PANDA: A PLA GENERATOR FOR MULTIPLY-FOLDED PLAs

by

Grace H. Mah

Memorandum No. UCB/ERL M84/95

7 April 1984

(cover)

PANDA: A PLA Generator for Multiply-Folded PLAs

by

Grace H. Mah

Memorandum No. UCB/ERL M84/95

7 April 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ABSTRACT

PLAs (Programmable Logic Arrays), are regular structures that are important components in large integrated circuits. The regularity in the structure of PLAs allows automatic design and layout. However, simple automatic layout algorithms usually result in circuits with large area. One method of reducing the area occupied by PLAs is to use multiple folding, a technique implemented at Berkeley in a program called PLEASURE. PLEASURE output is in a symbolic format. PANDA (PLA Analyzer and Design Aid) uses a combination of graphic and procedural methods to lay out the masks for multiply-folded PLAs using the personality matrix produced by the PLEASURE program.

Acknowledgements

I would like to thank Professor A.R. Newton for his guidance, patience, and supportive help through this project. Discussions with Professors D.O. Pederson and A.L. Sangiovanni-Vincentelli have been invaluable in regards to stimulating my interest in the field of computer-aided design and PLA design.

Generous financial and equipment support from Digital Equipment, IBM, Raytheon, the Society of Women Engineers, Tektronix, and the University Women's Club of Pasadena provided the resources for my work.

I would like to express gratitude to G. De Micheli, M. Hofmann, D. Kay, B. Mayo, and S. Pope for their discussions which directly contributed to my greater understanding of PLA generation, module generation, and circuit design.

Numerous people have contributed to my learning and greater appreciation of the computer system at Berkeley. G. Billingsley, C. Cole, J. Kleckner, K. Keller, E. Lock, P. Moore, T. Quarles, and R. Spickelmier have patiently taught me "the system."

Many conversations with my collegues, J. Burns, J. Deutsch, E. Eschen, R. Gyurcsik, B. Lee, C. Lob, R. Liu, F. Ma, K. Mayaram, B. Mayo, I. Ratiu, R. Rudell, R. Saleh, C. Sechen, B. Sheu, B. Valdez, A. Vladimirescu, and J. White have made my working environment and atmosphere a memorable one.

I'd like to especially thank my family for their supportive and constant encouragement. Mr. and Mrs. Gordon and Yu-Chen Mah, Leland, Sophia, Victoria, and Annie have been continuous sources of the moral support and whole-hearted confidence that I occasionally needed. My grandparents, Mr. and Mrs. Gordon S.K. Mah, have likewise provided constant encouragement. Special thanks to Don.

Research sponsored by Hewlett-Pakcard.

Table of Contents

CHAPTER 1

INTRODUCTION

PLAs (Programmable Logic Arrays) [1] are an important part of VLSI circuit design because they can be used to implement combinational logic quickly and with very regular structures. PLAs implement two-level combinational logic, and are often used to implement control, decode, and other glue logic needed between circuit blocks. With the addition of storage elements (e.g. flip-flops), PLAs can be used to implement complex sequential circuits as well.

At UC Berkeley, a number of CAD programs have been developed to allow a convenient implementation of combinational logic in the form of PLAs. Finite state machines, as well as logic equations, can be put into a personality matrix format via various translation programs. Once a personality matrix is created, logic and topological optimization programs can be used to reduce PLA area, and the symbolic representation for the PLA is translated into silicon masks for processing. Some of the most frequently used tools to generate PLAs at UC Berkeley are shown in Figure 1.

Logic equations are used as input to the PLA tools through a program called EQNTOTT [2] which transforms the logic equations into the form of a two-level logic personality matrix. The program PEG [3] reads the description of a finite state machine in a high level language and creates both logic equations and a personality matrix. The personality matrices can then be optimized using logic minimizers, POP [4] or ESPRESSO-IIC [5], which reduce the area of the PLA further using different heuristic algorithms. The function of this

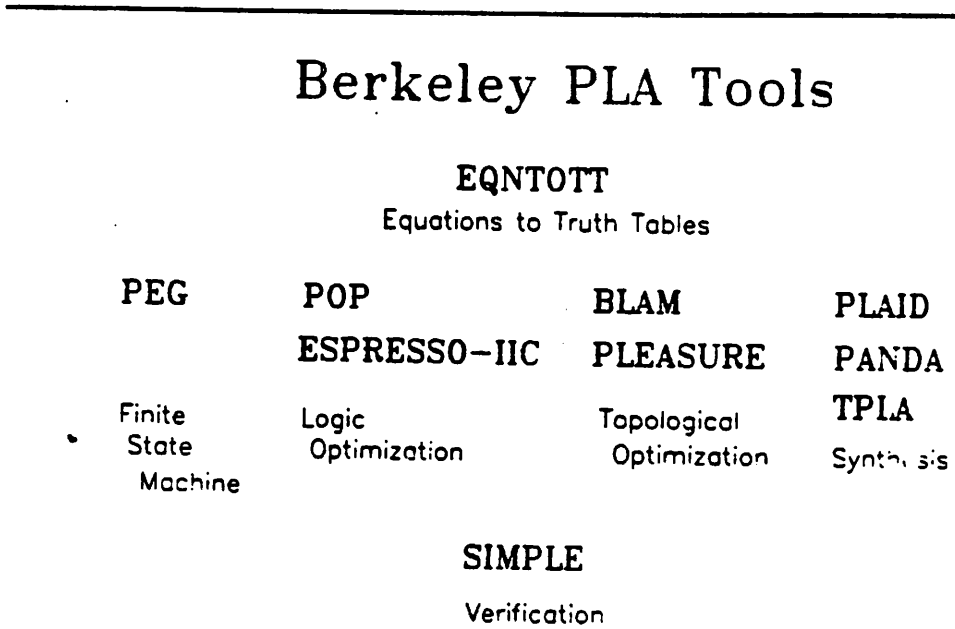


Figure 1. PLA Tools

new PLA can be verified with the program SIMPLE [6] which checks for functional correctness, comparing expanded logic equations with personality matrices via simulation.

Topological minimization is performed with the programs BLAM [7] and PLEASURE [8] which optimize silicon area by folding and splitting PLAs. The final symbolic representation of the PLA is converted into a layout by the programs TPLA [9] for technology-parameterized unfolded PLAs, PLAID for simply-folded nMOS PLAs, and PANDA for multiply-folded PLAs. PANDA generates technology-independent PLAs, and currently implements 3 micron static CMOS PLAs. The synthesis of PLA layout masks by PANDA is the topic of this report.

CHAPTER 2

DESIGN OF PLAS

PLAs are generally described in Sum-of-Products and Product-of-Sums canonical forms [10]. The equations are translated into an AND-OR personality matrix format, and the physical implementation of the personality matrix is generally in a NOR-NOR structure for nMOS or pseudo-nMOS technologies. The symbolic representation of PLAs is described using examples from programs in use at UC Berkeley.

2.1. PLA Creation

Generating a PLA usually begins with logic equations. Figure 2 shows how EQNTOTT translates logic equations into a personality matrix for PLA synthesis. A personality matrix is generally described using *input* and *output* planes. Inputs and outputs form the columns of the input and output planes, respectively. The rows of the PLA are the logic products of some of the inputs. In a finite state machine, the product terms could be different states. For historical reasons, the input and output planes are called the *AND* and *OR* planes. The *AND* plane is the input plane and the *OR* is the output plane. In the *AND* plane, a "1" denotes the true input signal, a "0" denotes a complemented signal, and an "x" denotes a don't-care. In the *OR* plane, connected signals are shown with a "1," a "0" signifies no-connection, and a "x" signifies a don't-care condition.

The AND-OR canonical form of logic equations lends itself quite easily to a NOR-NOR physical implementation. This can be shown from the third equation in the example given in Figure 2:

EQNTOTT

Equations

$F1 = A\bar{B} + C$

$F2 = B\bar{C} + AD + C$

$F3 = A\bar{B}D + A\bar{C} + AB$

Personality Matrix

	A	B	C	D	F1	F2	F3
$A\bar{B}$	1	0	X	X	1	0	1
C	X	X	1	X	1	1	0
$B\bar{C}$	X	1	0	X	0	1	0
AD	1	X	X	1	0	1	0
$A\bar{C}$	1	X	0	X	0	0	1
$A\bar{B}D$	1	0	X	1	0	0	1

Figure 2. Equations to Personality Matrix

$$F3 = A \cdot \bar{B} \cdot D + A \cdot \bar{C} + A \cdot \bar{B}$$

Using DeMorgan's Theorem:

$$= \overline{\overline{A \cdot \bar{B} \cdot D + A \cdot \bar{C} + A \cdot \bar{B}}}$$

$$= \overline{\overline{A \cdot \bar{B} \cdot D} \cdot \overline{A \cdot \bar{C}} \cdot \overline{A \cdot \bar{B}}}$$

This could be directly mapped into a NAND-NAND implementation.

$$F3 = \overline{\overline{\bar{A} + B + \bar{D}} \cdot \overline{\bar{A} + C} \cdot \overline{\bar{A} + B}}$$

$$= \overline{\bar{A} + B + \bar{D}} + \overline{\bar{A} + C} + \overline{\bar{A} + B}$$

Equations in NOR-NOR notation:

$$\overline{F3} = \overline{\bar{A} + B + \bar{D}} + \overline{\bar{A} + C} + \overline{\bar{A} + B}$$

Note that the direct implementation of the NOR-NOR equations require inverted inputs and inversion of the output.

From the personality matrix produced from the example in Figure 2, a CMOS static PLA can be generated easily with the program TPLA as is shown in Figure 3. In the physical implementation of the personality matrix, input

columns are expanded to two columns, one for the true signal and another for the complemented signal.

2.2. PLA Optimization

In general, EQNTOTT's generation of a personality matrix from logic equations is performed without considering the size of the resulting personality matrix, although EQNTOTT does perform some minimization with respect to unassigned states. Since PLAs tend to be very sparse, minimization of the area of a PLA can be implemented by both logic and topological optimization techniques.

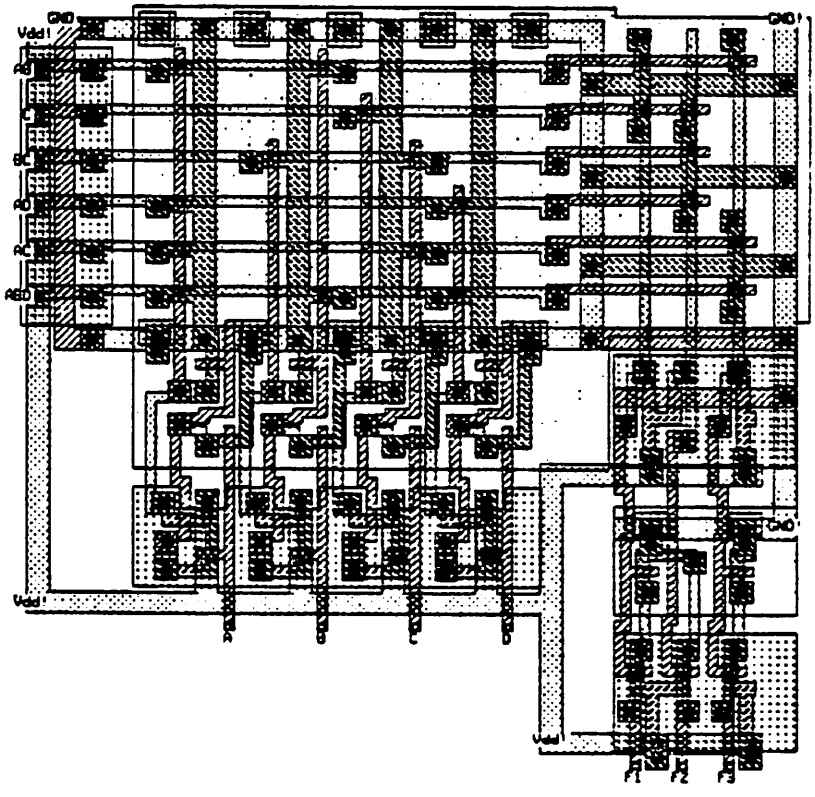


Figure 3. TPLA Implementation

2.3. Logic Optimization

Logic optimization is accomplished with two programs written at UC Berkeley, POP and ESPRESSO-IIC. Both programs use different heuristics to manipulate product terms in the personality matrix, changing and combining them to give a more compact logic description of the PLA. By making the PLA more compact, fewer product terms are needed to represent the logic equations, reducing the number of rows in a PLA. The height of the PLA personality matrix is thus diminished, while maintaining the same logic function.

The output personality matrix from POP and ESPRESSO-IIC is logically equivalent to the original logic equations. The topology of the PLA generated from the personality matrix is the general form: inputs and outputs come into and out of the PLA on only one side as was seen in Figure 3.

2.3.1. Topological Optimization by Simple Folding

Topological optimization involves folding and splitting the rows and columns of a PLA [11]. By rearranging the logic rows and columns of a PLA, physical rows and columns can be shared. Simple folding results in one logic input (or output) column sharing its physical column with another logic input (or output), the total width of the PLA thus being reduced by one column. This is illustrated in Figure 4.

