Chapter 4: Implementing Integrated System Designs: From Circuit Topology to Patterning Geometry to Wafer Fabrication

Sections:

This chapter presents the basic concepts involved in implementing integrated system designs, from the system designer's point of view. Tools are described which help the designer produce the geometrical layout patterns for each layer of an integrated system given the logic, circuit, or topological level design of the system. Procedures are described for encoding these layout patterns and then using the encoded layouts in the patterning and fabrication processes to implement the integrated system. In addition, we discuss how design tools and procedures are likely to evolve towards fully integrated design systems, under the influence of increased complexity of design and predictable changes in the technologies of implementation.

To enable groups of readers to actually design moderate sized LSI systems, we've included descriptions of easily constructed LSI design tools and procedures for organizing and implementing LSI multi-project chips. In each case, the tools are described as part of a complete system of design and implementation procedures, some of which are performed manually while others are machine assisted. Those experienced in software system design will recognize that construction of the machine-assisted portions of these systems is fairly straightforward. Contrary to what many may think, designing your own LSI projects, merging them onto collaborative multi-project chips, and having these implemented by commercial maskmaking and wafer-fabrication firms is now well within the computational and financial reach of most industrial R&D groups and university EE/CS departments.

We are firm believers in *learning by doing*, and hope that the information provided in this chapter will help and encourage many groups of readers to try their hand at building LSI design tools and designing LSI systems. Such first-hand experience will lead to a deeper understanding of the remaining material in this text.

An overview of the stages of integrated system design, layout, and implementation is given in figure 1. The designer first transforms the circuit and topological level designs into a geometrical layout of the system, using procedures described later in this chapter. In order to optimize the layout, perform various design checks, and discover errors, the designer usually iterates several times between *design* and *layout*. The result is a set of *design files* describing the layout. These files are in a particular representation called an *intermediate form*, which efficiently and unambiguously describes the layout geometry.

The design files are then converted into files for driving the chosen patterning mechanism. At present, design files are commonly converted into *pattern generator* (PG) files, for use by a *maskmaking* firm for driving an optical pattern generator, the first step of maskmaking. By a sequence of photolithographic steps, the mask house produces a set of *masks*, which a commercial *wafer fabrication* firm may then use to pattern silicon *wafers*. Each finished wafer contains an array of system chips. The wafers are then diced into separate chips, which are packaged and tested to yield working systems.

From the system designer's point of view, maskmaking and fabrication can be visualized as one would a film processing service: the designer produces the "artwork" (design files), from which the mask house makes "negatives" (masks), which are then run on a fab line to produce "prints" (wafers). The maskmaking and fabrication sequence is *function, design, and layout independent*: the mask and fab firms do not require detailed information about the integrated systems they fabricate. If the original layouts satify the design rules, and satisfy a few constraints imposed by patterning and fabrication, then these processes will yield correctly patterned wafers.

One need not closely bind a system's design to the detailed processing specifications of particular mask and fab firms. Various firms will differ somewhat in the minimum value of the length unit λ which they can successfully process. The transit time of the transistors fabricated, and the resistance per square and capacitance per unit area of fabricated features will also vary from one fab line to another. However, well structured and relatively process independent nMOS designs will function correctly if scaled to a value of λ appropriate for the chosen fabrication facilities, and operated using an appropriate system clock period.

We next examine some of the present implementation procedures a bit more closely, to set the stage for sections on design and layout. Those later sections will be clearer if one can visualize how the design files are to be used during patterning and fabrication.

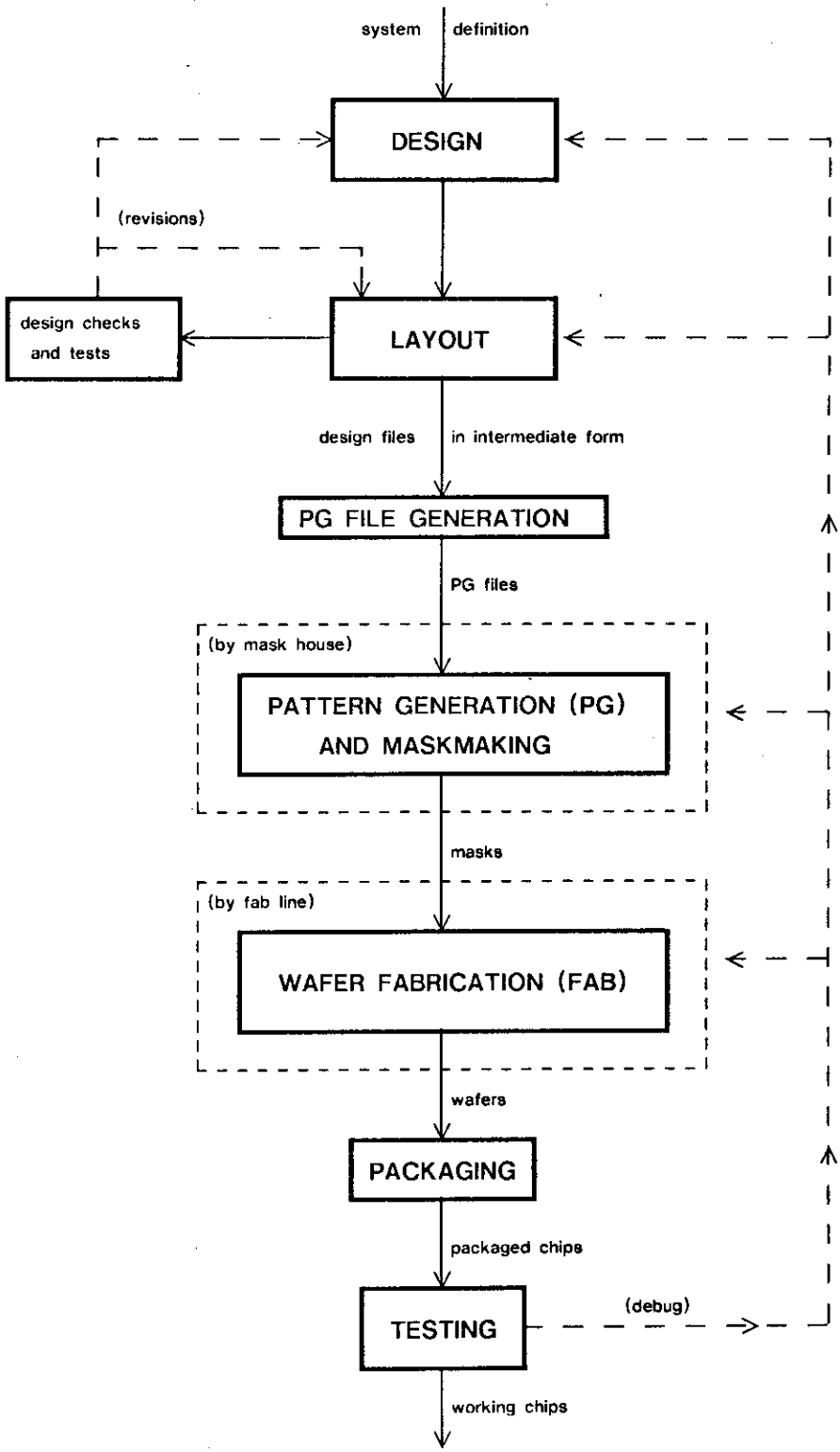


Fig. 1. Overview of Integrated System Implementation

(overview.sil)

Patterning and Fabrication

On completion of design and layout, the system design is contained in *system layout files in intermediate form.* Prior to fabrication, a final *check plot* of the layout is usually generated by converting these design files into files for driving a graphics plotter. Check plots are used for visually checking for design rule violations and other design errors. Once the designers have done as much visual checking as they are going to do, the system layout files are converted into *pattern generator* (PG) *files*, to be sent to the maskmaking facility. Figure 2 summarizes the sequence of patterning and fabrication procedures which then follows, and identifies the artifacts passed on at each step in the sequence.

Maskmaking begins with *pattern generation* to produce *reticles.* Present pattern generators are projector-like systems containing (i) a precisely movable stage, (ii) an aperture of precisely variable rectangular size and angular orientation, and (iii) a light source, all program controllable by a computer system. To produce a reticle, a photographic plate is mounted on the stage, and the PG file for a particular system layer is used to direct the flashing of a sequence of rectangular exposures, of particular sizes and orientations, onto a sequence of coordinate locations on the plate, as illustrated in figure 3.

The PG file contains a sequence of entries, each of which describes a rectangle. A typical representation uses five numbers for each rectangle: the x,y coordinates of its center, and its height, width, and angular orientation, as shown in figure 4. One can now visualize the nature of the conversion from intermediate form to PG files: the layout of each layer must be decomposed into its equivalent as a set of rectangles, each having [x,y,h,w,a] values flashable by the particular pattern generator, and these rectangles must be sorted into an efficient flashing sequence for that pattern generator.

When the flashing sequence is completed, the plate is developed, yielding the reticle. A sketch of such a reticle is given in figure 5. Each reticle is a photographic master copy much like a photo negative, of the layout of one system layer, usually at a scale ten times (10x) the final system chip size. Photo enlargements of reticles, called *"blowbacks"*, may be obtained from the mask house, to provide a further level of checking of design layout, PG file conversion, and pattern generation. At the current value of $\lambda = 3$ microns, blowbacks at approximately 100 to150 times actual chip dimensions have sufficient detail to enable visual checking of the smallest features. Blowbacks of reticles may also be obtained in the form of

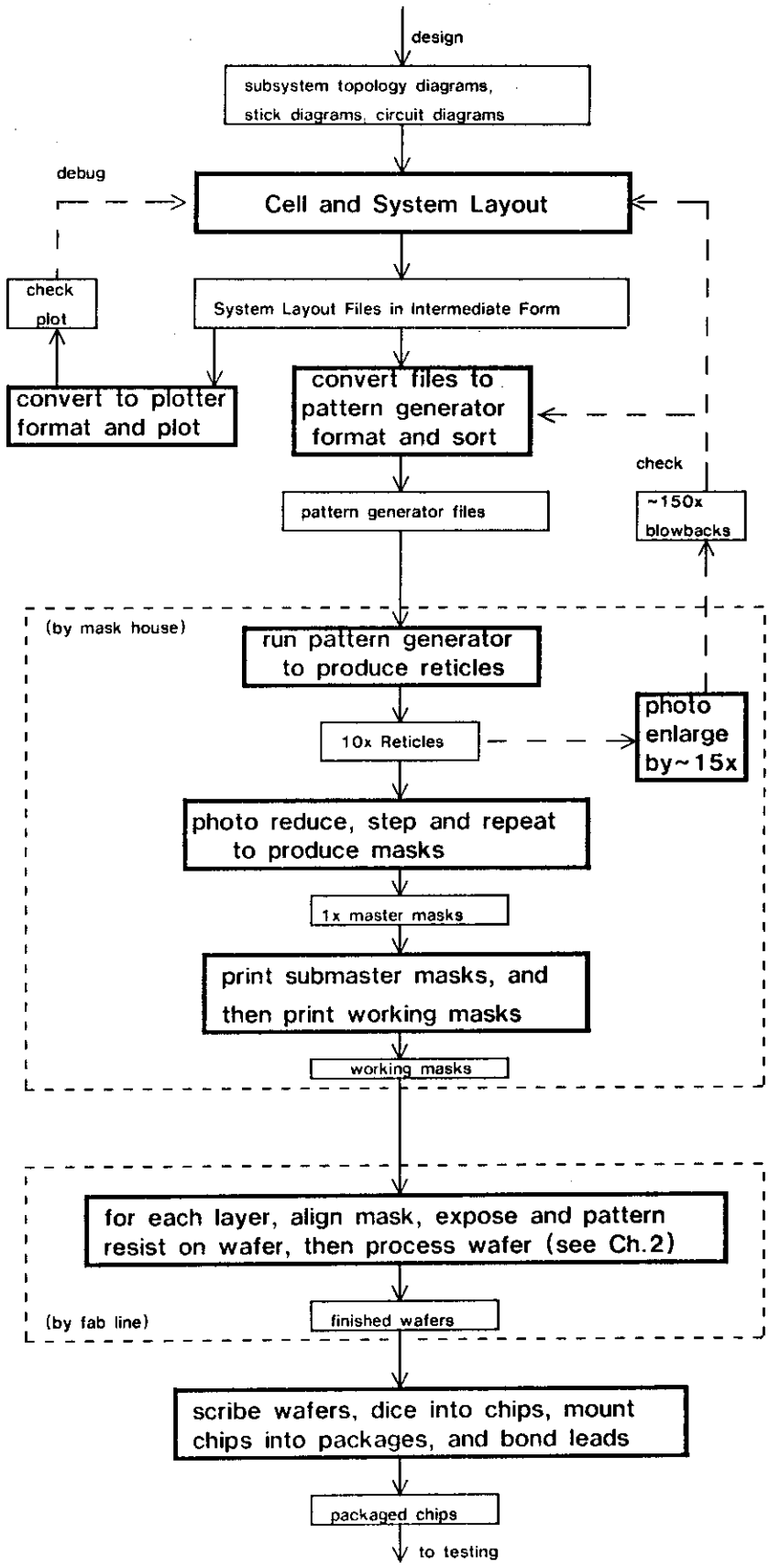


Fig. 2. The Procedures and Artifacts of the Typical Present Implementation Process

(proart.sil)

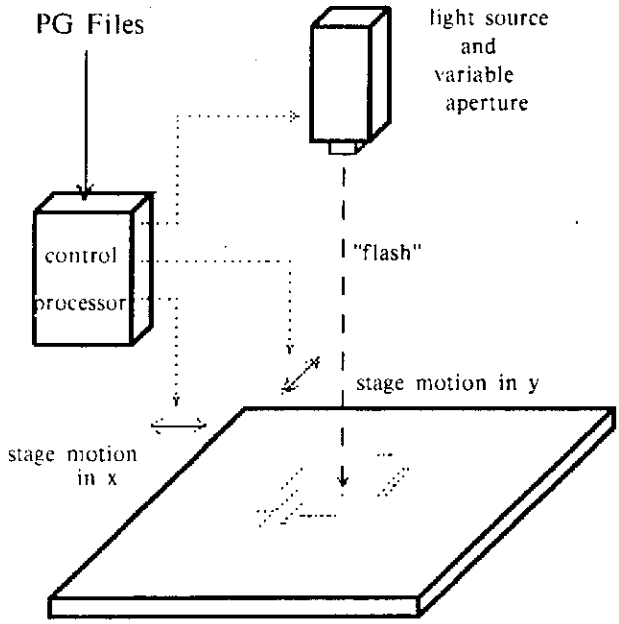


Fig. 3. Sketch Illustrating Function of Pattern Generator

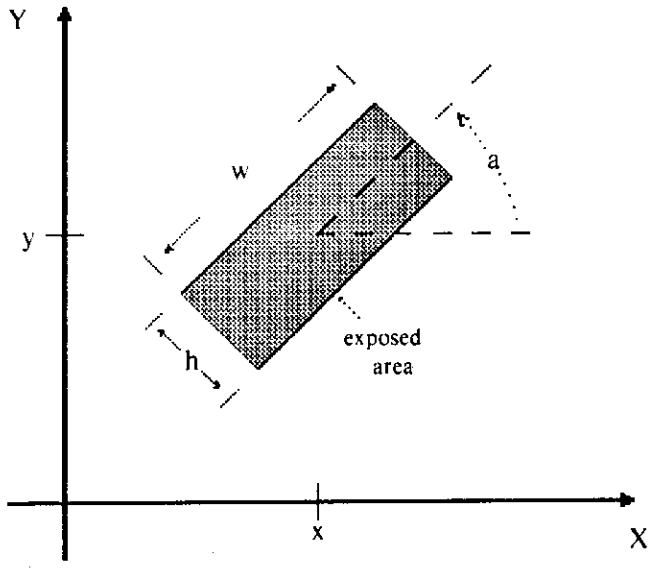


Fig. 4. Parameters of One Flash of Pattern Generator

color transparancies, to enable inspection of superposed overlays of various layers.

