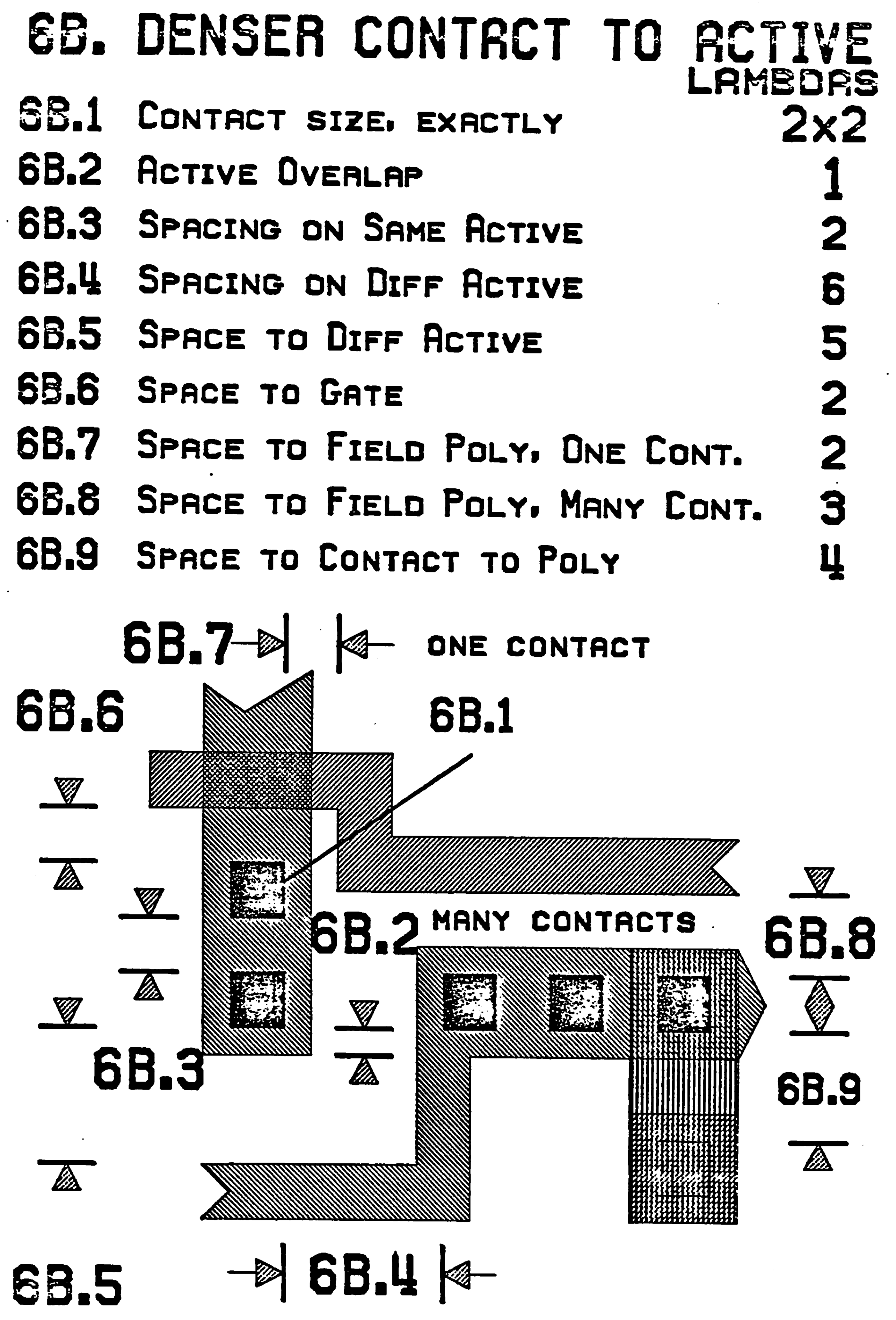


NOTE: YOUR ASSOCIATING CONTACTS WITH POLY OR ACTIVE ALLOWS MOSIS TO INDEPENDENTLY BLOAT THE LAYER AND THE LAYER OVERLAP OF THE CONTACT