One input (or output) now will enter the top of the PLA and one will enter the bottom of the PLA in the same physical column. In a similar fashion, two rows can be shared such that two logical rows share the same physical row. This results in AND-OR-AND structures with inputs coming into the PLA in two planes, and the outputs coming out in a middle plane. OR-AND-OR structures similarly have inputs coming into the PLA in a middle plane, and coming out of the PLA through two outer planes.

Simple Folding

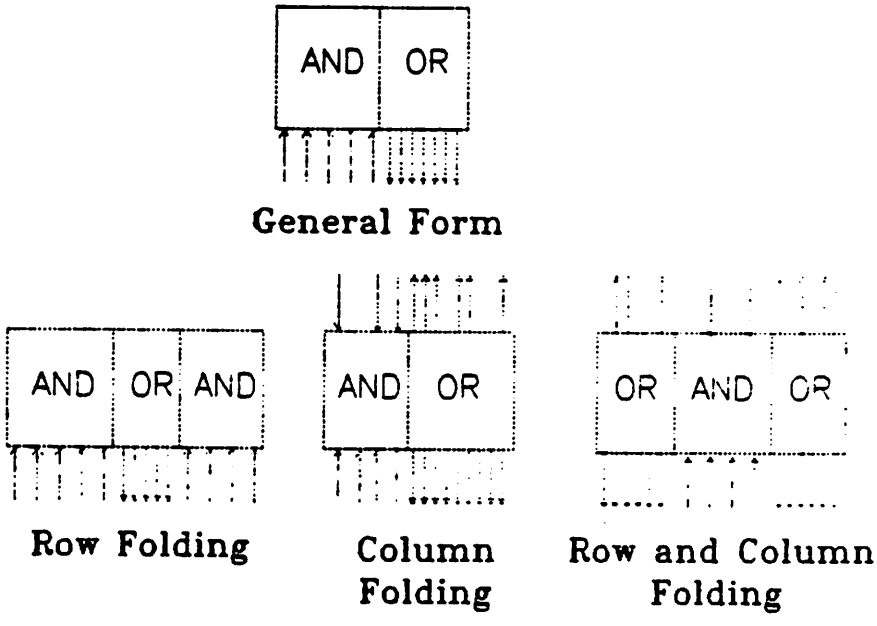


Figure 4. Simple Folding

The splitting and folding processes can be implemented by hand or they can be implemented automatically by a computer program. Unfortunately, for other than very simple PLAs, hand optimization is very time consuming and generally results in layouts which are far from optimal.

Automatic simple folding and splitting was first implemented at Berkeley in the program BLAM, which uses a branch-and-bound algorithm to find an optimum folding and splitting of columns and rows which results in the greatest reduction of area of the PLA under analysis. Using only simple row and column folding, the minimum area possible is 25% of the total original area, a maximum reduction by 75% of the original area. This is because if all columns and rows could be split and folded, the number of resulting rows and columns would each be half of the original number of rows and columns.

When both columns and rows are simply-folded, inputs and outputs will enter and leave the PLA only on two opposite sides of the PLA. Figure 5 shows mask layers for a CMOS PLA that has been simply folded in both columns and rows. These PLAs were generated by the program PLAID, an nMOS PLA generator for simply-folded PLAs.

2.3.2. Topological Optimization by Multiple Folding

Multiple folding takes simple folding one step further. Instead of sharing only two logic columns or rows in a simple physical column or row, multiple folding allows for many columns and rows of the PLA personality to be shared in one physical column or row. That is, one physical column can now

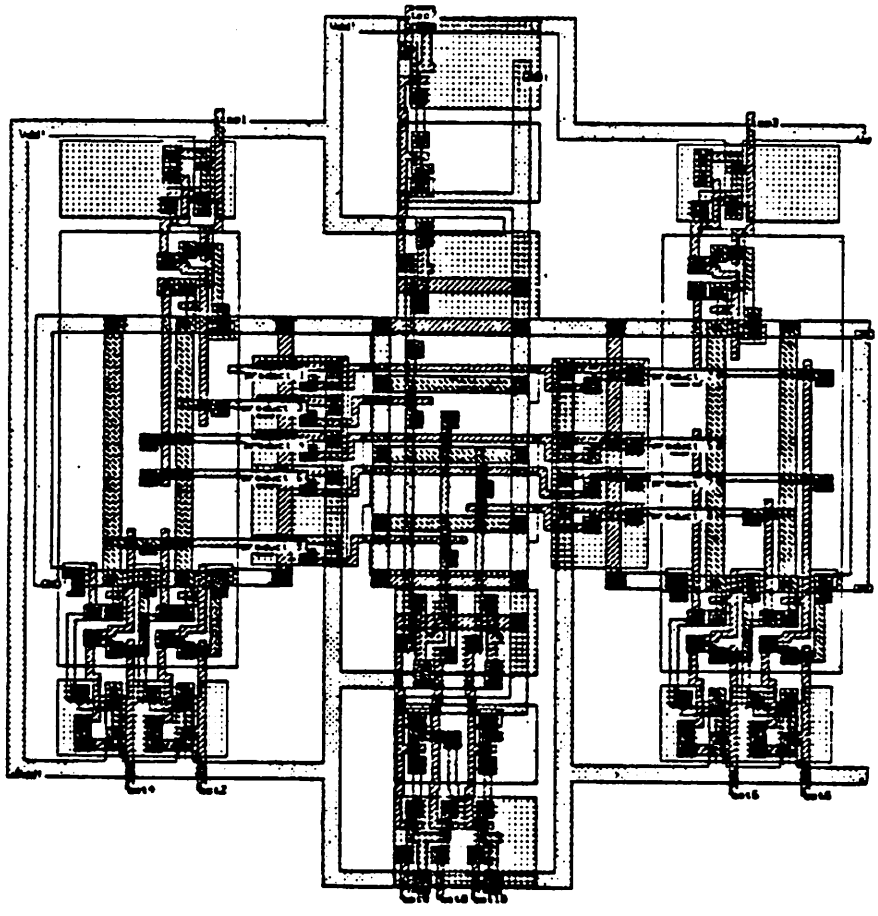


Figure 5. PANDA Simple Row and Column Folding

implement three or more logical columns. To implement this, an input (or output) can enter a PLA from the top, bottom, *or* sides of the PLA. Multiple folding thus permits routing of inputs and outputs on all four sides of a PLA.

With multiple folding, the maximum amount of area reduction for a PLA is now limited by only the sparsity of the original PLA personality and the structures like AND-OR-AND-OR-AND possible. Figure 6 shows some possible topologies of multiply-folded PLAs.

The program PLEASURE uses efficient heuristic techniques to find a compact multiple folding for a PLA. PLEASURE can be constrained in the ordering of inputs and outputs such that certain inputs and outputs remain in a

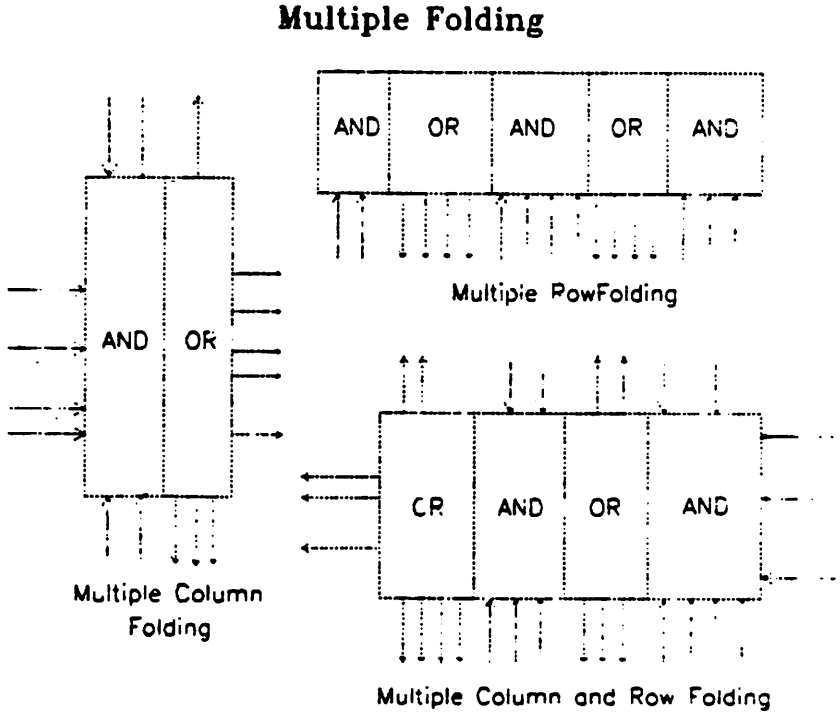


Figure 6. Multiple Folding

specific order. (These constraints are usually dictated by surrounding circuitry.) Inputs and outputs can also have window constraints limiting the general area an input or output can enter or leave the PLA. For instance, the user may specify that certain inputs must come into the PLA from the top, certain outputs from the left side, and so on.

2.4. The Symbolic Representation of Folded PLAs

The symbolic personality matrix required for folded PLAs contains more symbols than "1," "0," and "x", introduced earlier. The format for personality matrices expands the true and complemented signal columns in the AND plane, and shows input and output buffers along all sides of the PLA. Figure 7 shows the new symbols that represent folded PLAs. Appendix A contains user documentation for interpreting the symbols used for folding and splitting in

Symbols for *AND* Plane		
Contact Signal	No Contact	Explanation
1	-	Normal contact, no splits or folds
!	_	Split below
;	,	Fold to the right
:	.	Split below and fold to the right

Symbols for *OR* Plane		
Contact Signal	No Contact	Explanation
I	~	Normal contact, no splits or folds
i	=	Split below
|	'	Fold to the right
j	"	Split below and fold to the right

Additional Symbols	
Symbol	Explanation
*	Input buffer
+	Output buffer
X	No buffer
c	Contact within AND or OR plane
>, <	Routing lines to contacts for multiple folds

Figure 7. Symbolic Format for Folded PLAs

PLA.

The new personality matrix "looks" very much like the eventual implementation of the PLA. Input and output buffers are along the sides of the PLA where they will be placed in the physical implementation. The *X* character is a place holder representing *no-buffer*.

Multiple column folding requires input and output signals to come into the PLA from the sides and special *contact* symbols in the core of the PLA are shown as *c*'s with the contact row routing to the contact indicated with ">" or "<" symbols.

CHAPTER 3

USING PANDA

PANDA (PLA Analyzer and Design Aid) is a PLA generator for multiply-folded PLAs. Built on the TPACK[12] package of tiling routines and using the CAESAR [13] or KIC[14] graphics editor, PANDA can use a simple template of a multiply-folded PLA to compile PLEASURE's symbolic output into layout masks. With multiply-folded PLA templates designed in different technologies, PANDA can generate PLAs of different technologies easily, and more efficiently than by hand layout.

3.1. Generating a PLA with PANDA

PANDA was written under the UNIX[1] operating system. To make a PLA (under a UNIX environment) with PANDA, type:

panda *input_file*

The *input_file* is a personality matrix with control statements. The personality matrix symbols follow the previously described format in Chapter 2. The control statements are discussed below. A new file in CAESAR format is then created by PANDA with the name *input_file.ca* which will contain the PLA. For CIF (Caltech Intermediate Format [15]) format, the **-c** option of PANDA will generate a *input_file.cif* file. Numerous other options exist for setting scale factors (e.g. λ), stretching power and ground lines, for inserting extra ground lines where necessary to maintain acceptable performance, and for providing feedback during the construction of the PLA. These are all explained in the

[1]UNIX is a Trademark of Bell Laboratories

manual pages, included as Appendix B.

The input format for PANDA is compatible with the *.machine* output of PLEASURE. Including the *.machine* control statement in the PLEASURE input file results in the proper output format for PANDA. This format is described in Chapter 2 and Appendix A.

A typical input file to PANDA is shown in Figure 8. The first lines of the input file are control statements.

```
.and   2 4 3 1
.row   8
  * X     ++X+    X * X     +
X1---    ~i~I    --1---    ~X
X----    I~~i    ,;1-1-    iX
X_!--    ~|~I    i-_!--    ~X
X1--1    |~~I    -----1    ~X
X--1-    i~~~    ---1--    ~X
*c                         ~X
X>c                        ~X
X!_1-    ~|~I    1-;,--    ~X
X-1--    I~I~    1-----    IX
X---1    ~~~~    --1---    IX
  * *    ++++    * * *     +
.end
```

Figure 8. PANDA Input

Each control statement must begin with a "." (dot or period). The following control statements are understood by PANDA:

.[and | or] *num1 num2* [*num3* ...]

This line describes the structure of the PLA, with the **and** or **or** specifying that the plane on the left is an *AND* or *OR* plane. The numbers following the first plane designator are the numbers of inputs/outputs (depending on which plane is first) in each successive plane. For instance, the control statement: " *.or 9 3 4 5* " means that this PLA has an OR-AND-OR-AND structure with 9 outputs in the first OR plane, 3 inputs in the next AND plane, 4 outputs in the next OR

plane, and 5 inputs in the last AND plane. Note that at least two numbers must follow the first plane designator since a PLA must have at least one AND and one OR plane.

.row [*number of rows*]

This line describes the height of the input personality matrix.

.top [*label1 label2 label3*...]

.bottom [*label1 label2 label3*...]

.left [*label1 label2 label3*...]

.right [*label1 label2 label3*...]