Once the 10x reticles have been generated, a *1x master mask* is made from each reticle using a *photorepeater*, often called a *step and repeat camera*. The photorepeater exposes a photographic plate held on a moveable stage, as in the pattern generator. In this case, however, each plate exposure is a 10:1 photo reduction of the reticle pattern. Between exposures the stage is moved by a precise x,y stepping distance. This process is repeated until a complete array of 1x chip patterns for one layer of the system has been exposed. The plate is then developed to produce a 1x master mask. Figure 6 sketches such a mask made from the reticle in figure 5.

Note that when each reticle is inserted in the photorepeater, the position and angular orientation of the reticle pattern is carefully adjusted by microscopic examination of two *fiducial marks* on the reticle. These marks are placed as part of the pattern generation process, and have the same precise position relative to the chip pattern origin on each of the system's reticles, thus assuring that all mask levels produced with the photorepeater will accurately register with each other.

A succession of contact prints is made from each master mask to yield a number of *working masks*, sometimes called working *plates*, for each system layer. These are the actual masks used in wafer fabrication. During the contact printing step of the typical wafer fabrication procedure, the working plates occasionally become worn or damaged, so several are usually made for each layer.

The wafer fabrication facility uses the working plates in the sequence of patterning and process steps described in chapter 2, to produce finished wafers. The fab line requires no detailed information about the design or mask patterns of the integrated system being fabricated. However, several auxiliary patterns are normally included in the mask patterns, some of which are replicated on each chip and are examined during wafer fabrication: (i) *alignment marks*, which are used to accurately overlay successive masks with previous patterning steps, (ii) *line width testers*, sometimes called *critical dimensions* (C/D's), which are lines in each mask layer of stated width that may be examined during maskmaking and fabrication to control dimensional tolerances, and (iii) a few simple *test transistors* and their associated probe pads, which may be electrically tested prior to packaging to verify that the wafer fabrication process was successful.

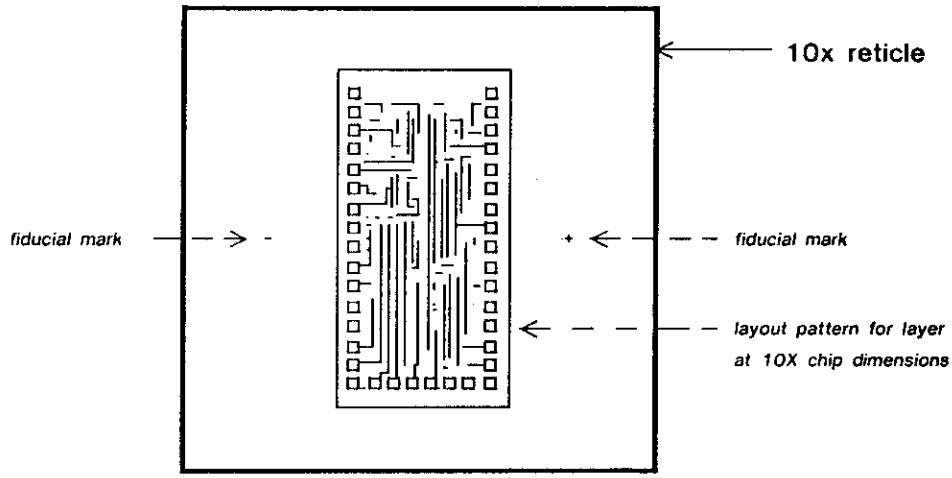


Fig. 5. Sketch of a 10x Reticle

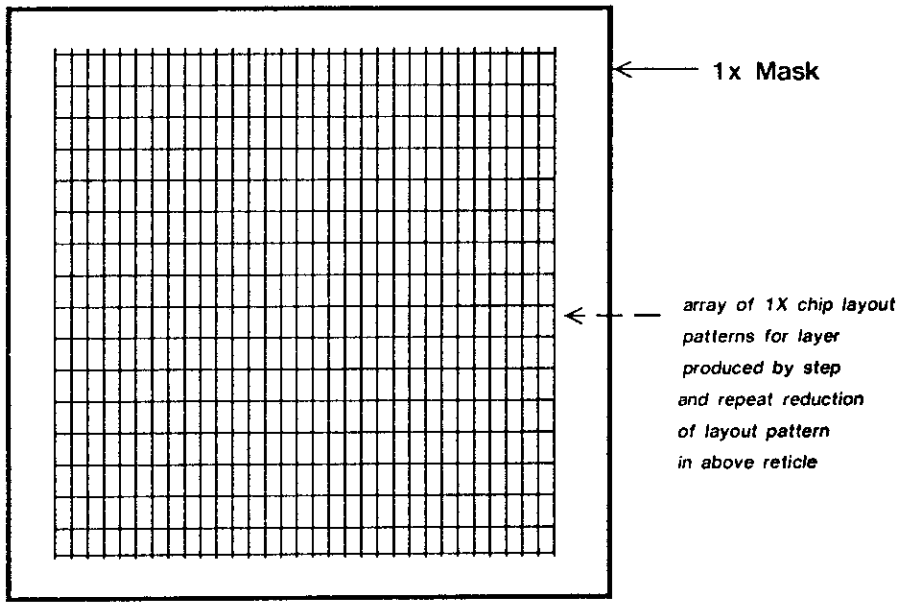


Fig. 6. Sketch of a Mask made from the above Reticle

The finished wafers are divided into chips and packaged by the sequence of steps sketched in figure 7. The wafers are diced into individual chips by first scribing their surface along the boundary lines between chips, called *scribe lines*, with either a diamond tipped scribe or a diamond edged saw blade, and then fracturing them along these lines. Each individual chip is then cemented into the cavity of a package. After fine wires are bonded between the contact pads on the chip and the leads of the package, and a cover cemented over the cavity, the system is ready for testing.

From the preceding we see that once a system's design files have been produced, all the remaining implementation procedures are design and layout independent, and largely automatic. However, the many extraneous parameters, patterns, and constraints involved in maskmaking and fabrication must be carefully thought through and defined in order to guarantee successful implementation within a reasonable turnaround time. The PG files must be correctly sorted and formatted for the pattern generator to be used. The 10x pattern of the chip must fit within the largest reticle that the pattern generator can produce. The photorepeater used will determine the shape, size, and location of the fiducial marks on the reticle. The size, surface material, and photographic polarity, either positive (*clear field-opaque features*) or negative (*dark field-transparent features*), of the working plates will be a function of the fabrication facility to be used. Each fab line also typically prescribes its own patterns for the alignment marks and test transistors to be included along with the system in the mask patterns.

While many designs may be scalable and have some longevity, the parameters, patterns, and constraints of maskmaking and fabrication are changing rapidly as the technologies evolve. This constant change complicates interactions with mask and fab firms. Later we describe procedures for implementating moderate sized LSI systems as part of multi-project chips. Such chips are collaborative efforts of many designers, enabling many projects to be merged into one maskmaking and wafer fabrication run. In this way the procedural overhead involved may be shared.

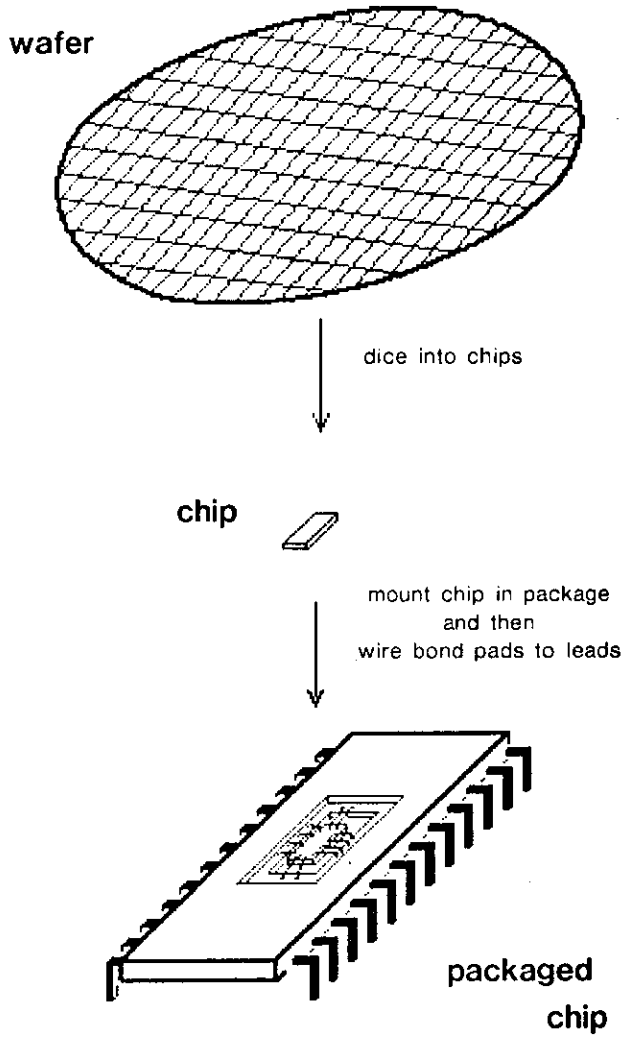


Fig. 7. Sketch Illustrating the Packaging Sequence

(wafer.press)

Hand Layout and Digitization using a Symbolic Layout Language

A simple and common method of producing system layouts is to draw them by hand. This is typically done on a one lambda grid using the familiar color codes to identify various system layers. Once the layout has been hand drawn it can then be *digitized*, or translated into machine readable form, by encoding it into a symbolic layout language. Hand layout and digitization using a symbolic layout language is quite a practical method of generating design files for highly structured system designs. Be warned, however, that implementing irregular structures using these primitive procedures is a difficult and tedious task.

If a system has only a few cell types which are replicated over and over, and otherwise has little "random wiring", one need draw only a single copy of each cell type, and then make reproductions or equivalent sized outlines of these cell drawings. All these cell reproductions may then be patched together to plan and build up the overall layout. Similarly, only one symbolic digitization need be made for each cell type. The replication of cells in various orientations and locations in the system layout can then be easily described using the symbolic layout language. In a sense, the ease with which a system's layout can be described using a primitive layout languge provides a measure of the regularity of its design. The OM2 Data Chip pictured in the frontispiece was laid out and digitized in this way, using only the simplest machine aids.

The function of a symbolic layout language, in its simplest form, is similar to that of a macro-assembler. The user defines *symbols* (macros) which describe the layout of basic system cells. The locations and orientations of instances of these symbols are described in the language, as a function of appropriate parameters. These symbolic descriptions may then be mechanically processed in a manner similar to the expansion of a macro assembly language program, to yield the intermediate form description of the system layout, which is analogous to machine code for generating output files. An example intermediate form is described in a later section. The intermediate form files may be processed to yield the PG files: each layer being a machine encoded collection of rectangles encoded as [x,y,h,w,a] values. The generation of PG files is analogous to the loading and execution of machine code to produce output files: it is a process of "unrolling" and fully instantiating all symbol descriptions into a sequence and format suitable for a particular output device. Definition of simple layout languages and the construction of their assemblers is fairly straightforward. The reader may define and implement layout languages by using the macro assembler or higher level language facilities of any commonly available computer system (R1, R3).

The following example will clarify the concepts and procedures of hand layout and symbolic layout description: We wish to create an array of shift registers consisting of parallel horizontal rows of inverters coupled by clocked pass transistors, as in figure 5a., chapter 2. Figure 8a sketches the stick diagram of one row of the array. The entire array can be constructed from one basic cell containing an inverter, the pass transistor following it, VDD and GND buses crossing through on metal, and a clock line passing through on poly. Figure 8b shows a hand sketch of the layout of the basic shift register cell, SRCELL, on a 1λ grid, subject to the design rules given in Ch.2, Sect.2. Since the inverters are coupled by pass transistors, the inverter pullup/pulldown ratio is ~ 8:1 (see Ch.1., Sect.2.). Also, while the 4λ wide metal lines could be 1λ narrower in between the contact regions, this would not decrease the cell size. As an exercise, the reader might check for design rule violations, and also for ways of further shrinking the cell size.