6A. SIMPLER CONTACT TO ACTIVE

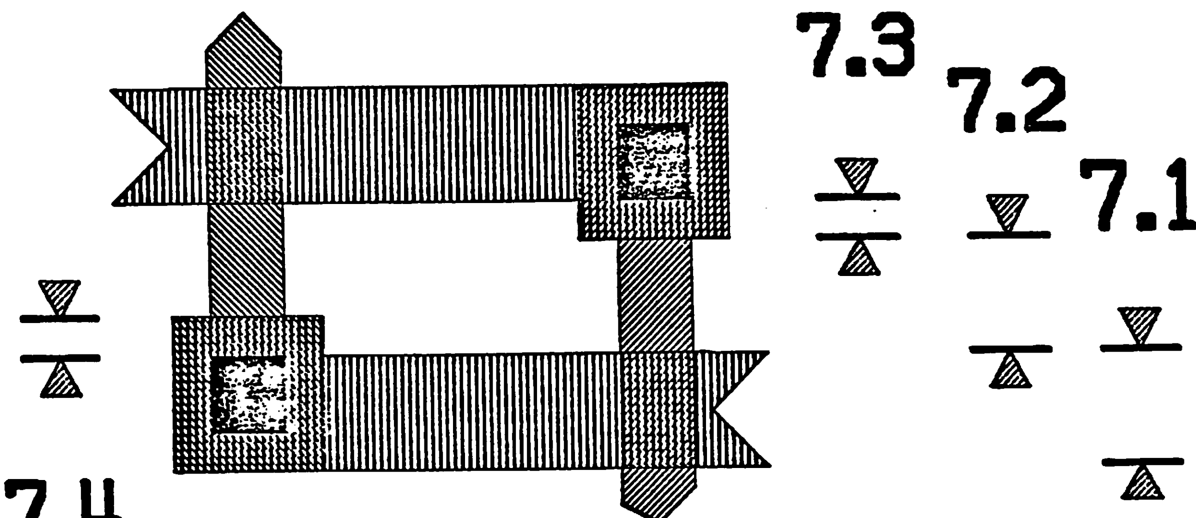
		LAMBDAS
6A.1	CONTACT SIZE EXACTLY	2x2
6A.2	ACTIVE OVERLAP	2
6A.3	SPACING	2
6A.4	SPACE TO GATE	2




6B. DENSER CONTACT TO ACTIVE
LAMBDAS
6B.1 CONTACT SIZE, EXACTLY 2x2
6B.2 ACTIVE OVERLAP 1
6B.3 SPACING ON SAME ACTIVE 2
6B.4 SPACING ON DIFF ACTIVE 6
6B.5 SPACE TO DIFF ACTIVE 5
6B.6 SPACE TO GATE 2
6B.7 SPACE TO FIELD POLY, ONE CONT. 2
6B.8 SPACE TO FIELD POLY, MANY CONT. 3
6B.9 SPACE TO CONTACT TO POLY 4
6B.7
ONE CONTACT
6B.6
6B.1
6B.2
MANY CONTACTS
6B.8
6B.3
6B.9
6B.5
6B.4