These control statements list the labels for inputs and outputs along the top, bottom, left, and right of the PLA. Note that these labels are not designated as being either inputs or outputs. The AND-OR-AND... structure of the PLA should be determined by the first **.[and | or]** control statement described.

.end

This line signals the end of the input file.

For the sample PLA in Figure 8, Figure 9 shows the output CIF file generated by PANDA.

3.2. Creating a PANDA Template

PANDA requires a template when generating its PLAs. The template contains rectangles filled with mask information. These rectangles are labeled with names that can be called by TPACK routines to be aligned according to a certain semi-regular structure which PANDA will define.

The first step in making a new PANDA template is to design a sample multiply-folded PLA in the new technology or new style. This PLA should include at least one example of each possible combination of template cells. With this template, PANDA will have a correct example to generate its own larger PLAs. The template should contain all possible features of a multiply-

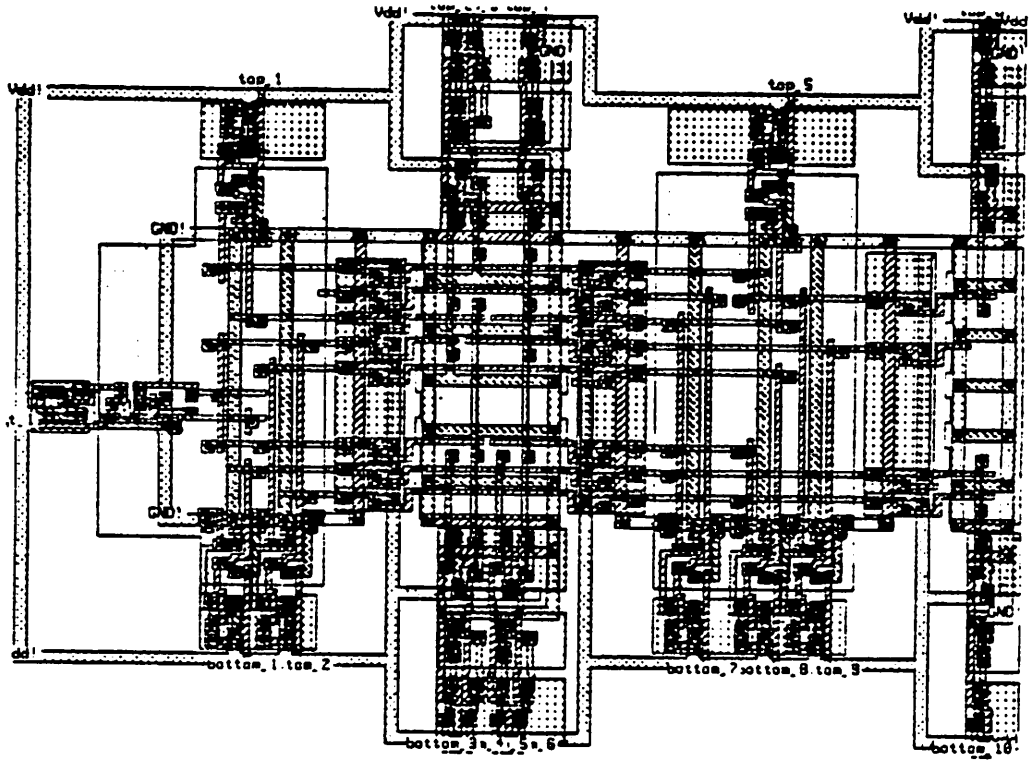


Figure 9. CIF File for Example PLA

folded PLA: all orientations of input and output buffers, transistors connected to true and complemented signals in the AND plane, permutations of the presence and absence of interconnect to minimize capacitance within the PLA core and more. The template for a multiply-folded static 3-micron p-well CMOS PLA is shown in Figure 10. The tile boundaries and names have been removed to show the basic fundamental features of the template.

Once a template has been designed for a sample multiply-folded PLA, rectangular tiles can be defined for each cell in the template with either the CAESAR or KIC2 graphics editors. PANDA requires specific tiles to be defined in the template. They can be divided into eleven groups of tiles:

1) Tiles in the core of the AND plane
2) Tiles in the core of the OR plane
3) Tiles along the outer left and right sides of the AND plane
4) Tiles along the outer left and right sides of the OR plane

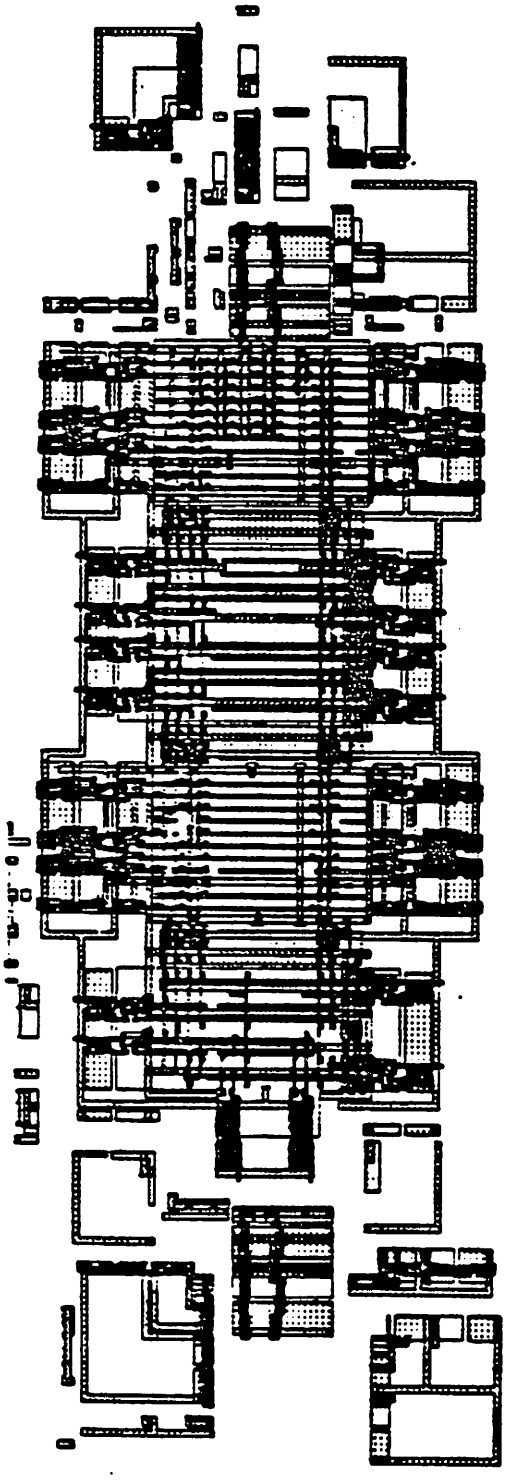


Figure 10. PANDA Template

5) Tiles along the top and bottom of the AND plane
6) Tiles along the top and bottom of the OR plane
7) Tiles between inner AND and OR planes
8) Horizontal spacing tiles in the AND plane
9) Vertical spacing tiles in both planes
10) Horizontal ground tiles
11) Vertical ground tiles

These are individually described in Appendix C, the PANDA manual for tile definition.

Tiles along the outer sides of the PLA (left, right, top, and bottom) may contain labels which are linear. Linear labels are lines with a name attached. Tiles with linear labels can be stretched along the label. That is, the tile would be figuratively "cut" along the line label and then the two "pieces" would be stretched apart by a designated amount. This feature is useful for stretching power and ground lines to prevent metal migration.

PANDA builds PLAs by stacking a row of tiles together along a line, and then aligning this row of tiles to the previous row. Beginning with the row of tiles along top, PANDA stacks tiles together, aligning their bottom edges. The top row of tiles consists of left and right corner tiles, and top input and output buffer tiles for column folded PLAs. When a buffer is not required at the top of a column, a connecting tile is placed to connect power and ground along the top of the PLA.

The next row of tiles is the first product row. This row is made up of a left edge tile, core tiles for the AND and OR planes, and a right edge tile, all aligned along their bottom edges. This product row is stacked below the "top" row. Successive rows of tiles for each product row are then similarly made and stacked under the preceding product row. The last row of tiles along the bottom of the PLA is made aligning the top edges of all the tiles. Then this last row is stacked below the previous core row and the PLA is com-

plete.

Rows in the AND plane are made up of pairs of *right* and *left* columns that implement *true* and *complement* signal connections. The OR plane(s) of a PLA are made up of alternating *up* and *down* rows. Additional *contact* rows may also appear in a PLA when multiple column folding is required.

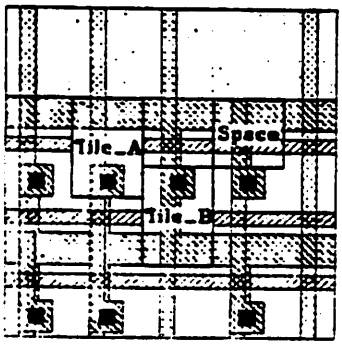
When defining new tiles for PANDA, careful notice should be taken of the alignment requirements for the tiles. Different tiles are aligned according to certain corners, depending on the location of the tile. Templates designed in new technologies must meet the specific alignment requirements described in Appendix C.

3.3. Alignment Example for PANDA Tiles

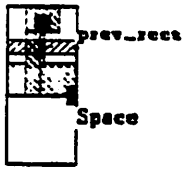
PANDA uses the TPACK package of tiling subroutines for generating PLAs. The TPACK subroutines manipulate rectangles and tiles defined in the PANDA template procedurally. The TPACK routines can add, subtract, label, and align corners of tiles in the template to other tiles. Other TPACK routines are capable of stretching tiles, adding and subtracting absolute points and rectangles, and creating and deleting tiles. Appendix D contains the manual pages for the TPACK program.

An example of PANDA's code constructing a set of tiles in the OR plane is shown in Figure 11. The rectangles that are drawn and the tiles that are placed by this code are shown next to the code defining their placement. Dark squares in the corners of the tiles show points of alignment. The subroutines **TPdisp** and **TPpaint** displace and draw tiles which are passed as parameters. The **TPdisp** subroutine uses only the boundary of a tile to control overlapping of tiles and placement of tiles to be drawn. **TPpaint** draws mask information contained within a tile.

Template Tiles

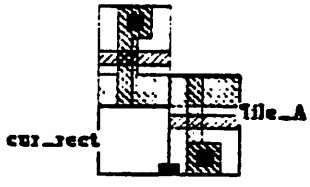



```
RECTANGLE cur_rect, prev_rect;
  /* drawn rectangles */
TILE Tile_A, Tile_B, Space;
  /* tiles to be drawn */
```



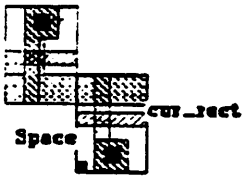

```
cur_rect = TPdisp_tile(Space,
  align(rLR(prev_rect), tUR(Space)))

  /* aligns lower right of prev_rect
       to upper right of Space */
```



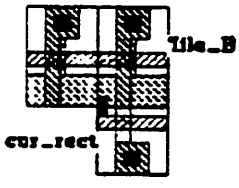

```
cur_rect = TPpaint_tile(Tile_A, out_tile,
  align(rLR(cur_rect), tLL(Tile_A)));

  /* aligns lower right of cur_rect
       to lower left of Tile_A */
```




```
cur_rect = TPdisp_tile(Space,
  align(rLL(cur_rect), tLL(Space)))

  /* aligns lower left of cur_rect
       to lower left of Space */
```




```
cur_rect = TPpaint_tile(Tile_B, out_tile,
  align(rUL(cur_rect), tLL(Tile_B)));

  /* aligns upper left of cur_rect
       to lower left of Tile_B */
```

Figure 11. Example of PANDA Code

CHAPTER 4

How PANDA Works

PANDA tiles a PLA according to an array that PANDA generates from a vertical and horizontal pass over PLEASURE's output personality matrix. PANDA's array is then directly translated into the physical PLA. The size of PANDA's array is determined from control statements that specify the number of

Symbols for *AND* Plane		
Symbol in Left Column	Symbol in Right Column	Explanation
1	I	Normal contact
0	O	No contact, pass column signal
!	o	Split below
;	@	Fold to the right
:	#	Split below and fold to the right
e	f	Don't pass row signal (optimizing tiles)
.	,	Don't pass column signal (optimizing tiles)
z	y	Don't pass row and column (optimizing tiles)

Symbols for *OR* Plane	
Symbol	Explanation
J	Normal contact
Q	No contact, pass column signal
i	Split below
|	Fold to the right
j	Split below and fold to the right
E	Don't pass row signal (optimizing tiles)
~	Don't pass column signal (optimizing tiles)
Z	Don't pass row and column signal (optimizing tiles)

Additional Symbols	
Symbol	Explanation
*	Input buffer
+	Output buffer
X	No buffer
c	Contact within AND or OR plane
> <	Routing lines for contacts

Figure 12. PANDA Array Symbols

columns and rows for the core of the PLA. Because of the constraints imposed by the output of PLEASURE, the number of contact rows for multiple folding can also be calculated. When PANDA processes the input array, a new set of symbols are used, internal to PANDA. These symbols are described in Figure 12.

In PANDA's vertical pass over the input personality matrix, the personality matrix is filled with symbols that reflect the shortening of vertical lines in both the *AND* and *OR* planes to minimize capacitance. This first minimization is from the top, going down in each column. Figure 13 shows an example of a PLEASURE file on the left and PANDA's first array on the right.