The SRCELL layout shown in figure 8b is composed using only rectangles placed at orientations which are integer multiples of 90°. The illustrations and descriptions in this section are considerably simplified by the use of such constrained layout constructions, and yet still illustrate the general principles involved. Were completely arbitrary shapes used, the SRCELL could be made somewhat smaller and still satisfy the design rules. Interestingly, experience has shown that the simple extension of including rectangles at orientations which are integer multiples of 45° enables most cell layouts to reach within a few percent of the minimum area achievable using arbitrary shapes. There is a clear tradeoff here: the inclusion of increasingly complex geometrical objects in a layout will tend to reduce the minimum achievable layout area, but will also increase the computational complexity of the associated machine aids.

We can informally characterize a simple layout language by examining figure 9, which contains a description of the layout of an array of SRCELLs using such a language. The language describes layouts as collections of BOXes on various layers. BOX statements describe each of these boxes by specifying their layer, the X,Y coordinates of their lower left corner and then their length, LX, in the x direction, and LY, in the y direction. The use of a box corner to encode its location simplifies the encoding task. BOX statements may describe arrays of identical boxes, with the array's lower left corner origin at X,Y, by including optional parameters which specify the number NX and replication interval IX in the x direction, and NY and IY in the y direction. Dimensions are given in the length unit, λ. A SCALE statement defines the value of λ for this particular layout as λ = 3.0 microns.

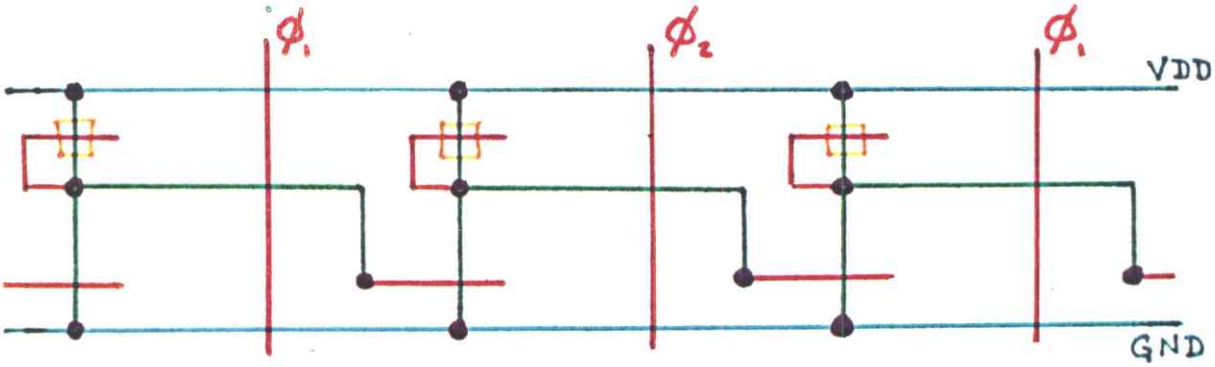


Fig. 8a. Stick Diagram of One Row of Shift Register Array

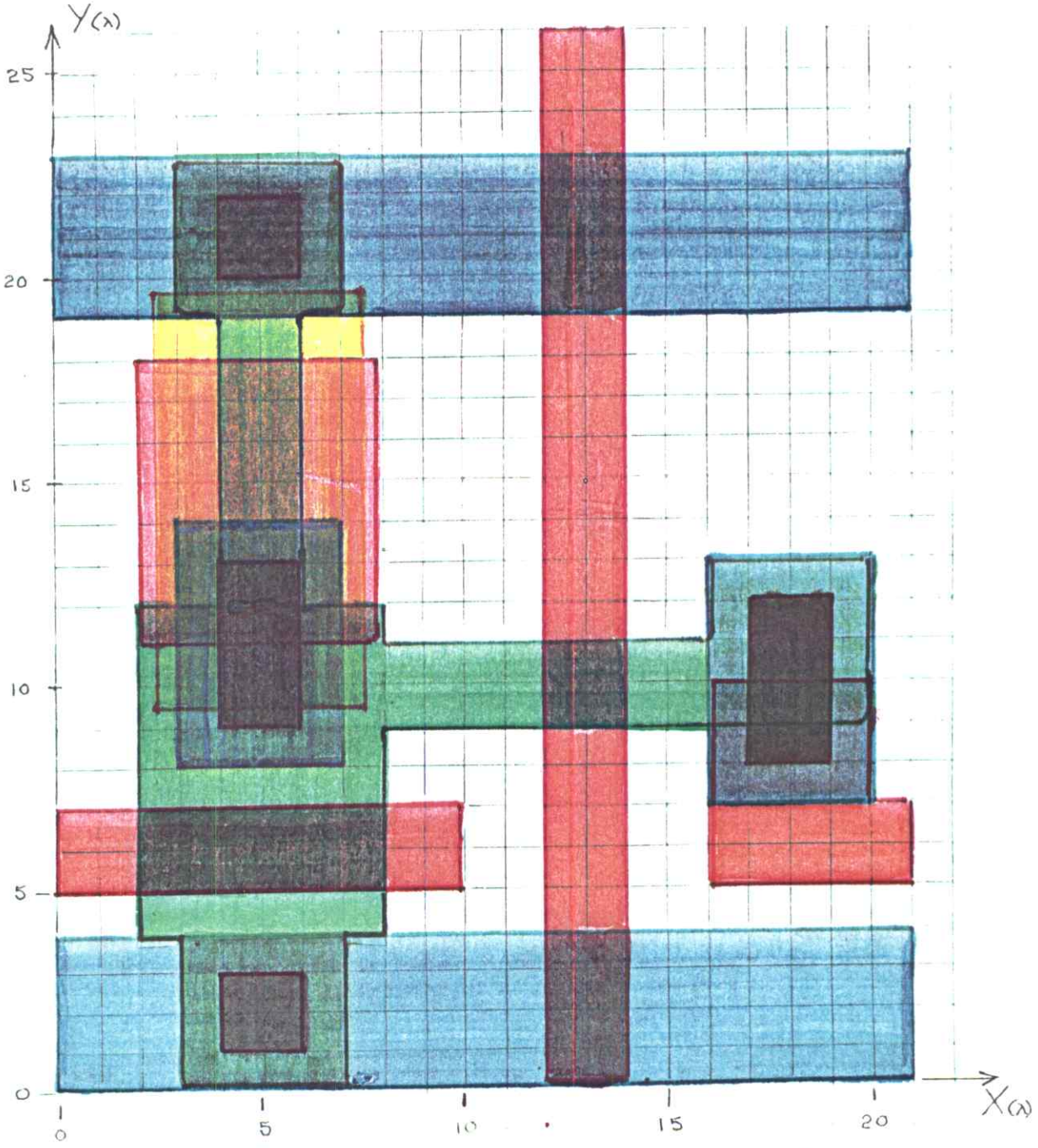


Fig. 8b. Hand Sketch of Layout of One Shift Register Cell

In figure 9, the SRCELL is first described as a macro, or SYMBOL. The reader can verify that the collection of BOXes in the definition of the SYMBOL SRCELL, when ORed together, produces the layout in figure 8b. This SRCELL is then replicated a number of times in various layout locations according to parameters in several DRAW statements.

Each DRAW statement describes the placement of an array of cells as follows: The cell described by the named SYMBOL definition is considered to be drawn at the origin. It is then *mirrored* (about the x and/or y axis), and/or *rotated* (by 0°, 90°, 180°, or 270°) about the origin, as specified by MIRROR or ANGLE transformations. The cell thus positioned may then be *replicated* NX times at distance intervals IX in the x direction, and that row of cells then be *replicated* NY times at intervals IY in the y direction. The resulting array of cells is then *translated* a distance X,Y from the origin, and placed into the layout.

```
SCALE    LAMBDA=3.0MICRON;
;
SYMBOL   START, SRCELL;
  BOX    DIFF,X=3,Y=0,LX=4,LY=4,NY=2,IY=19;
  BOX    DIFF,X=2,Y=4,LX=6,LY=8;
  BOX    DIFF,X=8,Y=9,LX=8,LY=2;              INVERTER  OUTPUT
  BOX    DIFF,X=16,Y=9,LX=4,LY=4;
  BOX    DIFF,X=4,Y=12,LX=2,LY=7;
  BOX    IMPL,X=2.5,Y=9.5,LX=5,LY=10;         PULLUP  IMPLANT
  BOX    POLY,X=0,Y=5,LX=10,LY=2;             CELL  INPUT
  BOX    POLY,X=12,Y=0,LX=2,LY=26;            CLOCKLINE
  BOX    POLY,X=16,Y=5,LX=5,LY=4;             CELL  OUTPUT
  BOX    POLY,X=16,Y=7,LX=4,LY=3;
  BOX    POLY,X=2,Y=11,LX=6,LY=7;
  BOX    CUTS,X=4,Y=1,LX=2,LY=2,NY=2,IY=19;
  BOX    CUTS,X=17,Y=8,LX=2,LY=4;
  BOX    CUTS,X=4,Y=9,LX=2,LY=4;
  BOX    METL,X=0,Y=0,LX=21,LY=4,NY=2,IY=19;  VDD  &  GND
  BOX    METL,X=3,Y=8,LX=4,LY=6;
  BOX    METL,X=16,Y=7,LX=4,LY=6;
SYMBOL   END;
;
  DRAW   SRCELL,NX=4,NY=2,IX=21,IY=38,X=0,Y=0;
  DRAW   SRCELL,MIRRORX,NX=4,IX=21,X=0,Y=42;
;
END;
```

Figure 9. Symbolic Description of Shift Register Array

The "program" in figure 9 describes an array of 3 rows and 4 columns of SRCELLs. After machine assembly of this program, the resulting design file can be used to generate check plots, which may be inspected to detect errors made in encoding the layout. A check plot of

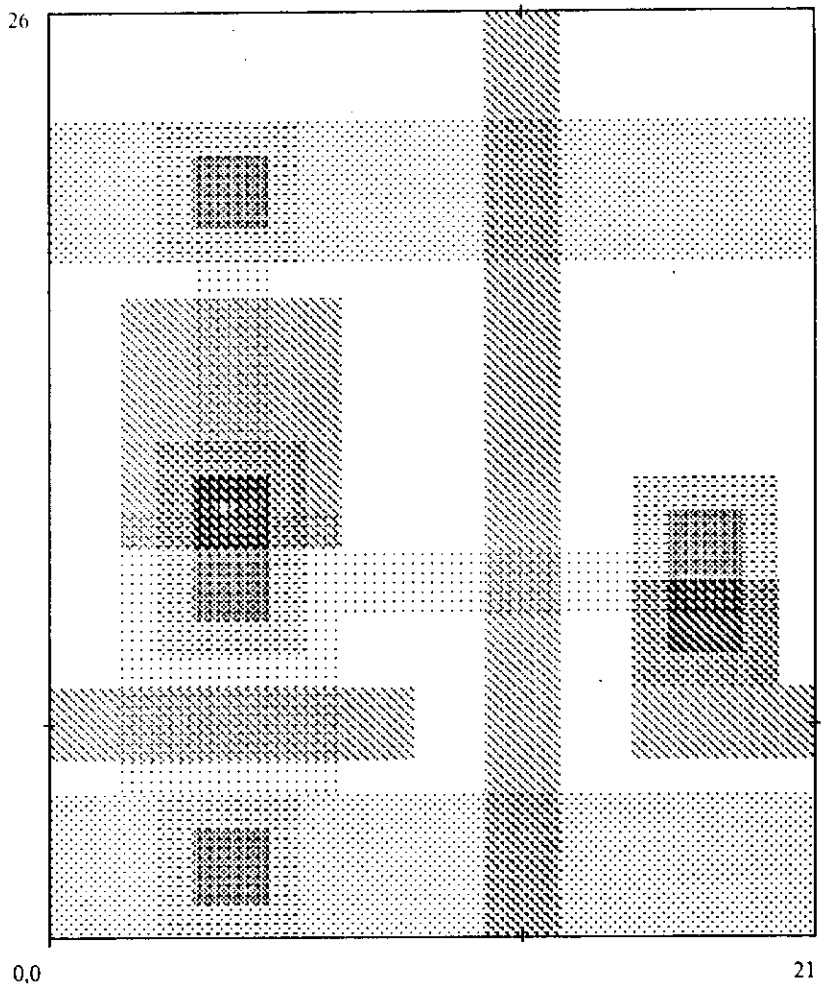


Figure 10a. Check Plot of the SRCELL

[Dimensions in lambda. Implant layer not shown]