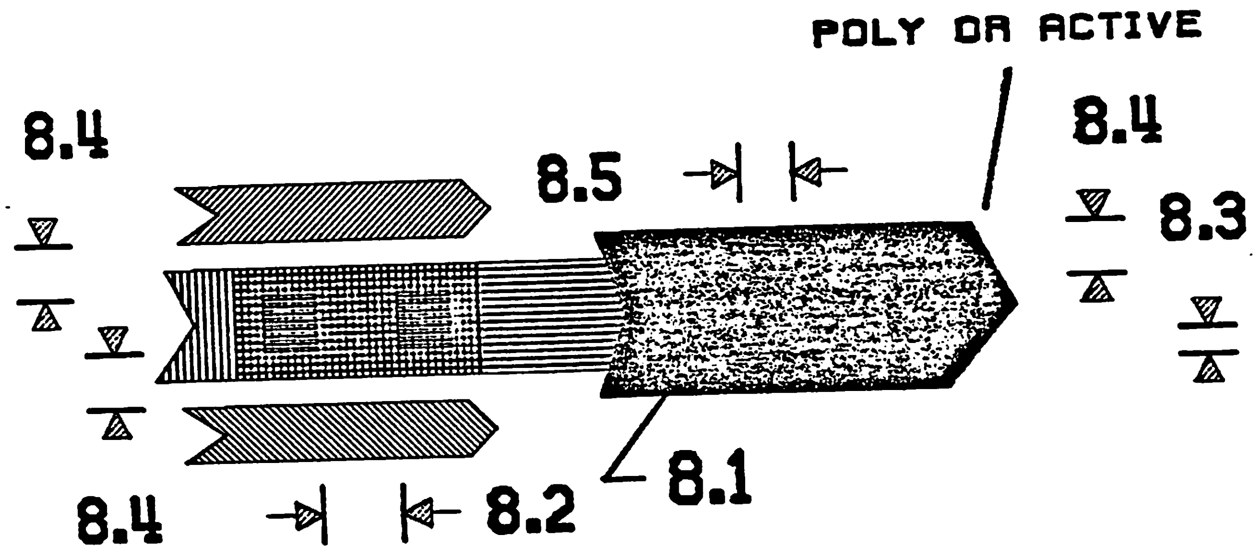
7. METAL1

		LAMBDAS
7.1	WIDTH	3
7.2	SPACE TO METAL1	3
7.3	OVERLAP OF CONTACT TO POLY	1
7.4	OVERLAP OF CONTACT TO ACTIVE	1



8. VIA

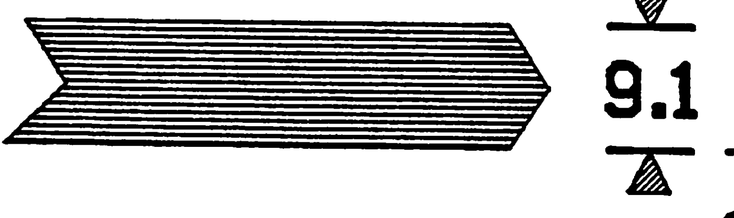
		LAMBDAS
8.1	SIZE, EXACTLY	2x2
8.2	SEPARATION TO VIA	3
8.3	OVERLAP BY METAL1	1
8.4	SPACE TO POLY OR ACTIVE EDGE	2
8.5	SPACE TO CONTACT	2



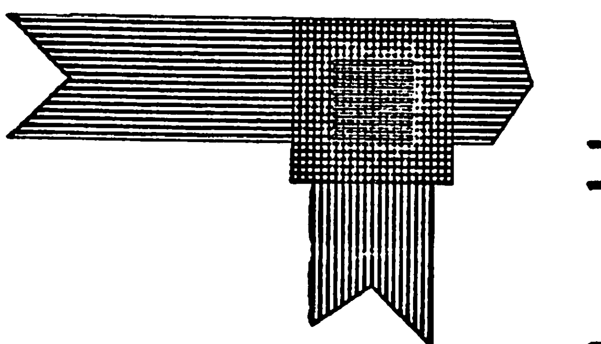
NOTE: OBJECTIVE IS VIA ON A FLAT SURFACE. VIA STACKED OVER CONTACT NOT ALLOWED.

9. METAL2

		LAMBDAS
9.1	WIDTH	3
9.2	SPACE TO METAL2	4
9.3	OVERLAP OF VIA	1



9.2



9.3

10. OVERGLASS

		MICRONS
10.1	BONDING PAD	100x100
10.2	PROBE PAD	75x75
10.3	PAD OVERLAP OF GLASS	6
10.4	PAD SPACE TO UNRELATED CKTRY	30

10.1,10.2

10.4

10.3

NOTE: THERE MUST BE METAL2 UNDER A GLASS CUT OR THE CHIP WILL BE UNBONDABLE