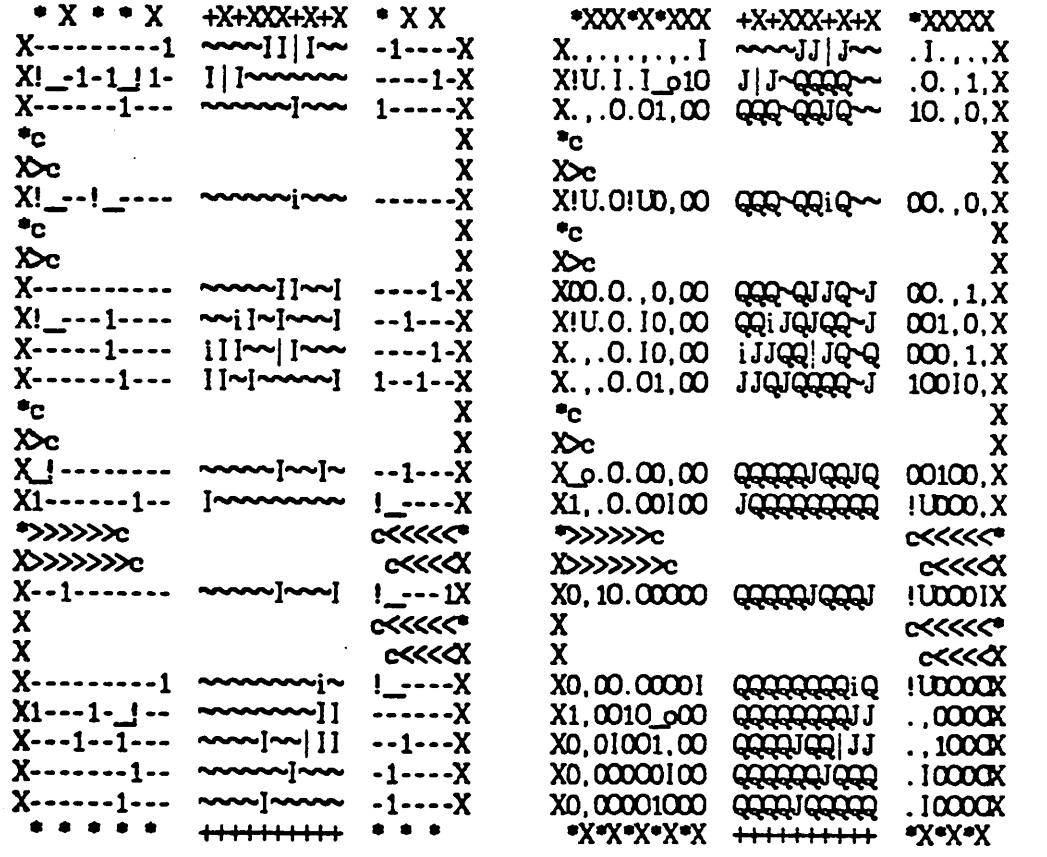


Figure 13. PLEASURE File and PANDA's First Pass

The resulting personality matrix is then processed a second time horizontally from the left to the right sides (and vice versa), to shorten horizontal lines thus reducing capacitances in the PLA in the horizontal direction. This is shown in Figure 14 for the same example.

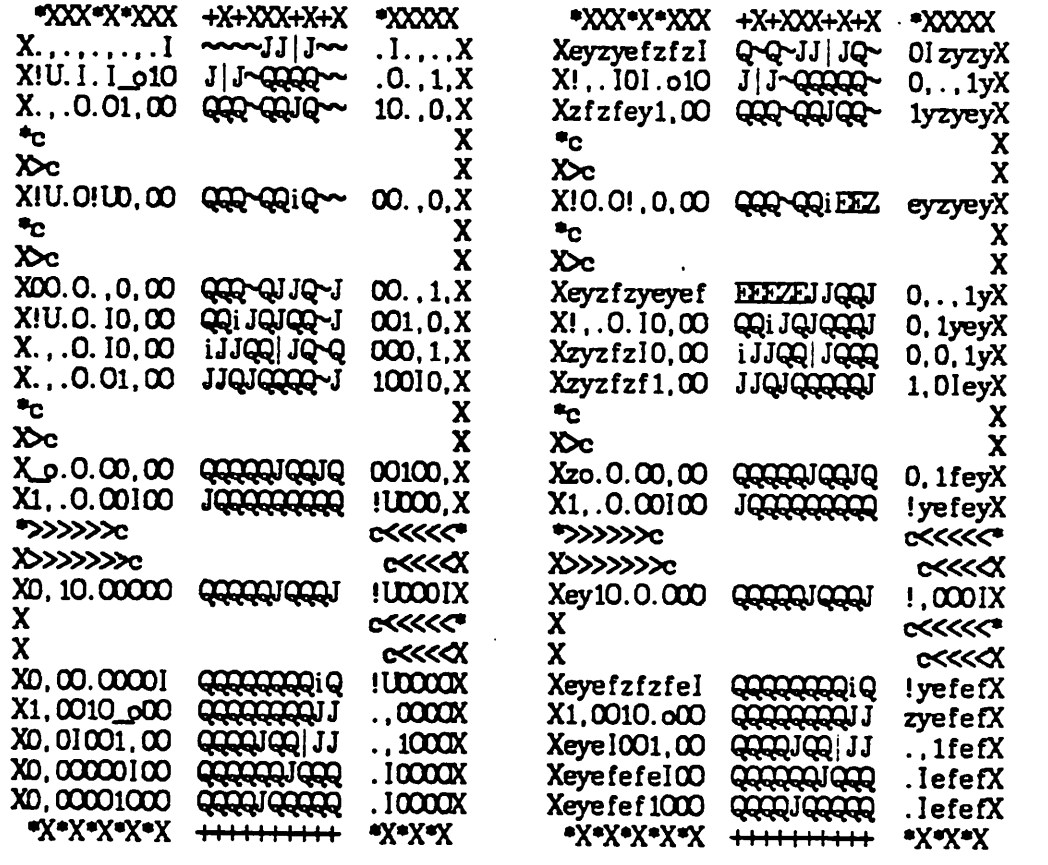


Figure 14. PANDA's First and Second Passes

The final personality matrix, after the horizontal pass, is then translated to layout geometry with each symbol representing a specific tile. Edge tiles and connecting middle tiles between planes are added for each row and the

final output from PANDA for this example is shown in Figure 15.

Extra spacing between rows and columns is occasionally required for anomalies during folding. These anomalies can exist in a row within the AND plane of a personality matrix when two logic rows are folded between two transistors that would normally share a contact. In the PLA in Figure 15, two logic rows were combined into one physical row, but the logic fold between these rows required the duplication of a contact plus some spacing between the contacts. This additional space is provided automatically by PANDA. This type of anomaly can also occur in the OR plane along columns that are folded between adjacent transistors that would usually share a contact. This can also be seen in the OR plane in the PLA of Figure 15 where PANDA automatically puts in the extra row of spacing.

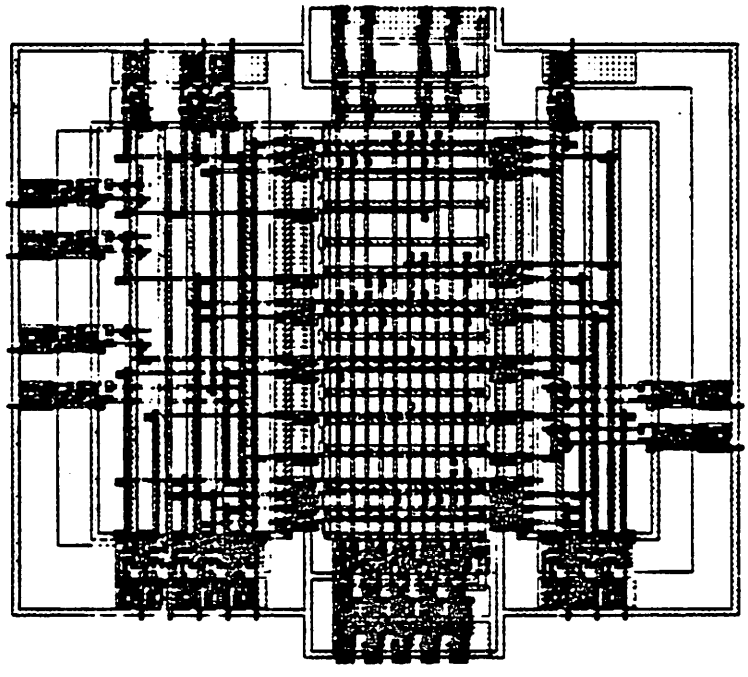


Figure 15. PANDA Output

Another situation in which extra spacing is required is in multiply-folding the columns of the output plane. The output buffers from the side of the PLA must contact columns in the core of the *OR* plane by creating a contact row. This contact row extends from the side output buffer into the PLA until it makes contact with the appropriate column. This *output contact* connects the output buffer with the folded column within the core of the *OR* plane. A product row below this contact row might have a transistor directly below the *output contact* in the contact row. Without the extra spacing, the product row transistor would be connected to the output contact above. This spacing anomaly is shown in Figure 16.

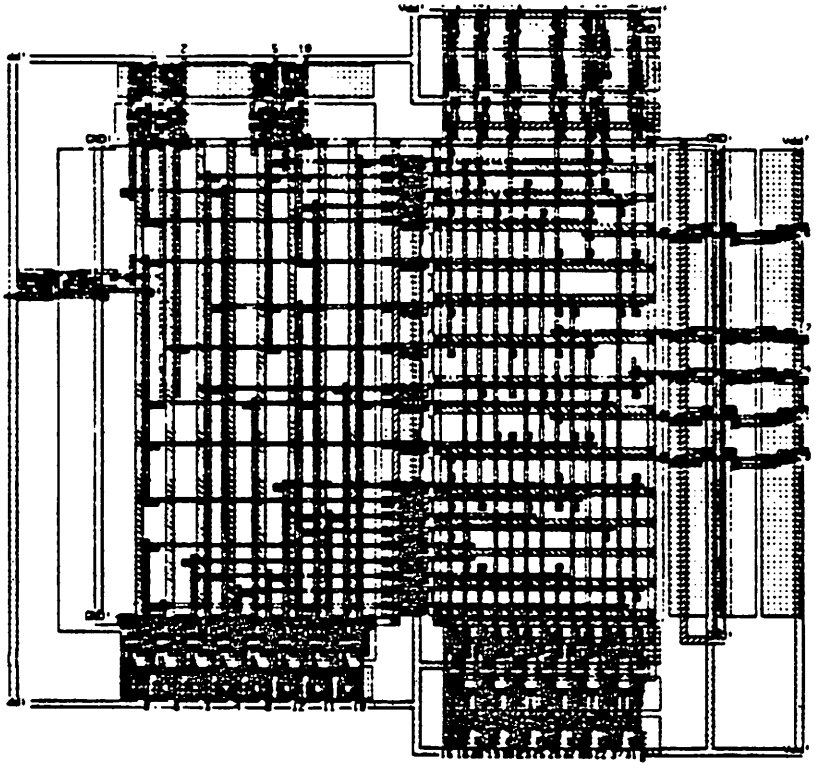


Figure 16. Output Contact Spacing Anomaly

CHAPTER 5

PANDA's Performance

PANDA automatically computes the needs of extra ground lines within the core of the PLA to keep voltage levels at sufficent logic levels. The stretching of power and ground lines to provide enough current capacity to prevent metal migration is also automatically calculated by PANDA. The methods for calculating these electrical constraints are similar to those used by the program MKPLA [16] and are based on the pa-CS3 PANDA template. A large CMOS PLA was built by PANDA and tested for speed and power consumption.

5.1. Extra Ground Lines

For large PLAs, extra ground lines are needed to insure proper switching voltage levels. Diffusion grounds that run the length and width of the AND and OR planes, respectively, can have parasitic resistances that would result in large voltage drops along the "grounds" and cause output voltage levels to fall below those required for proper logic switching.