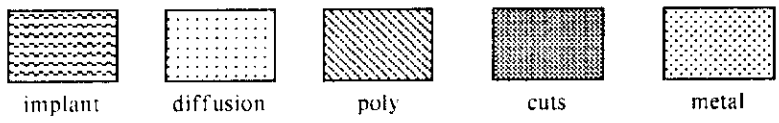


Figure 10b. Check Plot Stipple Codes

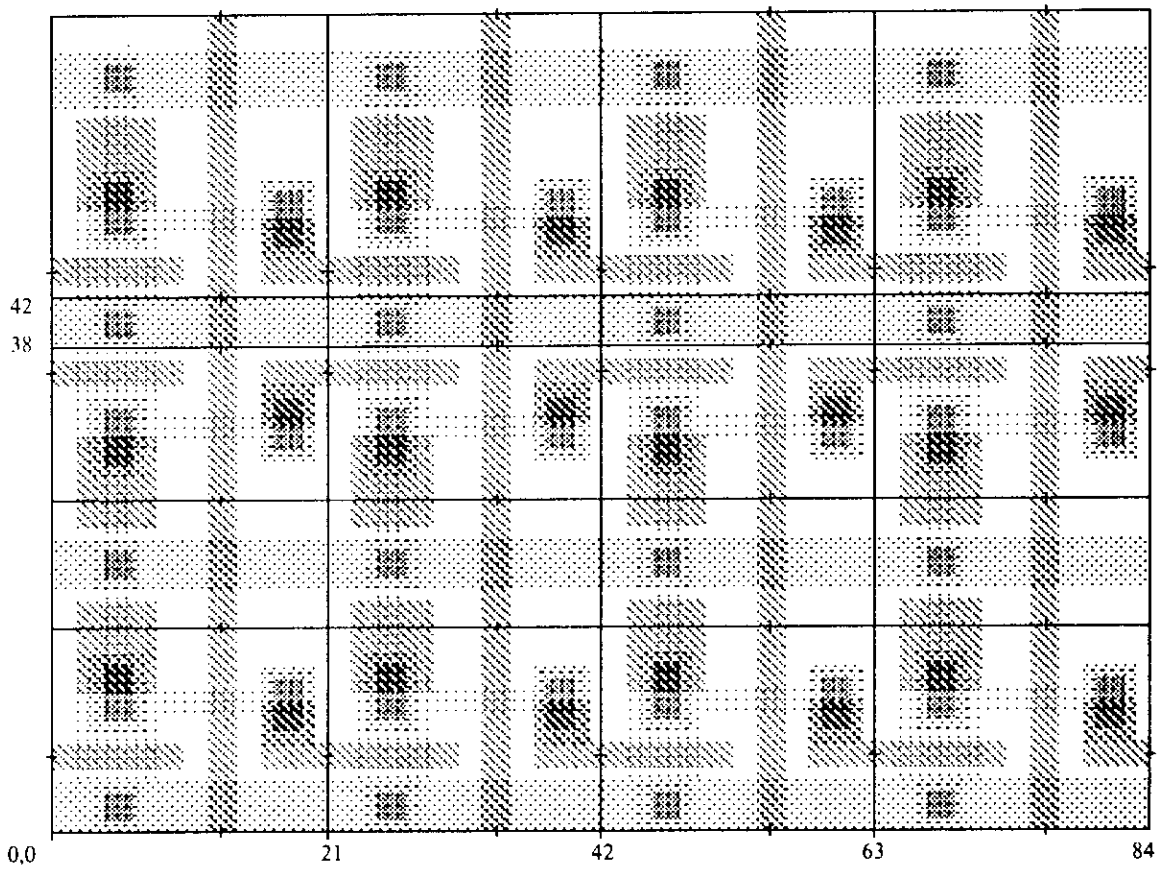


Figure 11. Check Plot of 3 by 4 Array of SRCELLS

[Dimensions in lambda, with the cell outlines indicating relative cell placements according to the program in fig.9]

one SRCELL is given in figure 10a, and we see that the cell has been correctly digitized. A set of stipple patterns is used in this check plot to encode the different system layers, with the coding specified in figure 10b. If available, color checkplots are much better: color checkplots can be made denser and still be readable, and association of colors with layers and functions is more easily made and subject to fewer errors in practice. Note: the implant layer hasn't been plotted in fig. 10a, so that the other layers may be more easily seen.

A check plot of the complete 3 by 4 array of cells is given in figure 11 (again the implant layer is not plotted). Although figure 11 is of insufficient scale to check details within the cells, it enables us to check for correct relative placement of the SRCELLs. The individual cell outlines are included in figure 11, to indicate the nature of the placement of the central row of the array. By mirroring the central row prior to its placement, that row is able to share VDD and GND with the other two rows, thus reducing the overall array size. There is one column of cells per 21 lambda in the x-direction, and one row of cells per 19 lambda in the y-direction. It is *very important* to note that the *outcome of each DRAW statement is determined by the order in which any mirror, rotate, replicate, and translate operations occur* (see the section on the Caltech Intermediate Form, and also R2, Ch.6). Any permutation in the order of these operations may lead to a completely different result.

In chapter 3 we found that the PLA is a useful subsystem structure, often used to implement finite state machines and combinatorial logic. We now present a worked out example of a PLA's layout, to further clarify symbolic layout description. Chapter 3 contains several stick diagrams of PLAs (figs. 13c, 15f). An examination of these stick diagrams reveals that the PLA can be constructed using 6 basic cell types and a slight amount of "random wiring". Once these 6 basic cells have been layed out by hand and symbolically digitized, it is easy to construct symbolic descriptions of different sized PLAs having various numbers of inputs, product-terms, and outputs.

The digitized layouts of four of these basic cells are check plotted in figure 12. The AND and OR planes of the PLA are constructed as arrays of the 14λ by 14λ PLAcellpair cell plotted in figure 12a, which contains two poly and two metal signal lines, and one ground line on the diffusion layer. Diffusion paths may be added in any of four locations in such cells to form transistors, and thus program the PLA. The connection between the AND and OR planes is made using the PLAconnect cell plotted in figure 12b: these cells change the signal paths from the metal to the poly layer. The pullup transistors to be placed at the edges of the AND and OR planes are implemented by the PullupPair cell in figure 12c. The

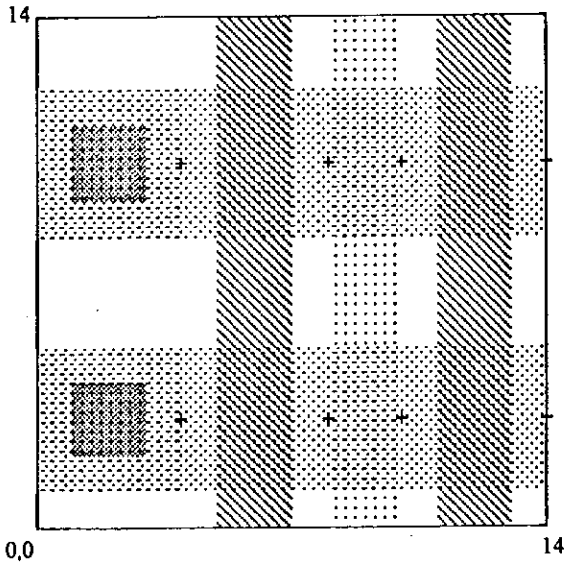


Fig. 12a. PLAcellpair

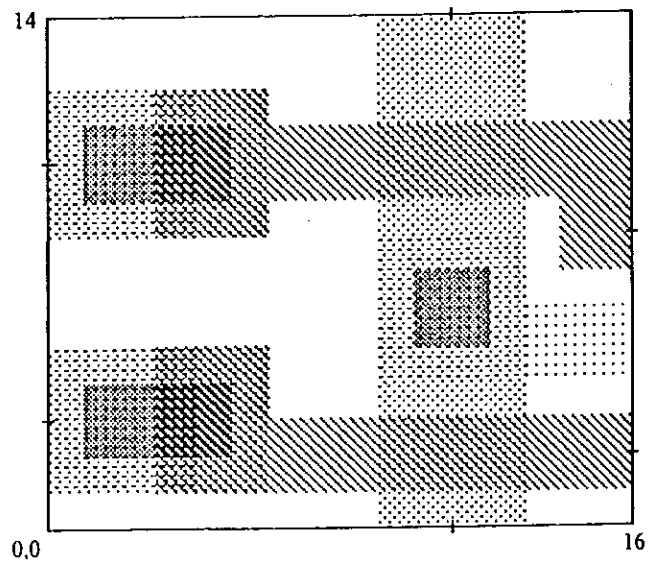


Fig. 12b. PLAconnect

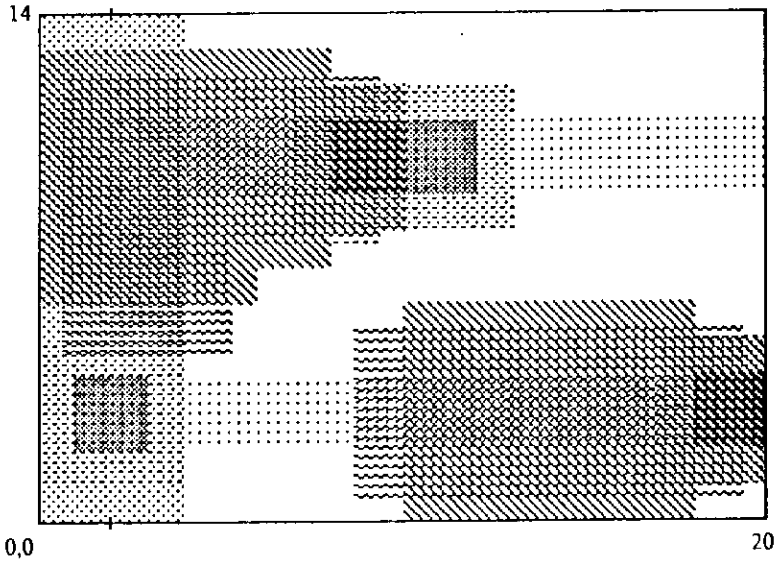


Fig. 12c. PullupPair

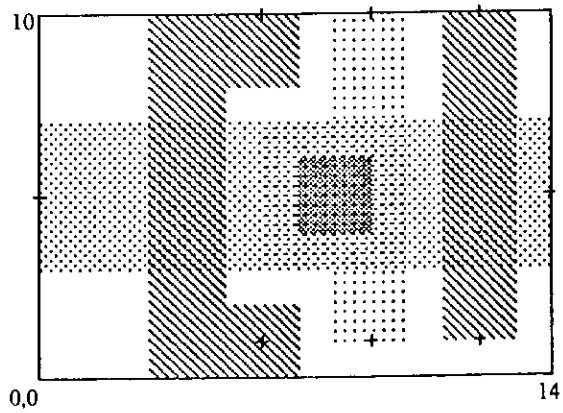


Fig. 12d. PLAground

[all dimensions in lambda]

(placells.press)

ground return paths, to be connected to the diffusion lines crossing the planes, are implemented by the PLAground cell in figure 12d. The PLAground cell is structured so that rows of the cell may be inserted at intervals within AND planes, and columns of the cell inserted at intervals within OR planes, to provide proper ground returns in large PLA's. The two other cell types required are the input drivers and output inverters: these cell layouts are left as exercises for the reader. The cells in figure 12 have been collectively planned so as to fit on a 14λ pitch surrounding the PLA's planes. Figure 13 contains a symbolic description of each of these cell types, and a description of a moderate sized PLA constructed from these cells:

Figure 13. Symbolic Description of a 5-Input, 10-Pterm, 8-Output PLA

```
SCALE    LAMBDA=3.0MICRON;
;
;        PLA CELL DEFINITIONS:
;
SYMBOL   START,PLACELLPAIR;                    [SEE FIGURE 12A.]
  BOX    DIFF,X=0,Y=1,LX=4,LY=4,NY=2,IY=7;
  BOX    DIFF,X=8,Y=0,LX=2,LY=14;              DIFF TO GND
  BOX    POLY,X=5,Y=0,LX=2,LY=14,NX=2,IX=6;
  BOX    CUTS,X=1,Y=2,LX=2,LY=2,NY=2,IY=7;
  BOX    METL,X=0,Y=1,LX=14,LY=4,NY=2,IY=7;    METL TO PULLUPS
SYMBOL   END;
;
SYMBOL   START,PLACONNECT;                     [SEE FIGURE 12B.]
  BOX    DIFF,X=0,Y=1,LX=4,LY=4,NY=2,IY=7;
  BOX    DIFF,X=9,Y=4,LX=4,LY=4;
  BOX    DIFF,X=13,Y=4,LX=3,LY=2;
  BOX    POLY,X=6,Y=1,LX=10,LY=2,NY=2,IY=8;
  BOX    POLY,X=3,Y=1,LX=3,LY=4,NY=2,IY=7;
  BOX    POLY,X=14,Y=7,LX=2,LY=2;
  BOX    CUTS,X=1,Y=2,LX=4,LY=2,NY=2,IY=7;
  BOX    CUTS,X=10,Y=5,LX=2,LY=2;
  BOX    METL,X=9,Y=0,LX=4,LY=14;              GND
  BOX    METL,X=0,Y=1,LX=6,LY=4,NY=2,IY=7;
SYMBOL   END;
;
SYMBOL   START,PULLUPPAIR;                     [SEE FIGURE 12C.]
  BOX    IMPL,X=8.5,Y=0.5,LX=11,LY=5;
  BOX    IMPL,X=0.5,Y=4.5,LX=5,LY=8;
  BOX    IMPL,X=0.5,Y=7.5,LX=9,LY=5;
  BOX    DIFF,X=0,Y=1,LX=4,LY=4;
  BOX    DIFF,X=4,Y=2,LX=16,LY=2;
  BOX    DIFF,X=2,Y=5,LX=2,LY=4;
  BOX    DIFF,X=2,Y=9,LX=18,LY=2;
  BOX    DIFF,X=9,Y=8,LX=4,LY=4;
  BOX    POLY,X=10,Y=0,LX=8,LY=6;
  BOX    POLY,X=18,Y=1,LX=2,LY=4;
  BOX    POLY,X=8,Y=8,LX=2,LY=4;
  BOX    POLY,X=0,Y=7,LX=8,LY=6;
  BOX    POLY,X=0,Y=6,LX=6,LY=1;
  BOX    CUTS,X=1,Y=2,LX=2,LY=2,NX=2,IX=17;
  BOX    CUTS,X=8,Y=9,LX=4,LY=2;
  BOX    METL,X=0,Y=0,LX=4,LY=14;              VDD
  BOX    METL,X=7,Y=8,LX=6,LY=4;
  BOX    METL,X=17,Y=1,LX=4,LY=4;
SYMBOL   END;
```

```
;
SYMBOL   START,PLAGROUND;                          [SEE FIGURE 12D.]
  BOX    DIFF,X=8,Y=1,LX=2,LY=9;
  BOX    DIFF,X=6,Y=3,LX=4,LY=4;
  BOX    POLY,X=3,Y=0,LX=2,LY=10;
  BOX    POLY,X=5,Y=0,LX=2,LY=2NY=2,IY=8;
  BOX    POLY,X=11,Y=1,LX=2,LY=9;
  BOX    CUTS,X=7,Y=4,LX=2,LY=2;
  BOX    METL,X=0,Y=3,LX=14,LY=4;                  GND
SYMBOL   END;
;
SYMBOL   START,PLAINPUT;
;
[ insert symbol definition; size: 14 wide by ~35 high ]
;
SYMBOL   END;
;
SYMBOL   START,PLAOUTPUT;
;
[ insert symbol definition; size: 14 wide by ~41 high ]
;
SYMBOL   END;
;
;          LAYOUT 5-INPUT,10-PTERM,8-OUTPUT PLA:
;                 [SEE FIGURE 14]
;
  DRAW   PLACELLPAIR,NX=5,NY=5,IX=14,IY=14,X=0,Y=0;
  DRAW   PLACONNECT,NY=5,IY=14,X=70,Y=0;
  DRAW   PULLUPPAIR,NY=5,IY=14,X=-19,Y=0;
  DRAW   PLAGROUND,NX=5,NY=2,IX=14,IY=79,X=0,Y=-10;
  DRAW   PLACELLPAIR,ANGLE=270,NX=4,NY=5,IX=14,IY=14,X=86,Y=14;
  DRAW   PULLUPPAIR,ANGLE=270,NX=4,IX=14,X=86,Y=89;
  DRAW   PLAGROUND,ANGLE=270,NY=5,IY=14,X=141,Y=14;
  DRAW   PLAINPUT,NX=5,IX=14,X=0,Y=-44;
  DRAW   PLAOUTPUT,NX=5,IX=14,X=86,Y=-41;
  BOX    DIFF,X=70,Y=-15,LX=4,LY=4;
  BOX    CUTS,X=71,Y=-14,LX=2,LY=2;
  BOX    METL,X=70,Y=-15,LX=4,LY=4;
  BOX    METL,X=-19,Y=70,LX=4,LY=9;                VDD
  BOX    METL,X=-19,Y=79LX=105,LY=4;               VDD
  BOX    METL,X=82,Y=83,LX=4,LY=6;                 VDD
  BOX    METL,X=142,Y=85LX=9,LY=4;                 VDD
  BOX    METL,X=151,Y=-40,LX=4,LY=129;             VDD
  BOX    METL,X=142,Y=-40,LX=9,LY=4;               VDD
  BOX    METL,X=-19,Y=-15,LX=19,LY=4;              VDD
  BOX    METL,X=-19,Y=-11,LX=4,LY=11;              VDD
  BOX    METL,X=70,Y=71,LX=9,LY=4;                 GND
  BOX    METL,X=70,Y=-7,LX=9,LY=4;                 GND
  BOX    METL,X=70,Y=-24,LX=9,LY=4;                GND
  BOX    METL,X=79,Y=-45,LX=4,LY=45;               GND
  BOX    METL,X=83,Y=-21,LX=3,LY=4;                GND
  BOX    METL,X=142,Y=-21,LX=2,LY=4;               GND
  BOX    METL,X=144,Y=-21,LX=4,LY=21;              GND
  BOX    POLY,X=-4,Y=-43,LX=4,LY=2;                PH1
  BOX    POLY,X=142,Y=-7,LX=15,LY=2;               PH2
;
[ insert the PLA "program", using BOXes on the diffusion
        layer to form transistors in the PLAcellpair cells ]
;
[ insert the PLA's input, output, clock, and power connections ]
;
END;
```

A check plot of the PLA described above is given in figure 14. This check plot has been simplified to include only the outlines of the basic cells, plus the additional wiring necessary to complete the PLA. The dimensions and orientations of the cells may be found by comparing these outlines with the cell details in figure 12. Note that in figure 12 some of

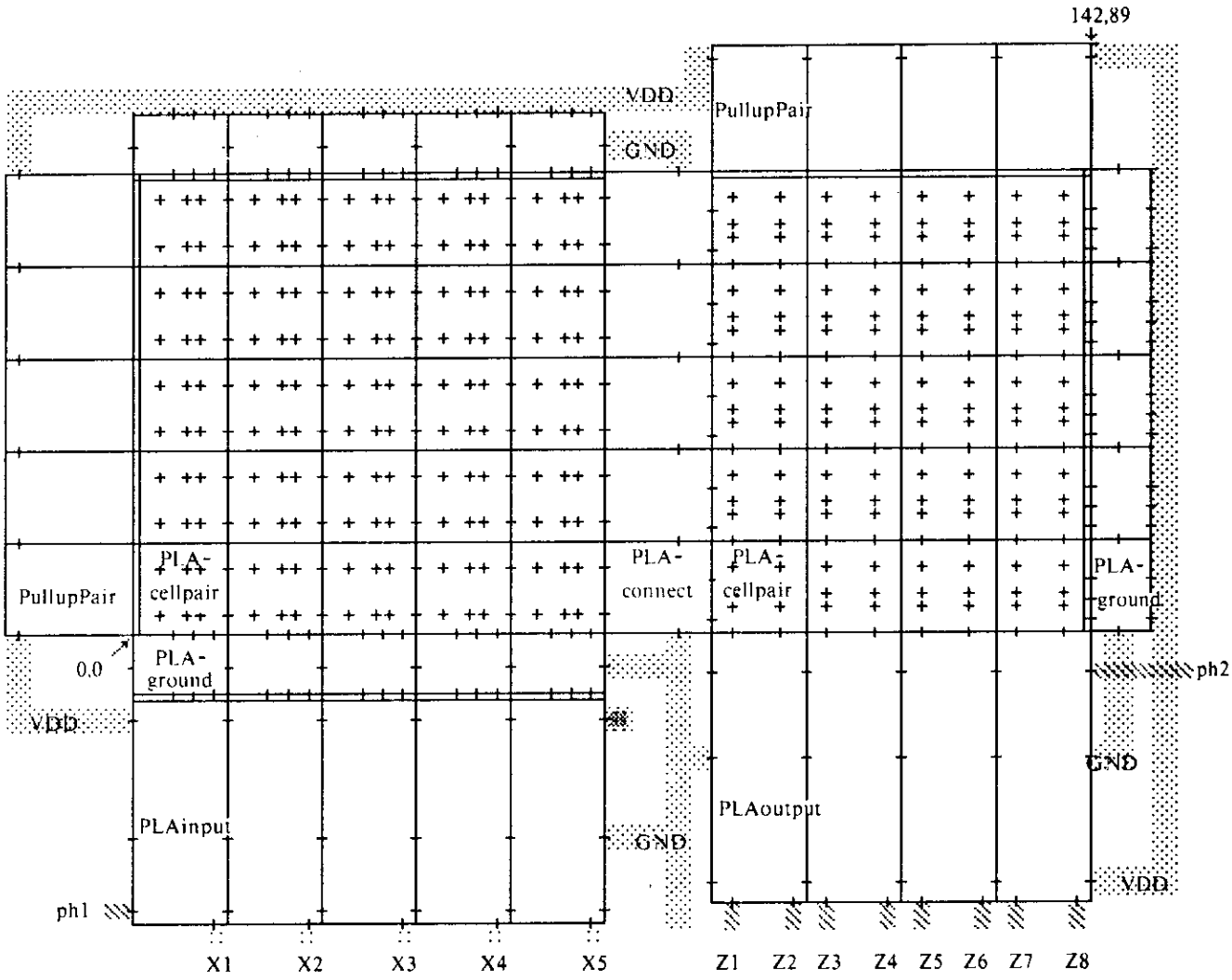


Fig. 14. Check Plot using only Cell Outlines, of the 5-Input, 10-Pterm, 8-Output PLA

[dimensions in lambdas; symbol labels on origin cells of the fig.13 DRAW statements]

the *connection points*, where paths leave or enter at cell edges or where internal connections may be later inserted, are tagged with tick marks. Cell placements and orientations in the check plot may be visualized by locating and identifying the appropriate connection point marks. A comparison of the check plot with the symbolic description above will clarify the function of the various DRAW statements. To assist in this comparison, the origin cell of the array of cells produced by each DRAW statement has been marked in figure 14 with its cell name. Note that this PLA layout could contain the PLA example of chapter 3, fig. 15f.

Symbolic layout languages are easy to define, and may be primitive or sophisticated, according to the requirements of the user. The function of the assembler for such a language is simply to scan and decode the statements and translate them into design files in intermediate form. Conversion of design files into check plot or pattern generator output files is straightforward for the above simple language, since we have used only boxes with a severe constraint on angular orientations. MIRROR and ANGLE transformations are easily handled: x and y coordinates of symbols and boxes are simply replaced by $\pm x$ or $\pm y$, according to the specific parameters, during the instantiation of symbols and drawing of boxes prior to their replication and translation into the layout output file.

The effectiveness of the above language could be further increased by constructing an assembler capable of handling nested sysmbols. By using nested symbols, system layouts may be described in a hierarchical manner, leading to very compact descriptions of structured designs. At the lowest level, one might define symbols for such small but commonly encountered structures as the various forms of contacts. Boxes and these simple symbols could then be used to construct cells such as those in the PLA example above. The PLA could be constructed with these cells, and then defined as a symbol to be used in a larger design. An example of the sort of function one might add to create a much more sophisticated language, and language processor, would be the capability of generating the layout description of a PLA from the collection of basic cells, as a function of its input, pterm, and output size parameters and logic function parameters.

Figure 15 summarizes the procedures and artifacts of hand layout, and layout description and digitization using a layout langauge. By studying figure 15, and thinking back over the material and examples of this section, one can visualize a complete, though primitive, sequence of steps sufficient to prepare a design for implementation. These procedures are entirely adequate for preparing small LSI projects for implementation. The procedures may also be used for those large LSI systems which have highly structured designs.

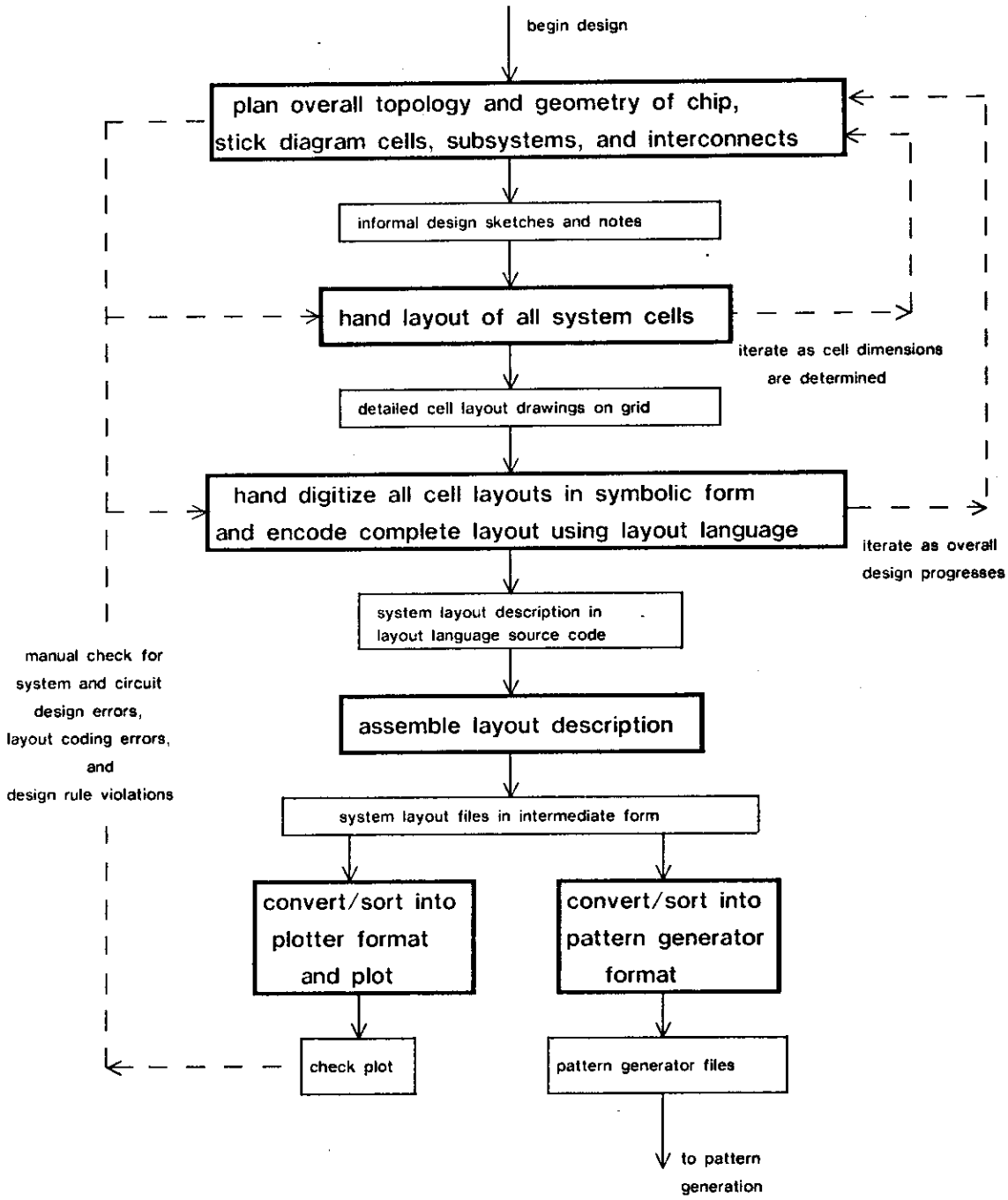


Fig.15. Design, Hand Layout, Design File Generation, And Design Checking Using a Symbolic Layout Language and Simple Machine Aids

The primary obstacle that these primitive procedures place in the path of the system designer is the sheer time and effort it takes to get through the loop to a new check plot each time a small design change is made. The enthusiasm aroused by a sudden insight, such as the conception of a completely new topological possibility for an important system cell, can be dampened by the tedious tasks of hand layout and box digitization required before one can really see the full effect of the idea on the overall system layout.

Though often supported by large batch mode CAD systems for containing, modifying, checkplotting, and simulating designs, the majority of LSI layout now done in industry begins with hand layout. Digitization is usually simplified by the use of digitizing tables, which are much like graphics plotters in reverse: a new section of a design, laid out by hand, is placed on the table and digitized by tapping switches while manually following the outlines of the cell's boxes with a pointer. Although this is less tedious than digitization using a layout language, it is still time-consuming and hardly interactive.

The next section describes an interactive graphics layout system which enables the system designer to quickly sketch new layout ideas and see their effect immediately.