P-channel transistors pull up product lines along the diffused ground and to first order can be modelled as current sources, of value i_p, along a chain of resistors. Since the diffused ground is symmetric, only half of the chain needs to be analyzed. Assume, for simplicity, that there are an even number of current sources, n. Half of the chain has $\frac{n}{2}$ current sources and $\frac{n}{2}$ resistors, r_i, with a solid ground at one end of the chain, as shown in Figure 17. The voltage drop across any resistor r is given by the total current through it:

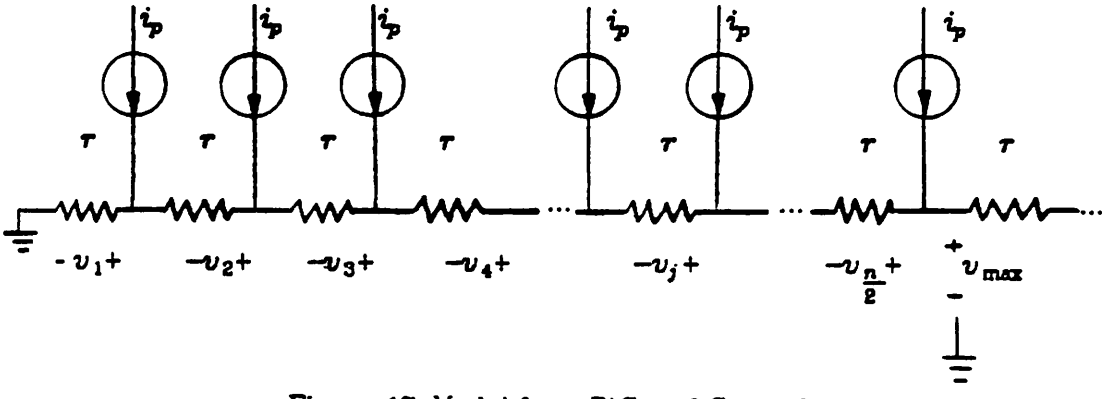


Figure 17. Model for a Diffused Ground

$$v_j = i_p \times \tau \times (\frac{n}{2} + j - 1).$$

The voltage drop across any resistor τ is given by the total current through it:

$$v_j = i_p \times \tau \times (\frac{n}{2} + j - 1).$$

Hence, the maximum voltage drop to the center of the line, v_{max}, is given by:

$$v_{max} = i_p \times \tau \times \sum_{j=1}^{\frac{n}{2}} (\frac{n}{2} + j - 1) = i_p \times \tau \times (\frac{n^2}{8} + \frac{n}{4})$$

A 6μm x 4μm p-channel transistor with its gate tied to ground generates .07mA, and the resistence τ of the diffusion between product rows is about 60Ω (assuming the diffusion has a resistivity of $\frac{30\Omega}{square}$). This results in $i_p \times \tau = (.07mA)(60\Omega) = 4.2mV$. Since the threshold voltage of an enhancement transistor is typically between .7V and 1V, a conservative maximum allowed value of v_{max} would be .3V, and the calculation becomes:

$$\frac{n^2}{8} + \frac{n}{4} \leq \frac{.3V}{.0042V}$$

$$n \leq 23$$

Thus, one extra ground is needed for every 23 product terms. However, the maximum current-carrying capability for metal at $\lambda = 2.5\mu m$ is approximately 1mA $/\mu m$, or 7mA for a nominal 7μm ground line. Therefore:

$$i_{max} = i_p \times (\frac{n^2}{8} + \frac{n}{4}) \leq 7mA$$

Therefore,

$$n^2 + 2n - 800 \leq 0$$

$$n = 27$$

In this case, the voltage drop limit determines the maximum number of product terms permitted between ground lines. Once the number of extra grounds is calculated, PANDA puts them evenly throughout the PLA.

5.2. Stretching Ground and Power Lines

Assuming V_{dd} stays at 5V, power and ground lines must be stretched for large PLAs to prevent metal migration. The width of metal will become:

$$\frac{totalcurrent}{1mA/\mu m}.$$

The total current is equal to the current sourced by a p-channel pull up, .07mA, times the number of product terms (pterms). Thus, for a large PLA with 150 pterms, the width of the final power and ground lines would be 10.5μm.

5.3. Speed and Power Dissipation

A PLA with 139 product terms and 33 inputs and outputs was constructed using PANDA. The worst-case propagation time as predicted by SPICE2[17] was 55ns for the critical path which extended the whole height of the PLA. The resistance of the poly was assumed to be $\frac{40\Omega}{square}$, and the

resistance of metal $\frac{.03\Omega}{square}$. The power dissipated by one p-channel pullup transistor was .34 μwatts, using the SPICE2 program. These parameters from the PLA are shown in Figure 18.

Parameter	Value	Units
Number of Inputs and Outputs	25	
Number of Product Terms	139	
Total Power Dissipation	.055	watts
Propagation Delay	55	ns

Figure 18. Parameters for a Large PLA

CHAPTER 6

Future Work and Conclusion

PANDA satisfies the requirements of generating any type of nonfolded, simply-folded and multiply-folded PLA that can be described by the program PLEASURE. Further work could be done to improve its efficiency and performance, as well as to provide a more effective interface with other CAD tools and protocols. More complex structures for PLAs with greater flexibility with respect to shape and form should also be representable with PANDA.

6.1. Technology Independence

Although PANDA is template-driven, some of the circuit design parameters are based upon the 3-micron CMOS process for which the **pa-CS3.tp** template was designed. The parameters within PANDA which are involved are the placement of extra ground lines within the core of the PLA, and the stretching of power and ground lines in the PLA. PLAs generated with other templates may have these defaults overridden using the -G and -S options; however, a better means of overcoming this problem is to have an entry (in a UNIX environment) in the ~/.cadrc file of the user specifying which technology is to be used. There, the user could specify the process parameters or requirements for PANDA to use when calculating the number of extra ground lines or amount of stretching for power and ground lines.

Another assumption PANDA makes is with regard to having only one layer of metal in the process. As further advances are made in using two layers of interconnect in VLSI, PANDA would have to be changed to have different spacing and overlapping rules for contact rows.

6.2. Folding Contact Rows

PANDA reads a personality matrix from PLEASURE and makes some optimizations based upon shortening long signal lines to reduce capacitances. The topology of the layout that PANDA outputs is the same as the PLEASURE output personality matrix. Contacts from the left and right sides of the PLA take up an entire horizontal row (two rows if the side contact is for an input buffer with signal and complement lines).

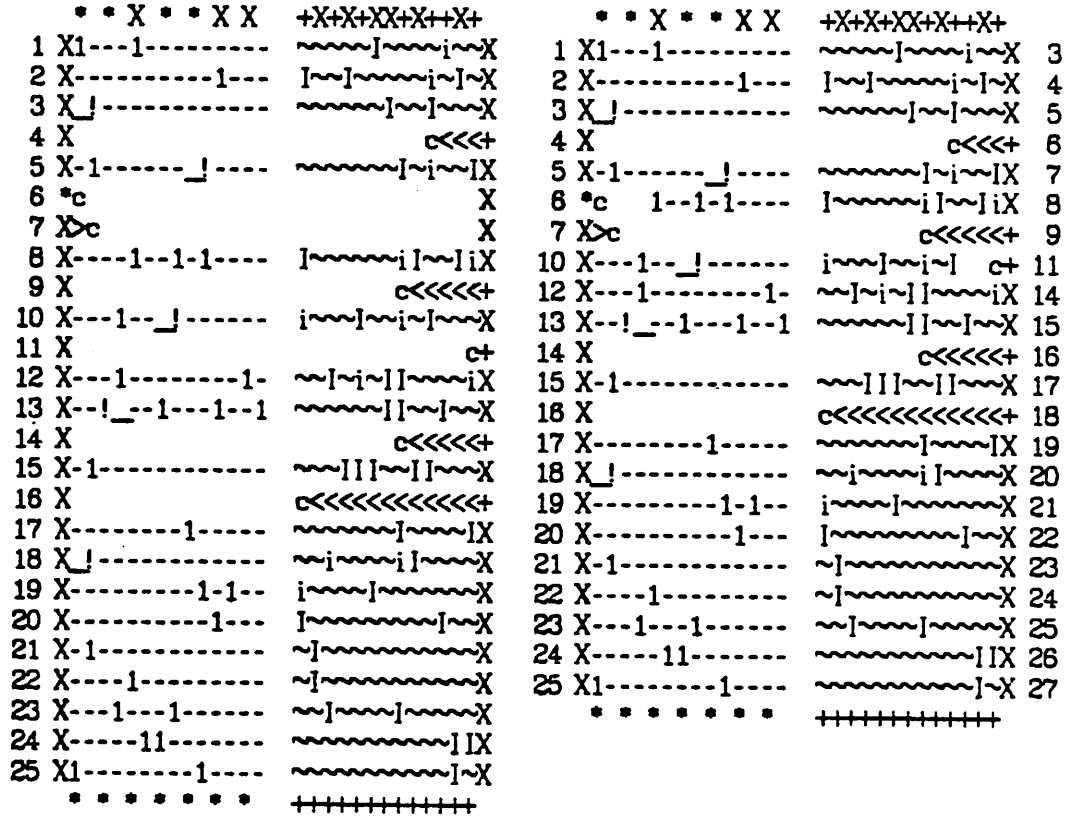


Figure 19. Folding Contact Rows, Before and After

In some PLAs, these contact rows could be used to compact columns. This can be more easily seen in Figure 19 where the personality matrix of a PLA is

shown before and after this type of compaction has been performed. After the compaction, rows 6, 7, and 10 are shared with rows 8, 9, and 11, respectively. The physical rows which were previously only used for side contacts have now also been folded. PLEASURE and PANDA do not produce PLAs which have such folded contact rows, but such a feature would be useful for further compacting folded PLAs.

In processes which use two layers of metal, folding contact rows would not be necessary since contacts and buffers from the side could be brought into the core of the PLA from above with the second layer of metal. With a second layer of metal, special spacing tiles will have to be put into PANDA for contact rows. Currently, PANDA treats contact rows like product rows with regards to spacing.

6.3. Jogs in Contact and Product Rows

Multiply-folded PLAs could be further compacted if contact and product rows could be jogged. The current PLEASURE and PANDA topology does not contain jogs in rows, but jogs would use up unused empty space in rows and columns.

The greater flexibility would require more complex heuristics in the folding and splitting of PLAs. The physical synthesis of the PLA would also need smarter algorithms to optimally place the jogs within the guidelines of PLEASURE's output personality matrix. Jogs in a personality matrix would require new symbols, perhaps V and ^ for signals that went up or down within a PLA that were connected to side contacts.

8.4. Extending Input and Output Signals

The current implementation of PLEASURE and PANDA do not allow input and output lines to extend past the cores of AND and OR planes, respectively. If output signal lines could run through an external AND plane, for instance, a more flexible and complex PLA could be synthesized. Extended output signals and jogs are shown in Figure 20.

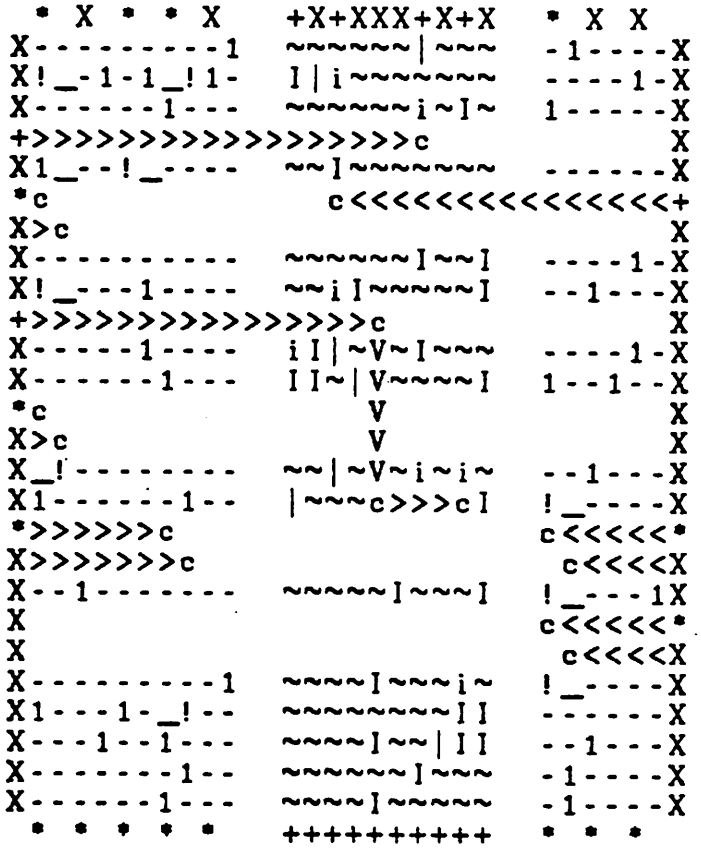


Figure 20. Jogs and Signal Extension

6.5. Optimization Filter

The optimizing routines in PANDA which carry out the shortening of signal lines to reduce capacitances could be separated from the main program and be put into a filter program. This would be convenient if a PLA did not need to be optimized. A direct construction of a PLA from the personality matrix could be done without the optimizing filter. PANDA's execution time would thus be improved.

6.6. Conclusion

PANDA, a layout generator for multiply-folded PLAs, was written in ~5000 lines of C code. The program is template driven, requiring ~100 tiles to fully characterize a multiply-folded PLA. For optimized PLAs with shortened signal lines, 54 optional tiles can be defined. The **pa-CS3.tp** template contains ~5000 lines of graphical data. PANDA can generate non-folded, simply-folded, and multiply-folded PLAs based on a standard symbolic format for a personality matrix.

APPENDIX A

PLA Input Format

This appendix contains the input format for PLA tools.

NAME

pla – Format for physical description of Programmable Logic Arrays.

SYNOPSIS

pla

DESCRIPTION

This format is used by programs which manipulate plas to describe their physical implementation. Lines beginning with a '#' are comments and are ignored. Lines beginning with a '.' contain control information about the PLA. Currently, the control information is given in the following order:

.i <number of inputs>
.o <number of outputs>
.p <number of product terms (pterms)>

What follows is a description of the AND and OR planes of the PLA with one line per product term. Connections in the AND plane are represented with a '1' for connection to the non-inverted input line and a '0' for connection to the inverted input line. No connection to an input line is indicated with 'x', 'X' or '-', with '-' being preferred. Connections in the OR plane are indicated by a '1'. No connection is indicated with 'x', 'X', '-', or '0', with '-' being preferred. Spaces or tabs may be used freely and are ignored.