An Interactive Layout System

[Section contributed by Douglas Fairbairn, Xerox PARC, and James Rowson, Caltech]

Computing hardware of sufficient power to support highly interactive graphics has in the past quite expensive, and this has inhibited the widespread application of interactive computing techniques. However, because of expected advances in VLSI technology, we are rapidly approaching the day when many will have access to personal computers with computing power rivaling today's medium to large-scale systems. It will be more difficult to provide effective software for these systems than it will be to build the computers themselves.[3] In this section we describe a highly interactive layout system which runs on a modest personal computer, rather than on an expensive, limited access, centralized system. This system was developed anticipating the work environment of the future, in which most "knowledge" workers will have personal computers as part of their normal office equipment.

ICARUS[1] (Integrated Circuit ARtwork Utility System) is a software system which enables the user to create and modify an integrated system layout directly on a CRT display screen. ICARUS was conceived with the idea that the designer would create and edit a layout at the display, without doing any more than a rough sketch or "stick diagram" before beginning work. Creating and moving items is fast and easy enough so that the designer can truly sketch on the screen. Once the layout is basically correct, the items can be moved or modified to arrive at the most compact layout.

The user is required to remember very little about the available commands or their use because the commands themselves are displayed on the screen and the system prompts the user for additional information as it is needed. The system can format and output check plots to matrix type printers or on raster-scan laser printers. ICARUS design files can be used to create standard pattern generation files from which masks can be made. An overview of design and layout procedures using the system is given in figure 16. It is instructive to compare this with figure 15, which presents equivalent steps for hand layout.

All the software to accomplish these various steps runs on a small experimental minicomputer known as the Alto. This machine was designed by researchers at Xerox PARC as a general purpose personal computer suitable for both text and graphics applications[3]. No additional, special hardware is used by ICARUS. The ICARUS system is programmed in BCPL, an ALGOL-like high level language. There are about 30K words of

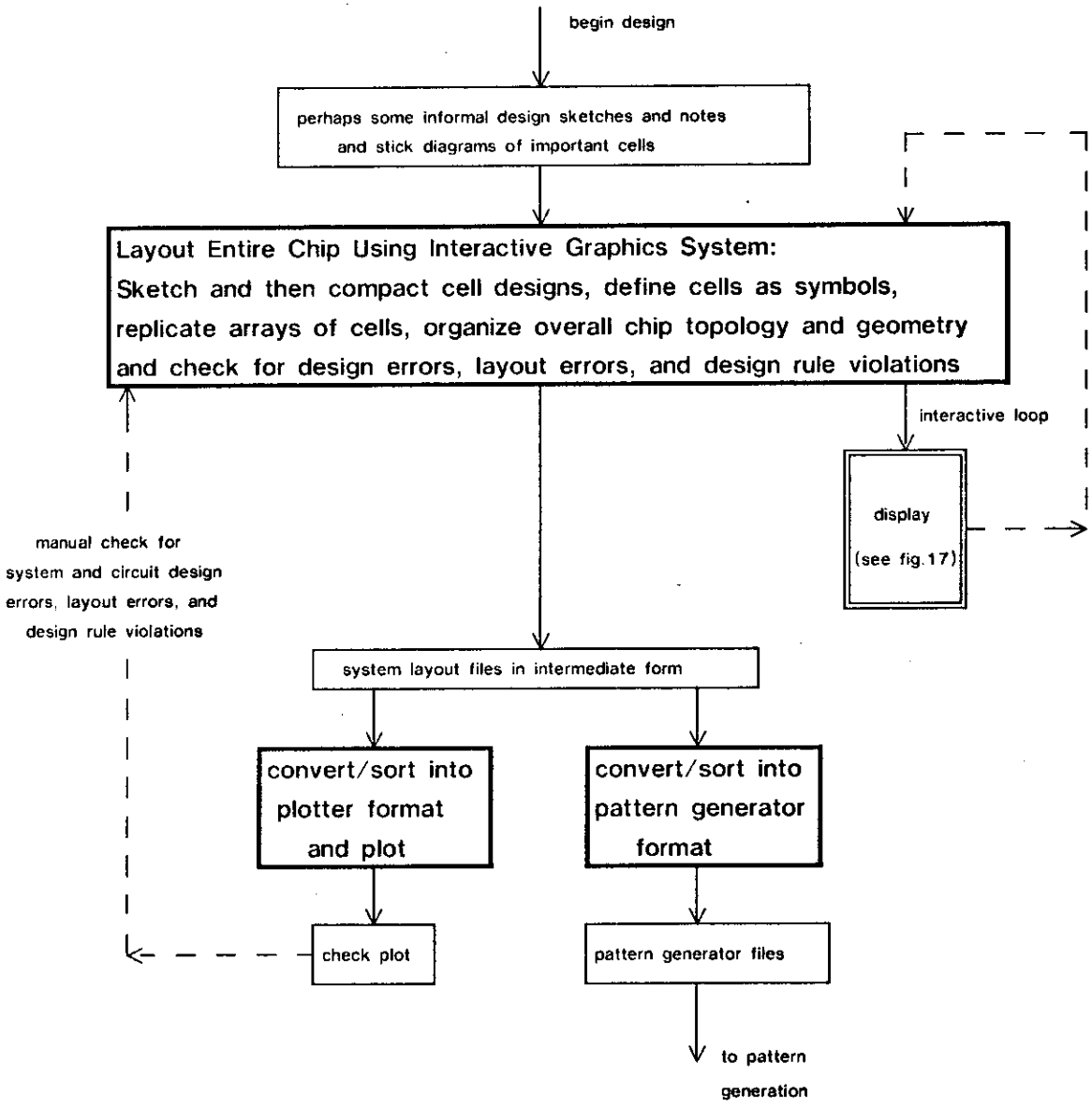


Fig. 16. Design, Layout, Design File Generation, and Design Checking Using an Interactive Graphics Layout System

compiled code in the system of which half is in memory at any given time. At minimum, the Alto memory has 64K 16 bit words. A 2.5 Mbyte cartridge disk drive is an integral part of the system. The user interacts with the system through an unencoded keyboard (software definable keys) and with a pointing device called a mouse (R2, p173). A cursor is controlled on the screen by moving the mouse around on a small area of the user's desk. A bit map display with a resolution of 600x800 dots is used for output, and printers for doing check plots are available through an in-house computer network.

The ICARUS display features two windows which provide a flexible working view of the layout, as shown in figure 17. The upper window is normally used for viewing a large piece of the layout at small magnification, and the lower window used for looking at a smaller section in more detail. The magnifications of the windows may be set independently.

In addition to the two windows there are various menus and status lines presented in the display. The menu on the left is the *command menu.* The menu under the upper window is the *parameter menu.* Under the parameter menu is the *stipple menu*, containing the mask level codes. Rectangles at a given level are stippled with the pattern for that level. The patterns were chosen so that, where necessary, one pattern could be seen through the other to verify that appropriate layers are overlapping properly. Current drawing coordinates and the status of system memory space are displayed to the right of the stipple menu.

The user interface is implemented principally through the display, the mouse and five conveniently located keys on the keyboard. Frequently used commands are given using only one or two simple hand operations, and can be done without glancing away from the display. These characteristics, coupled with rapid display redrawing, enhance the system's interactiveness.

The internal data representation in ICARUS is based on three types of items: rectangles, symbols, and text strings. The organization of these items into memory data structures, and the typical run-time memory space allocation is illustrated in figure 18.

Rectangles are created with the aid of the mouse. They may have angular orientations which are integer multiples of 45°. They can be moved, copied, or deleted using the mouse and one key. As items are created, they are added to an item list in main memory. Each rectangle is stored as 6 words in memory: the first word is the pointer to the next item, the second specifies what layer it is on, what type of item it is, etc. The third through sixth

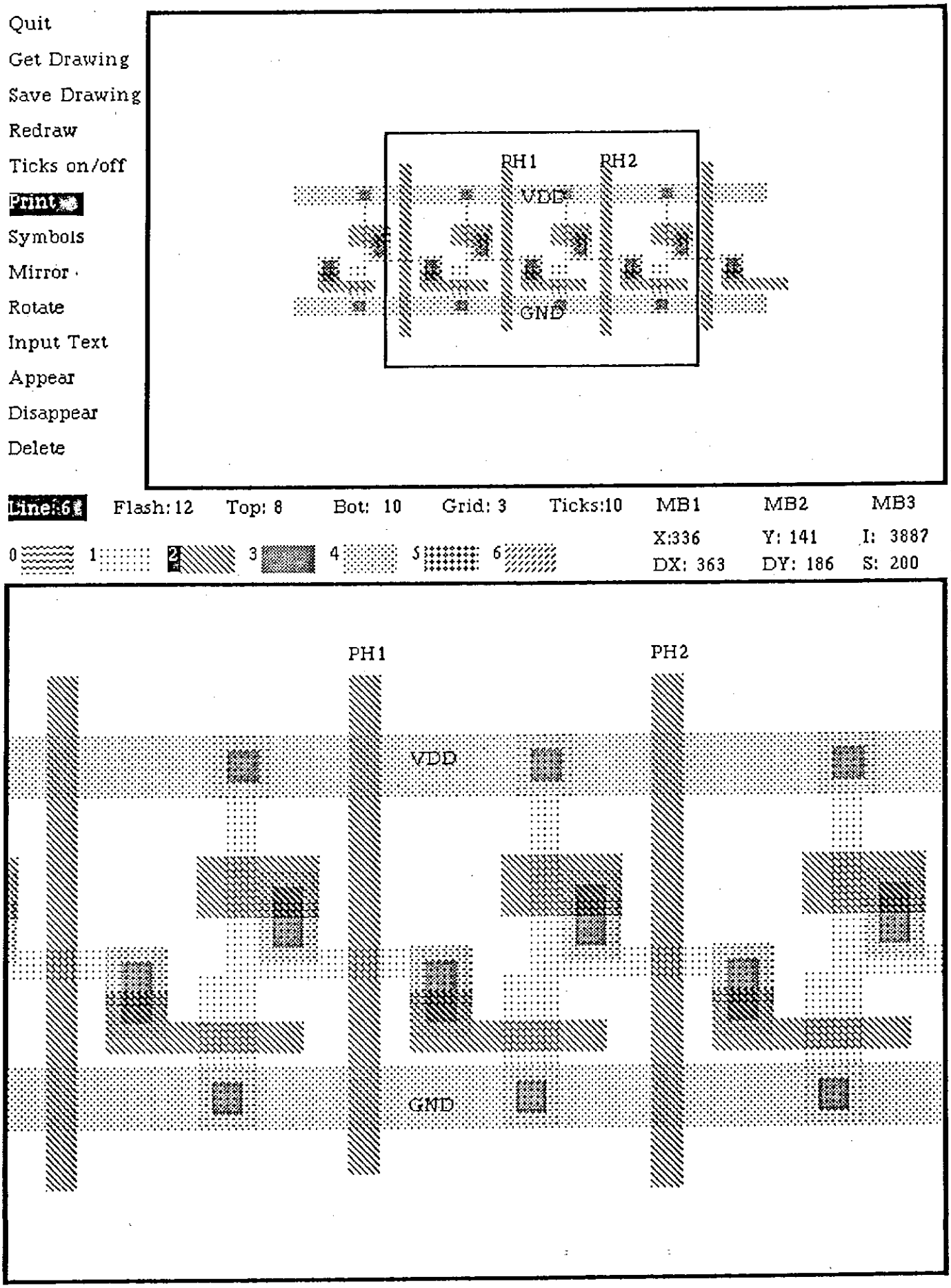


Fig. 17. The ICARUS Display: 2 Views of a Layout in Progress (windows.press)

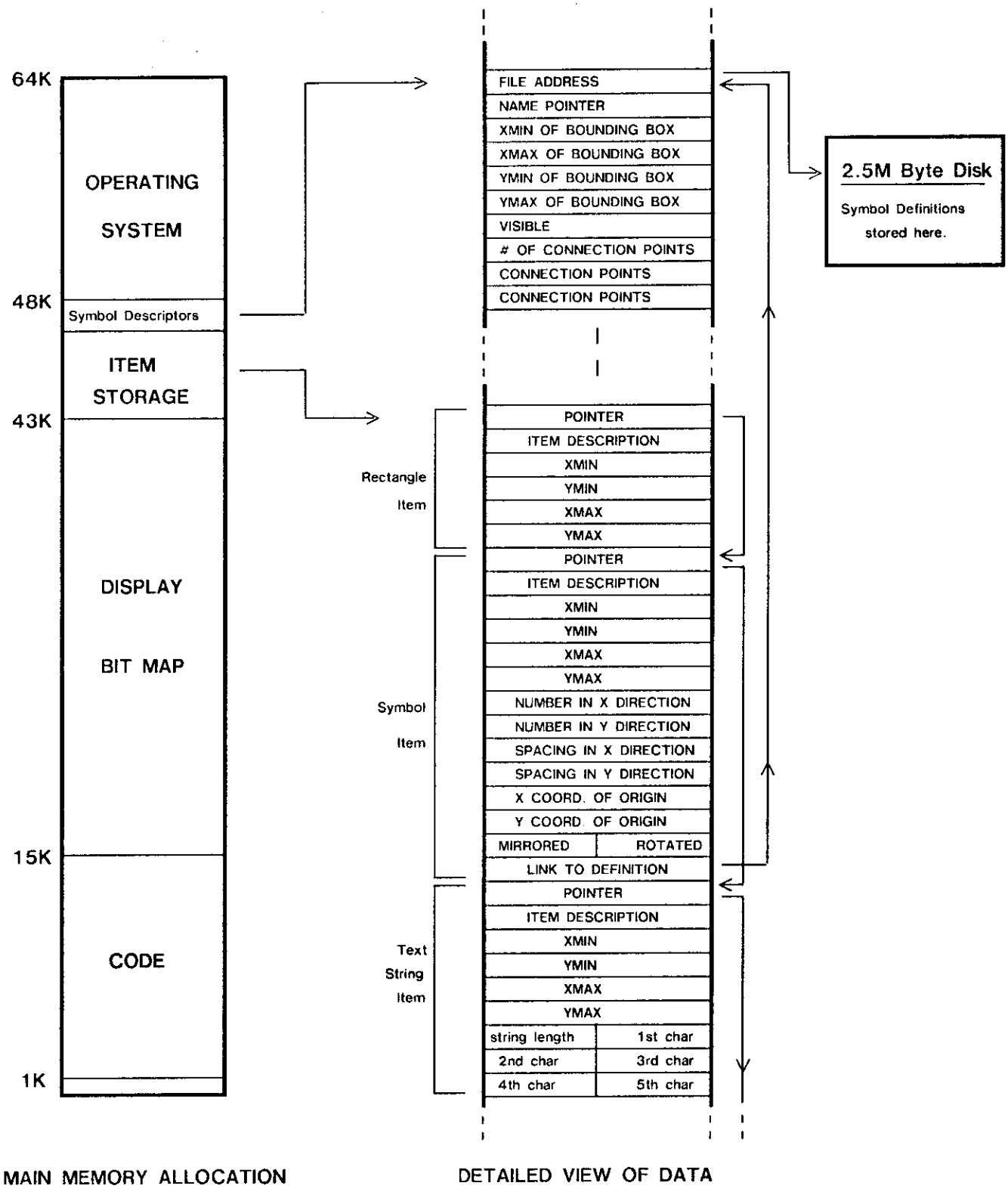


Figure 18. ICARUS Memory Allocation and Data Structure

(memory.sil)

words specify the minimum and maximum x and y coordinates. The items are kept in order of increasing values of minimum x coordinate, so that the display may be quickly redrawn.