The end of the PLA description is indicated with:

.end

Programs capable of handling split and folded arrays employ the following format:

Symbols for *AND* Plane

Contact Signal	No Contact	Explanation
1	-	Normal contact, no splits or folds
!	_	Split below
;	,	Fold to the right
:	.	Split below and fold to the right

Symbols for *OR* Plane

Contact Signal	No Contact	Explanation
I	~	Normal contact, no splits or folds
i	=	Split below
|	'	Fold to the right
j	"	Split below and fold to the right

Additional Symbols

Symbol	Explanation
*	Input buffer
+	Output buffer
X	No buffer
c	Contact within AND or OR plane
> <	Routing lines to contacts for multiple folds

Note that the decoding function of the AND plane is separated from the specification of its connectivity. This makes the AND and OR plane specifications

identical. Note also, that only one contact row can occur between successive product rows. The contact row may have two contacts, one from the left and another from the right.

These programs handle the following more general set of "." parameters:

.i	<number of inputs>
.o	<number of outputs>
.p	<number of product terms>
.top	<labels of inputs and outputs from the top
.left	<labels of inputs and outputs from the left
.right	<labels of inputs and outputs from the right
.bottom	<labels of inputs and outputs from the bottom

The first group of parameters must precede the second group. If there is only one AND or OR plane **plaid** assumes it to be the leftmost one.

In addition to the above parameters, **plaid** can handle the following parameters:

.il	<number of left-AND plane inputs>
.ir	<number of right-AND plane inputs>
.ol	<number of left-OR plane inputs>
.or	<number of right-OR plane inputs>
.p	<number of product terms>
.ilt	<labels left-top-AND plane>
.ilb	<labels left-bottom-AND plane>
.irt	<labels right-top-AND plane>
.irb	<labels right-bottom-AND plane>
.olb	<labels left-bottom-OR plane>
.olt	<labels left-top-OR plane>
.orb	<labels right-bottom-Or plane>
.ort	<labels right-top-Or plane>
.pl	<labels left product terms>
.pr	<labels right product terms>

SEE ALSO

blam(CAD1), eqntott(CAD1), mkfsm(CAD1), panda(CAD1), plaid(CAD1), plasort(CAD1), pop(CAD1)

APPENDIX B

PANDA Manual

This appendix contains the PANDA manual pages which create PLAs.

NAME

panda – technology independent PLA generator for multiply-folded PLAs

SYNOPSIS

panda [-acpvV] [-s *style*] [-G *num*] [-S *numR*] [-l *num*] [-t *template_name*] [-M *num*] [-D *num1 num2*] [-o *output_file*] *input_file*

DESCRIPTION

panda is a PLA generator that generates multiply-folded, simply-folded, and non-folded PLAs. **Panda** is a program written with the **tpack**(CAD3) system.

The input format for **panda** is compatible with the *.machine* output of **pleasure**(CAD1). Including the *.machine* control line in the **pleasure** input file results in the proper output format for **panda**.

Input files to **panda** contain *control lines* and a personality matrix of the PLA to be made. Each control line must begin with a "." (dot or period). The following control lines are understood by **panda**:

.[and | or] *num1 num2* [*num3* ...]

This line describes the structure of the PLA, with the **and** or **or** specifying that the leftmost plane is an *AND* or *OR* plane. The *AND* plane is the input plane and the *OR* plane is the output plane. The numbers following the first plane designator are the numbers of inputs/outputs (depending on which plane is first) in each successive plane. For instance, the control line: " *.or 9 3 4 5* " means that this PLA has an OR-AND-OR-AND structure with 9 outputs in the first OR plane, 3 inputs in the next AND plane, 4 outputs in the next OR plane, and 5 inputs in the last AND plane. Note that at least two numbers must follow the first plane designator since a PLA must have at least one AND and one OR plane.

.row [*number of rows*]

This line describes the height of the input personality matrix.

.top [*label1 label2 label3* ...]
.bottom [*label1 label2 label3* ...]
.left [*label1 label2 label3* ...]
.right [*label1 label2 label3* ...]

These control lines list the labels for inputs and outputs along the top, bottom, left, and right respectively, of the PLA. The label "0" is not allowed. Note that these labels are not designated as being either inputs or outputs. The AND-OR-AND... structure of the PLA is determined by the **.and|or** control line.

.product [*l1 l2 l3* ...]

This control line lists the labels for product rows within the PLA. These labels are used for debugging within the PLA and can be automatically numbered if no labels are put in. The label "0" is not allowed.

.end

This line signals the end of the input file.

The personality matrix format is compatible with **pleasure**, see PLA(5) for details. The table below summarizes the symbols **panda** accepts.

Symbols for *AND* Plane		
Contact Signal	No Contact	Explanation
1	-	Normal contact, no splits or folds
!	_	Split below
:	,	Fold to the right
;	.	Split below and fold to the right

Symbols for *OR* Plane		
Contact to Output Signal	No Contact	Explanation
I	~	Normal contact, no splits or folds
i	=	Split below
|	'	Fold to the right
j	"	Split below and fold to the right

Additional Symbols	
Symbol	Explanation
*	Input buffer
+	Output buffer
X	No buffer
c	Contact within AND or OR plane
> <	Routing lines to contacts for multiple folds

An example of an input file is shown below.

```
.and   2 4 3 1
.row   8
  * X     ++X+    X * X     +
X1---    ~i~I    --1---    ~X
X----    I~~i    ,;1-1-    iX
X_!--    ~|~I    1-_!--    ~X
X1--1    |~~I    -----1    ~X
X--1-    i~~~    ---1--    ~X
*c                          X
X>c                         X
X!_1-    ~|~I    1-;.--    ~X
X-1--    I~I~    1-----    IX
X---1    ~~~~    --1---    IX
  * *    ++++    * * *     +
.end
```

Note that the AND plane is expanded such that each input is represented as two columns, one for the signal, the other for its complement. Note also, that only

one contact row can occur between successive product rows. The contact row may have two contacts, one from the left and another from the right.

STYLES OF PLAs AVAILABLE

The following style of PLAs is currently supported:

CS3 CMOS static version with p-channel pull-ups as resistive loads. 3 micron MOSIS rules. The pull-ups are placed in the between the AND and OR planes.

It is easy to create a template for a new style of PLA, and **panda**(CAD5) has information on how to do it.

OPTIONS

-a produce **caesar**(CAD1) format (this is the default)

-c produce CIF format

-p (pipe mode) Send the output to **stdout**.

-v Be verbose, and show (in the **caesar** output) how the PLA was constructed from its basic components.

-V Be verbose, and print out information about what tpla is doing. This option implies **-v**.

-s *style*
The next argument specifies the style of PLA to generate. (This causes panda to use the file **~cad/lib/panda/pa-***style***.tp** as its template).

-G *num*
Insert an extra ground line every *num* rows in the AND plane and every *num* columns in the OR plane. This defaults to what is appropriate for the static 3 micron **CMOS** PLA, approximately every 10 rows. Note that for styles other than **CS3**, this option should be used for specifying extra ground lines.

-S *num*
Stretch power and ground lines by *num* lambda. This defaults to whatever is appropriate for the corresponding **CMOS** PLA. Note that for styles other than **CS3**, this option should be used for specifying stretching power and ground lines.

-l *num*
Set lambda to *num* centimicrons. (200 is the default)

-t *template_name*
The next argument specifies the template to use, this normally defaults to the standard library. A **.tp** suffix is added if no suffix was specified. This option is useful for generating styles of PLAs that are not included in the standard library.

-M *num*
Normally **tpack** merges rectangles to form maximal horizontal strips, just like **caesar**(CAD1). If the **-M** option is present **tpack** will only look back through the last *num* rectangles on each layer when doing merges. A small value for *num* will make **tpack** run faster, but not all possible merges will be found. The **-v** option gives information about the number of merges done.

-D *num1 num2*

The *Demo* or *Debug* option. This option will cause **tpack** to place only the first *num1* tiles, and the last *num2* of those will be outlined with rectangular labels. In addition, if a tile called "**blotch**" is defined then a copy of it will be placed in the output tile upon each call to the *align* function during the placing of the last *num2* tiles. The blotch tile will be centered on the first point passed to *align*, and usually consists of a small blotch of brightly colored paint. This has the effect of marking the alignment points of tiles. The last tile painted into is assumed to be the output tile.

-o *output_file*

The next argument is taken to be the base name of the output file. The default is the input file name with any extensions removed. If the input comes from the standard input and the **-o** option is not specified then the output will go to the standard output.

input_file

The file containing the control lines and personality matrix. See PLA(CAD5) for a description of the personality matrix symbols. If this filename is omitted then the input is taken from the standard input.

FILES

~cad/bin/panda -- executable
~cad/src/panda/* -- source
~cad/lib/panda/pa-*.tp -- standard templates for PLAs

SEE ALSO

eqntott(CAD1), pop(CAD1), plasort(CAD1), pla(CAD5), tpla(CAD5), tpack(CAD3)

AUTHOR

Grace H. Mah

BUGS

The -G and -S options have no way of knowing what the grounding requirements are for the style of PLA actually being generated.

The default placement of extra ground lines and stretching of power and ground lines are based upon the CMOS static pullup, **pa-CS3.tp**, template.

This program inherits any bugs that may exist in **tpack**(CAD3).

APPENDIX C

PANDA Tile Definition

This appendix contains the PANDA manual pages which define template tiles.

NAME

Template format for panda(CAD1)

DESCRIPTION

Making a template for **panda** consists of first designing a sample multiply-folded PLA in the desired style, and then labeling *tiles* using the **caesar**(CAD1) graphics editor. A *tile* is a rectangular area of paint, and is defined by a named label outlining the area. **Panda** assembles these tiles, row by row, to form a multiply-folded PLA.

There are 11 groups of tiles in a **panda** template:

1) the core of the AND plane
2) the core of the OR plane
3) the sides of the AND plane
4) the sides of the OR plane
5) the top and bottom of the AND plane
6) the top and bottom of the OR plane
7) the tiles between planes
8) the horizontal spacing tiles in the AND plane
9) the vertical spacing tiles in both planes
10) the horizontal ground tiles
11) the vertical ground tiles

Any of the tiles not in the core areas may contain linear labels with the name **[GND]** or **[Vdd]**. Linear labels are lines with a name attached. Labels with the names **[GND]** and **[Vdd]** will be stretched to allow increased current through the PLA. That is, the tile would be figuratively "cut" along the line label and then the two "pieces" would be stretched apart by a designated amount. A given tile may contain many occurrences of these linear labels, but none of them can be colinear. If 2 labels within a tile are colinear, the stretching of one of them will turn the other one into a rectangle, and it is not possible to stretch along a rectangle.

Point labels may occur anywhere in a tile. Global point labels **GND!** and **Vdd!** may be put in corner tiles, to be placed in all **panda**-generated PLAs. The point label **$input$** may occur in tiles on the top or bottom of the AND plane, and the **$output$** label may occur in tiles on the top or bottom of the OR plane. These labels will be replaced with the name of the corresponding input or output. There should be no more than one **$input$** label on each input, and no more than one **$output$** label on each output. Tiles between the AND and OR cores may contain point labels, **$product$**, which are placed when a product term bridges between AND and OR planes. These **$product$** labels are useful for debugging purposes within a PLA.

Panda builds PLAs by rows of tiles. All of the tiles in a row (except the bottom row) have their bottom edges aligned. The top row of tiles would be made up of edge tiles arranged from the left to the right of the PLA, all stacked together along a line. A product row of the PLA would have a left edge tile, core tiles for the AND and OR plane(s), and a right edge tile, all aligned. Each row of tiles is placed one beneath the next according to the alignment of the first tile on the left side of each row. The alignment of the first tile in each row will be discussed below. In the last row of tiles (bottom tiles), all of the tiles are aligned with each other by their top edges. The whole row is aligned to the previous string above it by aligning to the bottom edge of that previous row (the last product row).

THE CORE OF THE *AND* PLANE:

Required: **sp-and, l0-and, l1-and, l!-and, l;-and, l:-and, r0-and, r1-and, r!-and, r;-and, r:-and, lc-and, rc-and**

Optional: **lu-and, lh-and, lb-and, ru-and, rh-and, rb-and**

These tiles contain transistors that implement the PLA function. The vertical and horizontal pitch of the core of the AND plane is set by the tile **sp-and**. The first character of the tile name indicates whether it is a *left* tile or a *right* tile. Left tiles are placed in every other column in the AND plane core, starting with the first column. Right tiles are placed in every other column starting with the second column. The tile **sp-and** determines the amount of overlap between columns.

The second character of the name represents the function of the tile according to the format for folded PLAs (see PLA(CAD5)). For instance, a *1* stands for for tiles that contain a transistor and a *0* for tiles that pass the input line up to the next tile but have no transistor. For folded PLAs, additional tiles have, for the second character of the name, an *!* for tiles that contain a transistor and are split below, a *;* for tiles that contain a transistor and are folded to the right, and an *:* for tiles that contain a transistor and are folded below and to the right. For multiply-folded PLAs, a *c* represents a contact tile that will make a contact between a vertical and horizontal signal line, as is the case when an input or output is brought into the core from the sides of the PLA.