When a symbol is defined by the user, the items which are contained within it are stored on the disk, while a pointer, the name and the bounding box for the symbol are placed in main memory. Symbols can be nested to any level. Once a *symbol definition* has been created, one is free to define *symbol instances* which are references to that definition. The symbol instance may be a command to draw one copy of the symbol at a certain location, or a whole array. The size of the symbol instance, which resides in main memory, is the same in both cases. The use of symbols wherever possible tends to preserve main memory space. Rather large systems can be designed using ICARUS, if the systems are well structured and make extensive use of symbols. This is true even when using a minimum sized 64K memory, which leaves little space for layout data.

Text is used for identifying data and control lines and is merely a memory aid to the user. There is no attempt to make use of the text or other information in the drawing for connectivity or other types of checking.

Operations more complex than those such as draw and move are implemented through the use of menus as shown in figure 17. The desired command is chosen by pointing at it with the cursor and clicking a mouse button. The selected command is then inverted to white on black video to identify its selection, which the user then confirms with a key on the keyboard. At this point, the system prompts the user with instructions presented in the display area normally holding the stipple menu. The instructions lead the user through the individual steps required, for example, to mirror or rotate a group of items.

Operations on symbols are defined in a secondary menu which can be reached by selecting the command "symbols" on the primary menu. The secondary menu offers commands such as define symbol, draw symbol, list the names of the symbols is the symbol library, or expand symbol. This last command is used to modify a symbol which is already defined, the modified symbol definition immediately updating all symbol instances which point to it.

Various system parameters are displayed in the parameter line directly below the top window. Values such as the default line width for the currently selected layer, the magnification of the top and bottom windows, and the spacing of the tick marks are all displayed. The parameter values can be changed at any time by selecting the desired one

and typing the new parameter value on the keyboard. The X,Y layout coordinates of the point last clicked with the mouse are displayed at the right of the screen. The DX,DY distances between the last two clicks are also displayed. This feature provides a convenient "ruler" for measuring distances on the layout.

The construction of an interactive layout system such as ICARUS is a relatively straightforward task for one who is experienced in interactive computer graphics (R2), given a display oriented minicomputer system and effective systems building software. A first version of ICARUS was constructed in 3 man-months, and a mature version produced in an additional 4 man-months.

ICARUS has been used internally in Xerox to lay out many integrated system projects, and to organize a number of multi project chips. Among the users were a number of individuals previously unfamiliar with integrated circuit layout, who nevertheless successfully completed LSI projects with up to 10,000 transistors. We find that the interactive nature of such a system not only aids the experienced designer, but also enhances the learning process for the novice. We believe that such interactive, personal design systems greatly enhance the creative ability of the designer by enabling easy generation and examination of many more design alternatives per unit time than would be the case with centralized, non-interactive design systems.

However, there is more to integrated system design than circuit layout. Design rules must be checked, logic transfer functions tested, and, in certain cases, circuit transfer functions computed to determine delays and predict system performance. We believe that the direction in which to search for further improvements in design tools is in the replacement of the primitive ICARUS type of data structure with one which allows design functions other than just layout to also interactively operate upon the same data base. This is the subject of the later section on fully integrated design systems.

The Caltech Intermediate Form for LSI Layout Description

[Section Contributed by Robert F. Sproull, Carnegie Mellon University, and Richard F. Lyon, Xerox PARC]

The Caltech Intermediate Form (CIF Version 2.0) is a means of describing graphic items (mask features) of interest to LSI circuit and system designers. Its purpose is to serve as a standard machine readable representation from which other forms can be constructed for specific output devices such as plotters, video displays, and pattern-generation machines. The intermediate form is not intended as a symbolic layout language: CIF files will usually be created by computer programs from other representations, such as a symbolic layout language or an interactive design program. Nevertheless, the form is a fairly readable text file, in order to simplify combining files and tracing difficulties.

The basic idea of the form is to specify literally every geometric object in the design using ample precision. Use of this form provides participating design groups easy access to output devices other than their own, enables sharing designs with others, allows combining several designs to form a larger chip, and the like. It is not necessary for all participating groups to implement the entire set of features of CIF, as long as their programs and documents contain warnings about unimplemented functions; nevertheless, the syntax must be correctly interpreted by all programs that read CIF, to assure a reasonable result.

CIF thus serves as the common denominator in the descriptions of various integrated system projects. No matter what the original input methods are (hand layout and coding, or a design system), the designs will be translated to CIF as an intermediate, before being translated again to a variety of formats for output devices or other design aids.

The original CIF was conceived by Ivan Sutherland and Ron Ayers in 1976. Subsequent improvements were contributed by Carlo Sequin, Douglas Fairbairn, and Stephen Trimberger.

This specification is divided into four parts: a description of the syntax of the form, a description of the semantics, an explanation of the transformations used, and a discussion of the conversion of wires to boxes.

Syntax

A CIF file is composed of a sequence of characters in a limited character set. The file contains a list of commands, followed by an end marker; the commands are separated with semicolons. Commands are:

Command	Form
Polygon with a path	P path
Box with length, width, center, and direction (direction defaults to (1,0) if omitted)	B integer integer point point
Round flash with diameter and center	R integer point
Wire with width and path	W integer path
Layer specification	L shortname
Start symbol definition with index, a, b (a and b both default to 1 if omitted)	DS integer integer integer
Finish symbol definition	DF
Delete symbol definitions	DD integer
Call symbol	C integer transformation
User extension	digit userText
Comments with arbitrary text	(commentText)
End marker	E

A more formal definition of the syntax is given below. The standard notation proposed by Niklaus Wirth[14] is used: production rules use equals = to relate identifiers to expressions, vertical bar | for or, and double quotes " " around terminal characters; curly brackets {} indicate repetition any number of times including zero; square brackets [] indicate optional factors (i. e. zero or one repetition); parentheses () are used for grouping; rules are terminated by period. Note that the syntax allows blanks before and after commands, and blanks or other kinds of separators (almost any character) before integers, etc. The syntax reflects the fact that symbol definitions may not nest.

```
cifFile              = { { blank } [ command ] semi }  endCommand { blank }.
command              = primCommand | defDeleteCommand |
                       defStartCommand semi { { blank } [ primCommand ] semi } defFinishCommand.
primCommand          = polygonCommand | boxCommand | roundFlashCommand | wireCommand |
                       layerCommand | callCommand | userExtensionCommand | commentCommand.

polygonCommand       = "P" path.
boxCommand           = "B" integer sep integer sep point [ sep point ].
roundFlashCommand    = "R" integer sep point.
wireCommand          = "W" integer sep path.
layerCommand         = "L" { blank } shortname.
defStartCommand      = "D" { blank } "S" integer [ sep integer sep integer ].
defFinishCommand     = "D" { blank } "F".
defDeleteCommand     = "D" { blank } "D" integer.
callCommand          = "C" integer transformation.
userExtensionCommand = digit userText.
commentCommand       = "(" commentText ")".
endCommand           = "E".
```

```
transformation      = { { blank } ( "T" point | "M" { blank } "X" | "M" { blank } "Y" | "R" point ) }.

path                = point { sep point }.
point               = sInteger sep sInteger.

sInteger            = { sep } [ "-" ] integerD.
integer             = { sep } integerD.
integerD            = digit { digit }.

shortname           = c [ c ] [ c ] [ c ].
c                   = digit | upperChar.
userText            = { userChar }.
commentText         = { commentChar } | commentText "(" commentText ")" commentText.

semi                = { blank } ";" { blank }.
sep                 = upperChar | blank.
digit               = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
upperChar           = "A" | "B" | "C" | ... | "Z".
blank               = any ASCII character except digit, upperChar, "-", "(", ")", or ";".
userChar            = any ASCII character except ";".
commentChar         = any ASCII character except "(" or ")".
```

Semantics

The fundamental idea of the intermediate form is to describe unambiguously the geometry of patterns for LSI circuits and systems. Consequently, it is important that all readers and writers of files in this form have exactly the same understanding of how the file is to be interpreted. Many of the decisions in designing the file format were made to avoid ambiguity or small but troublesome errors: floating point numbers are avoided; there are no iterative constructs, though there may be in future additions to CIF.

A simple file format might include only primitive geometric constructs, such as polygons, boxes, flashes and wires. Unfortunately, the geometric description of a chip with hundreds of thousands of rectangles on it would require an immense file of this sort. Consequently, we have made provision for defining and calling symbols; this should reduce the size of the file substantially.

It is important that programs processing CIF files operate cautiously, maintaining a constant vigilance for mistakes or entries that will not be processed properly. The description below mentions implementation suggestions or cause for caution inside brackets [].

Measurements. The intermediate form uses a right-handed coordinate system shown in Figure 19a, with x increasing to the right and y increasing upward. (Directions and distances are always interpreted in terms of the front surface of the finished chip, not in terms of the various sizes and mirrorings of the intermediate artifacts.) The units of distance measurement are hundredths of a micron (μm); there is no limit on the size of a number. [Programs reading numbers from CIF files should check carefully to be sure that the number does not overflow the number of bits in the internal representation used, and should specify their own limits, if any.]

Directions. Rather than measure rotation by angles, CIF uses a pair of integers to specify a "direction vector." (This eliminates the need for trigonometric functions in many applications, and avoids the problem of choosing units of angular measure.) The first integer is the component of the direction vector along the x axis; the second integer along the y axis. Thus a direction vector pointing to the right (the +x axis) could be represented as direction (1 0), or equivalently as (17 0); in fact, the first number can be any positive integer as long as the second is zero. A direction vector pointing NorthEast (i.e., rotated 45 degrees counterclockwise from the x axis) would have direction (1 1), or equivalently (3 3), and so on. [A (0 0) direction vector may be defaulted to mean the +x axis; a warning should be generated].

Geometric primitives. The various primitives that specify geometric objects are not intended to be mutually exclusive or exhaustive. CIF may be extended occasionally to accommodate more exotic geometries. At the same time, it is not necessary to use a primitive just because it is provided. Notice in the examples below that lower case comments and other characters within a command are treated as blanks, and that blanks and upper case characters are acceptable separators.

Boxes: Box Width 60 Length 25 Center 80,40 Direction -20,20; (or B60 25 80 40 -20 20;)
The fields which define a box are shown graphically in Figure 19a. Center and direction (optional, defaults to +x axis) specify the position and orientation of the box, respectively. Length is the dimension of the box parellel to the direction, and Width is the dimension perpendicular to the direction.

Polygons: Polygon A 0,0 B 10,20 C -30,40; (or P0 0 10 20 -30 40;)
A polygon is an enclosed region determined by the vertices given in the path, in order. For a polygon with n sides, n vertices are specified in the path (the edge connecting the last vertex with the first is implied; see Figure 19b). [Programs that try to interpret polygons may place

various restrictions on their paths; no set of constraints has been generally accepted, and no program currently exists for converting completely general polygons to pattern generator output.]

Flashes: RoundFlash Diam 200 Center -500,800; (or R200 -500 800;)
The diameter of a flash is sufficient to specify its shape, and the center specifies its position. (see Figure 19b). [Some programs may substitute octagons, or other approximations, for round flashes.]

Wires: Wire Width 50 A 0,0 B 10,20 C -30,40; (or W50 0 0 10 20 -30 40;)
It is sometimes convenient to describe a long, uniform width run by the path along its centerline. We call this construct a wire (see Figure 19b). An ideal wire is the locus of points within one half-width of the given path. Each segment of the ideal wire therefore includes semicircular caps on both ends. Connecting segments of the wire is a transparent operation, as is connecting new wires to an existing one: the semicircular overlap ensures a smooth connection between segments in a wire and between touching wires. [For output devices that have a hard time constructing circles, we approximate the ideal wire with squared-off ends. Notice that squared-off ends work nicely for segments meeting at right angles, but cause problems if wires or wire segments are connected at arbitrary angles. A way to circumvent this problem is to convert, prior to output, any wires in a file into connected sets of boxes of appropriate length, width, angle and center position (Figure 19c). The width of each box is the same as the width of the wire. The length of the boxes must be adjusted to minimize unfilled wedges and overlapping "ears". An algorithm for constructing boxes from a wire description is given in a later subsection. If the wire is specified within a symbol definition, the approximation need be computed only once, and can then be used each time the symbol is instantiated.]

Layer specification: Layer ND nmos diffusion; (or LND;)
Each primitive geometry element (polygon, box, flash, or wire) must be labeled with the exact name of a fabrication mask on which it belongs. Rather than cite the name of the layer for each primitive separately, the layer is specified as a "mode" that applies to all subsequent primitives, until the layer is set again (layer mode is preserved across symbol calls, which are discussed later).

The argument to the layer specification is a short name of the layer. Names are used to improve the legibility of the file, and to avoid interfering with the various biases of designers and fabricators about numbers (one person's "first layer" is another's "last"). [The intention of the layer specification command is to label locally the layer for a particular geometry. It is therefore senseless to specify a box, wire, polygon or flash if no layer has been specified. In order to detect this error, the command LZZZZ is implicitly inserted at the beginning of the file, and as the first command of

a symbol definition (DS; see below). Any attempt to generate geometric output on layer ZZZZ will result in an error.]