Optional tiles that minimize excess interconnect in the length and width of the PLA begin with *u* for tiles that do not pass the vertical signal up to the next row, *h* for tiles that do not pass the horizontal signal across to the next column, and *b* for tiles that do not pass both the vertical and the horizontal signals. These tiles are not necessary for the functionality of PLAs, but they help reduce capacitances and therefore delay times throughout the PLA.

Columns in the PLA AND plane are grouped into (left, right) pairs, according to (signal, complement) pairs. The selection of the core tiles in these column pairs is determined by the symbols occuring in a personality matrix as is described in PLA(CAD5). If the symbol is a "0", then the tiles **l0-and** and **r1-and** are placed in the column as a pair. If the symbol is a "1", then the tiles **l1-and** and **r0-and** are placed. If the symbol is a "!", then the tiles **l!-and** and **r0-and** are placed. If the symbol is a "o", then the tiles **l0-and** and **r!-and** are placed. Similar pairings of tiles are done for the symbols "@", ";", "#", and ":".

The optimizing tiles are used to replace **l0-and** and **r0-and** tiles when signal lines do not need to extend the full length and width of the PLA. For example, all **l0-and** tiles above the topmost **l1-and** (**l!-and**, **l;-and**, and **l:-and**) tile are replaced with **lu-and** tiles in order to allow shortened vertical poly lines. A similar substitution is done with **ru-and** tiles in the alternating right columns.

In a similar fashion, all **l0-and** and **r0-and** tiles that extend horizontally to the edge of the PLA are replaced with **lh-and** and **rh-and**, respectively. If a horizontal *and* vertical line can be minimized, the **lb-and** and **rb-and** tiles automatically replace the **l0-and** and **r0-and** tiles.

All of the tiles in this group are assumed to be of the same height. (However, creative designers may design otherwise.) When **panda** aligns a row of core tiles in the AND plane, the **sp-and** tile is used to control the overlap between column pair tiles. When the core of the AND plane is made, the lower left corner of a left (or right) tile is aligned to the lower left corner of the **sp-and** tile. The next right

(or left) tile is then aligned such that its lower left corner is aligned to the **sp-and** tile's lower right corner. This pattern of placing a core tile (left or right), spacing a distance of **sp-and**, and then placing the next tile (left or right), is done throughout the core of the AND plane. It is highly recommended that the user look at the **pa-CS3.tp** template before trying to define a new one.

THE CORE OF THE *OR* PLANE:

Required: **sp-or, u0-or, u1-or, ui-or, u|-or, uj-or, r0-or, d1-or, di-or, d|-or, dj-or**

Optional: **uu-or, uh-or, ub-or, du-or, dh-or, db-or**

These tiles are similar to the ones in the AND plane. A *u* as the first character indicates that the tile occurs in every other row, starting with the first (the *up* rows). A *d* indicates that the tile will be placed in the other *down* rows. All of the tiles in this group are assumed to be of the same width. The tile **sp-or** sets the horizontal spacing for the OR plane. Since **panda** builds PLAs row by row, all tiles defined in the OR plane for one row are aligned in a line by their bottom edges.

THE SIDES OF THE *AND* PLANE:

Required: **ul-and, ll-and, ur-and, lr-and, right-and, left-and, left-in, right-in**

Optional: **hul-and, hur-and, nul-and, nur-and, vul-and, vll-and, vur-and, vlr-and, nright-and, nleft-and**

These tiles align to the left and right sides of the AND plane, when the AND plane is an exterior plane, as in an AND-OR-AND structure. The tile **ul-and** is placed in the upper left corner of the AND plane, while the tile **ll-and** goes in the lower left corner. Along the left side of the AND plane, each product row has a **left-and** tile. For AND planes on the outer right edge of the PLA, the tile **ur-and** is placed in the upper right corner of the AND plane, while the tile **lr-and** goes in the lower right corner. The rows in between the top and bottom contain **right-and** tiles along the right side of the AND plane.

AND planes which are interior to a PLA, such as in an OR-AND-OR PLA, do not have side tiles. When a multiply-folded PLA is made, the tiles **left-in** and **right-in** replace the **left-and** and **right-and** tiles, respectively. These side input buffers are twice as tall as the **left-and** and **right-and** tiles because connections must be made to the input signal and its complement in the multiply-folded column.

Optional corner tiles provide for PLAs that do not have folding. The extra overhead is in either the height (those tiles that begin with *h*) or the width (those tiles that begin with *w*) of the tile. Tiles that begin with *n* are used in PLAs that are not folded and thus, do not have the extra overhead in the horizontal and vertical direction.

The tiles along the top of the PLA are placed in a row with their bottom edges aligned. The tiles along the bottom of the PLA are placed in a row with their top edges aligned. The side tiles of the AND plane are assumed to match the height of the core tiles of the AND plane and to be aligned along the same bottom edge as the core tiles for each row.

Since the first tile in a row determines the alignment for the whole row, the **left-and** or **left-in** tiles (for PLA structures which begin with an AND plane) are assumed to be stacked exactly below succeeding rows. That is, there is *no* overlapping of left side tiles for AND-first PLAs.

THE SIDES OF THE *OR* PLANE:

Required: **ul-or, ll-or, ur-or, lr-or, rightu-or, rightd-or, leftu-or, leftd-or, leftu-out, leftd-out, rightu-out, rightd-out**

Optional: **hul-or, hur-or, nul-or, nur-or, vul-or, vll-or, vur-or, vlr-or nrightu-or, nrightd-or, nleftu-or, nleftd-or**

These tiles function in a manner analogous to the tiles on the sides of the AND plane. Note that the **rightu-or** tile is placed in the *up* rows, while the **rightd-or** tile is placed in the *down* rows. The output buffer tiles, **leftu-out, leftd-out, rightu-out,** and **rightd-out,** are the same height as tiles in the core of the OR plane.

When creating rows of tiles (for PLAs that begin with an OR plane), there is some overlapping of these first "left" tiles. The **leftu-or** (or **leftu-out** or **leftu-out** or **leftd-out**) tiles, for PLA structures which begin with an OR plane, are assumed to be stacked below succeeding rows such that they overlap an amount controled by **sp-or**. That is, when aligning a new "left-or" tile for a new row, the sequence of alignments is as follows: align the lower left corner of the previous row with the upper left corner of the **sp-or** tile, align the lower left corner of the **sp-or** tile with the lower left corner of the "left-or" tile. Thus, there *is* overlapping of left edge tiles for OR-first PLAs.

THE TOPS AND BOTTOMS OF *AND* PLANES:

Required: **top-in, bot-in, top-and, bots-and, tops-and**

Optional: **ntop-and**

These tiles function in a manner analogous to the tiles on the left and right sides of the AND and OR planes. The **top-in** and **bot-in** tiles contain input buffers coming into the PLA from the top and bottom, respectively. All of these tiles are only placed in every other column, starting with the first because connections must be made with signal and complement lines. The tiles **bots-and** and **tops-and** control the amount of horizontal spacing and overlap between adjacent tiles in the same way that the **sp-and** tile controls horizontal spacing in the AND plane. The **ntop-and** tile is used for PLAs that do not have column folds.

The top tiles are aligned by their bottom edges in a row. The bottom tiles are aligned by their top edges.

THE TOPS AND BOTTOMS OF *OR* PLANES:

Required: **topl-out, topr-out, botl-out, botr-out, topl-or, topl-or**

Optional: **ntopl-or, ntopr-or, nbotl-or, nbotr-or**

These tiles function in a manner analogous to the tiles on the top and bottom sides of the AND plane, except that the **topl-or** tile is placed in every other column starting with the first (the *left* columns) while **topr-or** is placed in the alternating columns. These tiles are also aligned along the top edges (unlike the top tiles in the AND plane which are aligned along the bottom edges).

THE TILES BETWEEN PLANES:

Required: **topmid-ao, botmid-ao, topmid-oa, botmid-oa, midu-ao, midd-ao, midu-oa, midd-oa, nmidu-ao, nmidd-ao, nmidu-oa, nmidd-oa, cmidd-ao, cmidu-ao, cmidd-oa, cmodu-oa**

Optional: **ntopmid-ao, ntopmid-oa**

These tiles are between the AND and OR planes. When an AND plane is followed by an OR plane, the *ao* tiles are used, and when an OR plane is followed by an AND plane, the *oa* tiles are used. The tiles that connect product rows between

the AND and OR cores, **midd_ao**, **midu_ao**, **midd_oa**, and **midu_oa**, usually contain some type of circuit element that pulls up each product row. Tiles that begin with a *n* do not contain these circuit elements and are placed when the product row does not need to be connected between two planes. The tiles that begin with a *c* are for middle tiles in contact rows.

In the top row of tiles, the **topmid-ao** tile matches the AND tiles in the top rows by its lower left corner. The **topmid-oa** tile matches the OR tiles in the top rows by its upper left corner. In the bottom row of tiles, the **botmid-ao** and **botmid-oa** tiles match the other tiles in the bottom rows by their top edges. The "midd" and "midu" tiles are assumed to be the same height as the core tiles in the AND plane. Their bottom edges are aligned along the same edge as the other tiles in their row.

THE HORIZONTAL SPACING TILES IN THE *AND* PLANE

Required: **sh-and**, **both-and**, **toph-and**

Optional: **shx-and**

These tiles handle the extra horizontal spacing that is needed in the AND plane for folding rows in the AND plane. When two logical rows must share one physical row, the structure of the AND plane (pairings of left and right core tiles) is such that they may need extra spacing in the horizontal direction. This occurs when a right tile (containing a transistor) of one logic row is adjacent to the left tile (containing a transistor) of another logic row. Usually, one contact is used between these transistors to connect them to the product term. However, when a logical fold must be made between these transistors, the contact can not be split, so the duplication of the contact and the extra space needed for physical correctness is contained in the **shx-and** (for no connection between right and left tiles) and **sh-and** tiles (for connecting between right and left tiles, when the logical rows are not folded but horizontal spacing was required on another row). The **both-and** and **toph-and** tiles provide the same amount of horizontal spacing in the top and bottom tiles in the AND plane.

THE VERTICAL SPACING TILES IN BOTH PLANES

Required: **sv-or**, **lv-and**, **rv-and**, **midv-ao**, **midv-oa**, **shv-and**, **leftv-and**, **leftv-or**, **rightv-and**, **rightv-or**

Optional: **svx-or**, **lvx-and**, **rvx-and**

These tiles function in a manner analogous to the horizontal space tiles in the AND plane. In the OR plane, the up and down tiles require the **sv-or** (for connecting) and **svx-or** (for non-connecting) tiles for vertical splits along columns in the OR plane. The consequences of spacing out the OR plane vertically, require the tiles, **leftv-and**, **rightv-and**, **leftv-or**, and **rightv-or** along the sides of the PLA. In the AND plane, the tiles **lv-and** and **rv-and**, allow for vertical spacing and connect the vertical signals. The tile **lvx-and** and **rvx-and** provide vertical spacing and donot connecting (the *x* in the name) vertical signals. The **shv-and** tile is placed when vertical and horizontal spacing intersect in the core of the AND plane. The **midv-ao** and **midv-oa** tiles contain the vertical spacing in the middle tiles between planes.

These tiles are assumed to be the same height as the core tiles in the AND plane. Their bottom edges are aligned along the same edge as the other tiles in their row.

THE HORIZONTAL GROUND TILES

Required: **HGleft-and, HGright-and, HGl-and, HGr-and, HGright-or, HGleft-or, HGmid-ao, HGmid-oa, HG-or, HGabv-and**

Optional: **nHGleft-and, nHGright-and, nHGright-or, nHGleft-or, HGlx-and, HGrx-and**

For the extra ground lines that are needed for large PLAs, these horizontal ground tiles contain metal lines that run the width of the PLA from one side to the other. These tiles are aligned in the PLA in a manner similar to the horizontal spacing tiles.

THE VERTICAL GROUND TILES

Required: **VGtop-or, VG-or, VGd-or, HVG-or, cHVG-or, VGbot-or**

Optional: **nVGtop-or, VGux-or, VGdx-or**

These tiles are aligned in the PLA in a manner analogous to the horizontal ground tiles and vertical spacing tiles.

SEE ALSO

panda(CAD1), tpack(CAD3), PLA(CAD5), tpla(CAD1), tpla(CAD5)

AUTHOR

Grace H. Mah

APPENDIX D

TPACK Manual

This appendix contains the manual pages for TPACK.

NAME

tpack – routines for generating semi-regular modules

DESCRIPTION

Tpack (tile packer) is a library of 'C' routines that aid the process of generating semi-regular modules. Decoder planes, barrel shifters, and PLAs are common examples of semi-regular modules.

Using Caesar, a tpack user will draw an example of a finished module and then break it into tiles. These tiles represent the building blocks for more complicated instances of the module. The tpack library provides routines to aid in assembling tiles into a finished module.

MAKING AN EXAMPLE MODULE

The first step in using tpack is to create an example instance of the module, called a *template*. The basic building blocks of the structure, or *tiles*, are then chosen. Each tile should be given a name by means of a rectangular label which defines its contents. If the tiles in the module do not abut (e.g. they overlap) it is useful to define another tile whose size indicates how far apart the tiles should be placed.

Templates should be in Caesar format and, by convention, end with a **.tp** suffix. With some programs, it is possible to generate the same structure in a different technology or style by changing just the template. If this is the case, each template should have a filename of the form basename-*style*.**tp**. The *style* part of the filename interacts with the **-s** option (see later part of this manual).

WRITING A TPACK PROGRAM

A tpack program is the 'C' code which assembles tiles into the desired module. Typically this program reads a file (such as a truth table) and then calls the tile placement routines in the tpack library.

The tpack program must first include the file **~cad/lib/tpack.h** which defines the interface to the tpack system. Next the **TPinitialize** procedure is called. This procedure processes command line arguments, opens an input file as the standard input (**stdin**), and loads in a template.

The program should now read from the standard input and compute where to place the next tile. Tiles may be aligned with previously placed tiles or placed at absolute coordinates. If a tile is to overlap an existing tile the program must space over the distance of the overlap before placing the tile.

When all tiles are placed the program should call the routine **TPwrite_tile** to create the output file that was specified on the command line.

To use the tpack library be sure to include it with your compile or load command (e.g. **cc** *your_file* **~cad/lib/tpack.lib**).

ROUTINES

Initialization and Output Routines

TPinitialize(*argc*, *argv*, *base_name*)

The tpack system is initialized, command line arguments are processed, and a template is loaded. The file descriptor **stdin** is attached to the input file specified on the command line. The template's filename is formed by taking the *base_name*, adding any extension indicated by the **-s** option, and then adding the **.lp** suffix if no suffix was provided. The **-t** option allows the user to override

base_name from the command line.

Argc and *argv* should contain the command line arguments. *Argc* is a count of the number of arguments, while *argv* is an array of pointers to strings. Strings of length zero are ignored (as is the flag consisting of a single space), in order to make it easy for the calling program to intercept its own arguments. *Argc* and *argv* are of the same structure as the two parameters passed to the main program. A later section of this manual summarizes the command line options.

TPload_tiles(*file_name*)
The given *file_name* is read, and each rectangular label found in the file becomes a tile accessible via TPname_to_tile. No extensions are added to *file_name*.

TILE TPread_tile(*file_name*)
A tile is created and *file_name* is read into it. The tile is returned as the value of the function.

TPwrite_tile(*tile*, *filename*)
The tile *tile* is written to the file specified by *filename*, with **.ca** or **.cif** extensions added. See the description of the **-o** option for information on what file name is chosen if *filename* is the null string. The choice between Caesar or CIF format is chosen with the **-a** or **-c** command line options.

Tile creation, deletion, and access

TPdelete_tile(*tile*)
The tile *tile* is deleted from the database and the space occupied by it is reused.

TILE TPcreate_tile(*name*)
A new, empty tile is created and given the name *name*. This name is used by the routine **TPname_to_tile** and in error messages. The type **TILE** returned is a unique ID for the tile, not the tile itself. Currently this is implemented by defining the type TILE to be a pointer to the internal database representation of the tile.

int TPtile_exists(*name*)
TRUE (1) is returned if a tile with the given *name* exists (such as in the template or from a call to TPcreate_tile).

TILE TPname_to_tile(*name*)
A value of type **TILE** is returned. This value is a unique ID for the tile that has the name *name*. This name comes from a call to TPcreate_tile(), or from the rectangular label that defined it in a template that was read in by TPread_tiles() or TPinitialize(). If the tile does not exist then a value of NULL is returned and an error message is printed.

RECTANGLE TPsize_of_tile(*tile*)

A rectangle is returned that is the same size as the tile *tile*. The rectangle's lower left corner is located at the coordinate (0, 0). All coordinates in tpack are specified in half-lambda.

Painting and Placement Routines

RECTANGLE TPpaint_tile(*from_tile*, *to_tile*, *ll_corner*)
The tile *from_tile* is painted into the tile *to_tile* such that its lower left corner is placed at the point *ll_corner* in the tile *to_tile* . The location of the newly painted area in the output tile is returned as a value of type RECTANGLE. The tile *to_tile* is often an empty tile made by **TPcreate_tile()**. The point *ll_corner* is almost never provided directly, it is usually generated by routines such as **align()**.

TPdisp_tile(*from_tile*, *ll_corner*)
A rectangle the size of *from_tile* with the lower left corner located at *ll_corner* is returned. Note that this routine behaves exactly like the routine TPpaint_tile except that no output tile is modified. This routine, in conjunction with the **align** routine, is useful for controlling the overlap of tiles.

RECTANGLE TPpaint_cell(*from_tile*, *to_tile*, *ll_corner*)
This routine behaves like **TPpaint_tile()** except that the *from_tile* is placed as a subcell rather than painted into place. The tile *from_tile* must exist in the file system (i.e. it must have been read in from disk or have been written out to disk).

Label Manipulation Routines

TPplace_label(*tile*, *rect*, *label_name*)
A label named *label_name* is place in the tile *tile*. The size and location of the label is the given by the RECTANGLE *rect*.

int TPfind_label(*tile*, *&rect1*, *str*, *&rect2*)
The tile *tile* is searched for a label of name *str*. The location of the first such label found is returned in the rectangle *rect2*. The function returns 1 if such a label was found, and 0 otherwise. The rectangle pointer *&rect1*, if non-NULL, restricts the search to an area of the tile.

TPstrip_labels(*tile*, *ch*)
All labels in the tile *tile* that begin with the character *ch* are deleted.

TPstretch_tile(*tile*, *str*, *num*)
The string *str* is the name of one or more labels within the tile *tile*. Each of these labels must be of zero width or zero height, i.e. they must be lines. Each of these lines define a line across which the tile will be stretched. The amount of the stretch is specified by *num* in units of **half**-lambda. Stretching such a line turns it into a rectangle. Note that if the tile contains 2 lines that are co-linear, the stretching of one of them will turn both into rectangles.

Point-Valued Routines

POINT tLL(*tile*)
POINT tLR(*tile*)
POINT tUL(*tile*)
POINT tUR(*tile*)

The location of the specified corner of tile *tile*, relative to the tile's lower left corner, is returned as a point. LL stands for lower-left, LR for lower-right, UL for upper-left, and UR for upper-right. Note that **tLL()** returns (0, 0).

POINT rLL(*rect*)
POINT rLR(*rect*)
POINT rUL(*rect*)
POINT rUR(*rect*)

The location of the specified corner of the rectangle *rect* is returned as a point. LL stands for lower-left, LR for lower-right, UL for upper-left, and UR for upper-right.

POINT align(*p1*, *p2*)

A point is computed such that when added to the point *p2* gives the point *p1*. *p1* is normally a corner of a rectangle within a tile and *p2* is normally a corner of a tile. In this case the point computed can be treated as the location for the placement of the tile.

For example, TPpaint_tile(outtile, fromtile, align(rUL(rect), tLL(fromtile))) will paint the tile *fromtile* into *outtile* such that the lower left corner of *fromtile* is aligned with the upper-left corner of *rect*. In this example *rect* would probably be something returned from a previous TPpaint_tile() call.

Point and Rectangle Addition Routines

POINT TPadd_pp(*p1*, *p2*)
POINT TPsub_pp(*p1*, *p2*)

The points *p1* and *p2* are added or subtracted, and the result is returned as a point. In the subtract case *p2* is subtracted from *p1*.

RECTANGLE TPadd_rp(*r1*, *p1*)
RECTANGLE TPsub_rp(*r1*, *p1*)

The rectangle *r1* has the point *p1* added or subtracted from it. This has the effect of displacing the rectangle in the X and/or Y dimensions.

Miscellaneous Functions

int TPget_lambda()

This function returns the current value of lambda in centi-microns.

INTERFACE DATA STRUCTURES

In those cases where tiles must be placed using absolute, (half-lambda) coordinates, it is useful to know that **RECTANGLE**s and **POINT**s are defined as:

```
typedef struct {
   int x_left, x_right, y_top, y_bot;
} RECTANGLE;

typedef struct {
   int x, y;
} POINT;
```

The variable **ORIGIN_POINTER** is predefined to be (0, 0). **ORIGIN_RECT** is defined to be a zero-sized rectangle located at the origin.

OPTIONS ACCEPTED BY TPinitialize()

Typical command line: *program_name* [-t *template*] [-s *style*] [-o *output_file*] *input_file*

-a produce Caesar format (this is the default)

-c produce CIF format

-v be verbose (sequentially label the tiles in the output for debugging purposes; also print out information about the number of rectangles processed by tpack)

-s *style*
generate output using the template for this style (see TPinitialize for details)

-o The next argument is taken to be the base name of the output file. The default is the input file name with any extensions removed. If there is not input file specified and no -o option specified, the output will go to **stdout**.

-p (pipe mode) Send the output to **stdout**.

-t The next argument specifies the template base name to use. This overrides the default supplied by the program. (see TPinitialize)

-l *num*
Set lambda to *num* centimicrons. (200 is the default)

input_file
The name of the file that the program should read from (such as a truth table file). If this filename is omitted then the input is taken from the standard input (such as a pipe).

-M *num*
Tpack can merge rectangles to form maximal horizontal strips, just like Caesar(CAD). If the **-M** option is present tpack will look through the last *num* rectangles on each layer to look for possible merges. A large value for *num* will make tpack run slower, but more possible merges will be found and the resulting file may be smaller. Normally only a little bit of merging is done (*num* defaults to 5). The **-v** option gives information about the number of merges done.

-D *num1 num2*
The *Demo* or *Debug* option. This option will cause **tpack** to place only the first *num1* tiles, and the last *num2* of those will be outlined with rectangular labels. In addition, if a tile called "blotch" is defined then a copy

of it will be placed in the output tile upon each call to the *align* function during the placing of the last *num2* tiles. The blotch tile will be centered on the first point passed to *align*, and usually consists of a small blotch of brightly colored paint. This has the effect of marking the alignment points of tiles. The last tile painted into is assumed to be the output tile.

EXAMPLE

It is highly recommended that the example in **~cad/src/quilt** be examined. Look at both the template and the 'C' code. A more complex example is in **~cad/src/tpla**.

FILES

~cad/lib/tpack.h	(definition of the tpack interface)
~cad/lib/tpack.lib	(linkable tpack library)
~cad/src/quilt/*	(an example of a tpack program)
~cad/lib/caesar/*.tech	(technology description files)

ALSO SEE

Caesar(CAD)
'C' Manual
Quilt(CAD)
Tpla(CAD)
Robert N. Mayo and John K. Ousterhout, *Pictures with Parentheses: Combining Graphics and Procedures in a VLSI Layout Tool*, Proceedings of the 20th Design Automation Conference, June, 1983.

AUTHOR

Robert N. Mayo

BUGS

When a tile contains part of a subcell, or touches a subcell, then the whole subcell is considered to be part of the tile. The same goes for arrays of subcells.

References

[1] 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 Jour. of Res. and Dev.*, vol. 19, pp. 110-119, March, 1975.

[2] B. Cmelik, "EQNTOTT Reference Manual," *1983 VLSI Tools*, Report No. UCB/CSD 83/115, CS Div. of EECS, U.C. Berkeley 1983.

[3] G. Hamachi, "PEG Manual," *1983 VLSI Tools*, Report No. UCB/CSD 83/115, CS Div. of EECS, U.C. Berkeley 1983.

[4] P. Simanyi, A.R. Newton, A.L. Sangiovanni-Vincentelli, "The POP PLA Optimization Program," in preparation.

[5] R. Rudell, "ESPRESSO-IIC Manual," *CAD Toolbox User's Manual*, U.C. Berkeley.

[6] J. Deutch, "SIMPLE Manual," *CAD Toolbox User's Manual*, U.C. Berkeley.

[7] M. Hofmann, "A Method of Topological Compaction of Programmed Logic Arrays," *Masters Report*, Dept. of EECS, U.C. Berkeley 1983.

[8] G. De Micheli, "Computer-Aided Synthesis of PLA-Based Systems," *PhD Dissertation*, Dept. of EECS, U.C. Berkeley 1983.

[9] R. N. Mayo, "Combining Graphics and Procedures in a VLSI Layout Tool: The Tpack System," *Master's Report*, Report No. UCB/CSD 84/166, CS Div. of EECS, U.C. Berkeley, January, 1984.

[10] C.H.Roth, *Fundamentals of Logic Design*, West Publishing Company, St. Paul, Minn. 1979.

[11] S. Kang, "Automated Synthesis of PLA Based Systems," *PhD Dissertation*, Stanford University, 1981.

[12] R.N. Mayo, J.K. Ousterhout, "Pictures with Parentheses: Combining Graphics and Procedures in a VLSI Layout Tool," *Proc. of 20th DAC*, June 1983.

[13] J.K. Ousterhout, "Caesar: An Interactive Editor for VLSI Layouts," *VLSI Design*, pp34-38, Fourth Quarter 1981.

[14] K. Keller, A.R. Newton, "KIC2: A Low-Cost, Interactive Editor for Integrated Circuit Design," pp. 305-306, *Digest of Papers, Compcon82*, IEEE Computer Society 1982.

[15] C. Mead, L. Conway, *Introduction to VLSI Systems*, Addison Wesley, 1980.

[16] H.A. Landman, "Automatic Layout of Optimized PLA Structures," *Masters Report*, CS Div. of EECS, U.C. Berkeley 1980.

[17] A. Vladimirescu, K. Zhang, A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli, *SPICE Version 2G User's Guide*, U.C. Berkeley, August, 1981.