It is important that layer names be unique, so that combining several files in intermediate form will not generate conflicts. The general idea is that the first character of the name denotes the technology, and the remainder is mnemonic for the layer. At present, the following layers are defined:

ND	NMOS Diffusion
NP	NMOS Polysilicon
NC	NMOS Contact cut
NM	NMOS Metal
NI	NMOS depletion mode Implant
NB	NMOS Buried contact
NG	NMOS overGlass openings

New layer names will be defined as needed.

[Programs that read CIF will want to check to be sure that layer names used do in fact correspond to fabrication masks being constructed. However, the file may cite layer names not used in a particular pass over the CIF file. It would be helpful for the program to provide a list of the layer names that it ignored.]

Symbols. Because many LSI layouts include items that are often repeated, it is helpful to define often-used items as "symbols." This facility, together with the ability to "call" for an instance of the symbol to be generated at a specific position, greatly reduces the bulk of the intermediate form.

The symbol facilities are deliberately limited, in order to avoid mushrooming difficulties of implementing programs that process CIF files. For example, symbols have no parameters; calling a symbol does not allow the symbol geometry to be scaled up or down; there are no direct facilities for iteration. The main reason for symbol facilities is to limit the file size; if the symbol mechanism is not adequate for some application, the desired geometry can still be achieved with less use of symbols, and more use of explicit geometrical primitives. [Symbols need not be used at all; this eliminates the need for intermediate storage for symbol definitions, but results in larger design files. Machines which must process a fully-instantiated representation of a layer (such as pattern generators) might only accept CIF files without symbol definitions, to reduce the cost of implementation. Therefore, it would be useful to have a program that would convert general CIF files to fully instantiated CIF files, and maybe to sort by layer, location, or whatever.]

The ability to call for iterations (arrays) of symbols is not provided in CIF Version 2.0.

This is primarily due to the difficulty of defining a standard method of specifying iterations, without introducing machine-dependent computation problems. It is still possible to acheive a great deal of file compaction by defining several layers of symbols (e.g. cell, row, double-row, array, etc.). However, the ability to iterate symbol calls is a likely prospect for a future addition to CIF.

Defining symbols: Definition Start #57 A/B = 100/1; ... ; Definition Finish; (or DS57 100 1; ... ;DF;)
A symbol is defined by preceding the symbol geometry with the DS command, and following it with the DF command. The first argument of the DS command is an identifying symbol number (unrelated to the order of listing of symbol definitions in the file).

The mechanism for symbol definition includes a convenient way to scale distance measurements. The second and third arguments to the DS command are called a and b respectively. As the intermediate form is read, each distance (position or size) measurement cited in the various commands (polygons, boxes, flashes, wires and calls) in the symbol definition is scaled to (a*distance)/b. For example, if the designer uses a grid of 1 micron, the symbol definition might cite all distances in microns, and specify a=100, b=1. Or the designer might choose lambda (characteristic fabrication dimension) as a convenient unit. This mechanism reduces the number of characters in the file by shrinking the integers that specify dimensions and may improve the legibility of the file (it does not provide scaling, or the ability to change the size of a symbol called within the definition).

Definitions may not nest. That is, after a DS command is specified, the terminating DF must come before the next DS. The definition may, however, contain calls to other symbols, which may in turn call other symbols.

There is only one restriction on the placement of symbol definitions in the file: a symbol must be defined before its instantiation becomes necessary. This constraint can be satisfied by placing all symbol definitions first in the file, and then calls on the symbols. In fact, it is often convenient to have the file consist exclusively of symbol definitions and ONE call on a symbol. This call will be the last command in the file before the end command. [If a definition redefines a symbol that already exists, the previous definition is discarded; a warning message should be generated. When several people contribute to a design, some symbol management is therefore necessary; see *Deleting symbol definitions* below.]

Calling symbols: Call Symbol #57 Mirrored in X Rotated to -1,1 then Translated to 10,20;
The C command is used to call a specified symbol and to specify a transformation that should be applied to all the geometry contained in the symbol definition. The call command identifies the symbol to be called with its "symbol index," established when the symbol was defined.

The transformation to be applied to the symbol is specified by a list of primitive transformations given in the call command. The primitive transformations are:

T point	Translate the current symbol origin to this point.
M X	Mirror in X, i.e., multiply X coordinate by -1.
M Y	Mirror in Y, i.e., multiply Y coordinate by -1.
R point	Rotate symbol's x axis to this direction.

Intuitively, each coordinate given in the symbol is transformed according to the first primitive tranformation in the call command, then according to the second, etc. Thus "C1 T500 0 MX" will first add 500 to each x coordinate from symbol 1, then multiply the x coordinate by -1. However, "C1 MX T500 0" will first multiply the x coordinate by -1, and then add 500 to it: the order of application of the transformations is therefore important. In order to implement the transformations, it is not necessary to perform each primitive operation separately; the several operations can be combined into one matrix multiplication (see the subsection on transformations).

Symbol calls may nest; that is, a symbol definition may contain a call to another symbol. When calls nest, it is necessary to "concatenate" the effects of the transformations specified in the various calls (see the subsection on transformations). [There is no sensible way in which a symbol may be invoked recursively (i.e., call itself, either directly or indirectly). Programs that read the intermediate form should check that no recursion occurs. This can be achieved by retaining a single flag with each symbol to indicate whether the symbol is currently being instantiated; the flags are initialized to "false." When a symbol is about to be instantiated, we check the flag; if it is "true," we have detected recursion, print an error message and do not perform the call. Otherwise, we mark the flag "true," instantiate the symbol as specified, and mark the flag "false" when the instantiation is complete.]

Layer settings are preserved across symbol calls and definitions. Thus, in the sequence:

```
LNM;
S6 20 0;
C 57 T45 13;
DS 114...;
DF;
LNM;
S3 0 0;
```

the second LNM is not necessary, regardless of the specification of symbols 57 and 114.

Deleting symbol definitions: Delete Definitions greater than or equal to 100; (or DD100;)
The DD command signals the program reading the file that all symbols with indices greater than or equal to the argument to DD can be "forgotten" -- they will not be instantiated again. This feature is included so that several intermediate form files can be appended and processed as one. In such a case, it is essential to delete symbol definitions used in the first part of the file both because the definitions may conflict with definitions made later and because a great deal of storage can usually be saved by discarding the old definitions.

The argument to DD that allows some definitions to be kept and some deleted is intended to be used in conjunction with a standard "library" of definitions that a group may develop. For example, suppose we use symbol indices in the range 0 to 99 for standard symbols (pullup transistors, contacts, etc.) and want to design a chip that has 2 student projects on it. Each project defines symbols with indices 100 or greater. The CIF file will look like:

```
/Definitions of library symbols;
DS 0 100 1;
/ ...definition of symbol 0 in library;
DF;
DS 1 100 1;
/ ...definition of symbol 1;
DF;
/ ...remainder of library;

/Begin project 1;
DS100 100 1;
/ ...first student's first symbol definition;
DF;
...
DS109 100 1;
/ ...first student's main symbol definition;
DF;
C109 T403 -110;/ call on first student's main symbol;

DD100;/Preserve only symbols 1 to 99;

/Begin project 2;
DS100 100 1;
/ ...second student's first symbol definition;
DF;
...
DS113 100 1;
/ ...second student's main symbol definition;
C1 T-3 45;/Call on library symbol, still available;
DF;
C113 T401 0;/ call on second student's main symbol;

E
```

User expansion: 3'SYMBOL.LIBRARY'; 5:NONSTANDARD DESIGN RULES: LAMBDA = 4.0;

Several command formats (any command starting with a digit) are reserved for expansion by individual users; the authors of the intermediate form agree never to use these formats in future expansions of the standard format. For example, private expansions might provide for (1) requesting that another file be "inserted" at this point in the processing, thus simplifying the use of symbol libraries; (2) inserting instructions to a preprocessor that will be ignored by any program reading only standard intermediate form constructs; or (3) recording ancillary information or data structures (e.g., circuit diagrams, design-rule check results) that are to be maintained in parallel with the geometry specified in the style of the intermediate form.

Comments: (HISTORY OF THIS DESIGN:);
The comment facility is provided simply to make the file easier to read. [It is possible to deactivate any number of commands by simply enclosing them within a pair of parentheses, even if they already include balanced parentheses.]

End Command: End of file.
The final E signals the end of the CIF file. [Programs that read CIF should gave an error message if the file ends without an End Command, or a warning if more text other than blanks follows the E.]

Transformations (see also reference R2)

When we are expanding a symbol, we need to apply a transformation to the specification of an item in the symbol definition to get the specification into the coordinate system of the chip. There are three sorts of measurements that must be transformed: distances (for widths, lengths), absolute coordinates (for "points" in all primitives) and directions (for boxes).

Distances are never changed by a symbol call, because we allow no scaling in the call. Thus a distance requires no transformation.

A point (x,y) given in the symbol is transformed to a point (x',y') in the chip coordinate system by a 3x3 transformation matrix T:

$$[x'\ y'\ 1] = [x\ y\ 1]\ T$$

[It is a good idea to check either the last column of T, or the 1 at the end of the transformed vector, even though they never need to be computed.]

T is itself the product of primitive transformations specified in the call: T = T1 T2 T3, where T1 is a primitive transformation matrix obtained from the first transformation primitive given in the call, T2 from the second, and T3 from the third (of course, there may be fewer or more than 3 primitive transformations specified in the call). These matrices are obtained using the following templates for each kind of primitive transformation:

T a b.	Tn =	1	0	0	
		0	1	0	
		a	b	1	
M X.	Tn =	-1	0	0	
		0	1	0	
		0	0	1	
M Y.	Tn =	1	0	0	
		0	-1	0	
		0	0	1	
R a b.	Tn =	a/c	b/c	0	
		-b/c	a/c	0	where $c = \mathrm{Sqrt}(a^2+b^2)$
		0	0	1	

Transformation of direction vectors (x y) is slightly different than the transformation of coordinates. We form the vector [x y 0], and transform it by T into the new vector [x' y'

0]. The transformed direction vector is simply (x' y'). [Note that some output devices may require rotations to be specified by angles, rather than direction vectors. Conversion into this form may be delayed until necessary to generate the output file. Then we calculate the angle as ArcTan(y/x), applying care when x=0.]

Nested calls require that we combine the transformations already in effect with those specified in the new call. Suppose we are expanding a symbol a, as described above, transforming each coordinate in the symbol to a coordinate on the chip by applying matrix Tac. Now we encounter, in a's definition, a call to b. What is to happen to coordinates specified in b? Clearly, the transformations specified in the call will yield a matrix Tba that will transform coordinates specified in symbol b to the coordinate system used in symbol a. Now these must be transformed by Tac to convert from the system of symbol a to that of the chip. Thus, the full transformation becomes

```
[x' y' 1] = [x y 1] Tba Tac
```

The two matrices may be multiplied together to form one transformation Tbc = (Tba Tac) that can be applied to convert directly from the coordinates in symbol b to the chip. This procedure can be carried to an arbitrary depth of nesting.

To implement transformations, we proceed as follows: we maintain a "current transformation matrix" T, which is initialized to the identity matrix. We use this matrix to transform all coordinates. When we encounter a symbol call, we:

1. "Push" the current transformation and layer name on a stack.
2. Set layer name to ZZZZ.
3. Collect the individual primitive transformations specified in the call into the matrices T1, T2, T3 etc.
4. Replace the current transformation T with T1 T2 T4 ... T; i.e., premultiply the existing transformation by the new primitive transformations, in order).
5. Now process the symbol, using the new T matrix.
6. When we have completed the symbol expansion, "pop" the saved matrix and layer name from the stack. This restores the transformation to its state immediately before the call.

Decomposing Wires Into Boxes

The following algorithm for decomposing wires into boxes was developed by Carver Mead, and first implemented at Caltech by Ron Ayers; it was was further modified to be consistent with the use of direction vectors, to allow more general path lengths, and to avoid use of trigonometric functions. [Note that this decomposition covers more area than the locus of points within w/2 of the path for small angles of bend, but less area for sufficiently sharp bends; in particular, if a path bends by 180 degrees (reverses) it will have no extension past the point of reversal (it is missing a full semicircle). Other decompositions are possible, and may better approximate the correct shape.]

Let the wire consist of a path of n points $p_1,...,p_n$.
Let w represent the width of the wire.

```
"Initialization:"
IF n = 0 THEN DONE; "no path"
IF n = 1 THEN
  {MAKEFLASK[Diameter ← w, Center ← p1]; "single-point gets a flash";
  DONE;};
i ← 1;
OldEntension ← w/2; "initial end of wire"
Segment ← p2 - p1;  "p1 and p2 are points in path, Segment is a vector (a point)"
"LoopConditions:"
FOR pi, pi+1 in path UNTIL pi+1 is last DO
"calculate the box for the segment from pi to pi+1:"
IF pi+1 is last THEN { Entension ← w/2;  "final end of wire" }
  ELSE
  { "compute Entension for intermediate point:"
  NextSegment ← pi+2 - pi+1;  "next vector in path"
  T ← MATRIX[  X[Segment], -Y[Segment],
               Y[Segment],  X[Segment] ];
  "T transforms Segment to +x axis."
  Bend ← MULTIPLY[ NextSegment, T ];  "relative direction vector"
  "if Bend is (0 0), delete pi+1, reduce n, and start over"
  Entension ← w/2 * ( ABS[Y[Bend]] /
              ( LENGTH[Bend] + ABS[X[Bend]] ) );
  };
MAKEBOX [ { Length    ← LENGTH[Segment] + Entension + OldEntension; },
          { Width     ← w; },
          { Center    ← (pi + pi+1)/2 + ( Segment / LENGTH[Segment] ) *
                        (Entension - OldEntension)/2; },
          { Direction ← Segment;  "careful, may be zero vector" } ];
i ← i + 1;
OldEntension ← Entension;
Segment ← NextSegment;  "next vector in path"
ENDLOOP;
DONE;
```

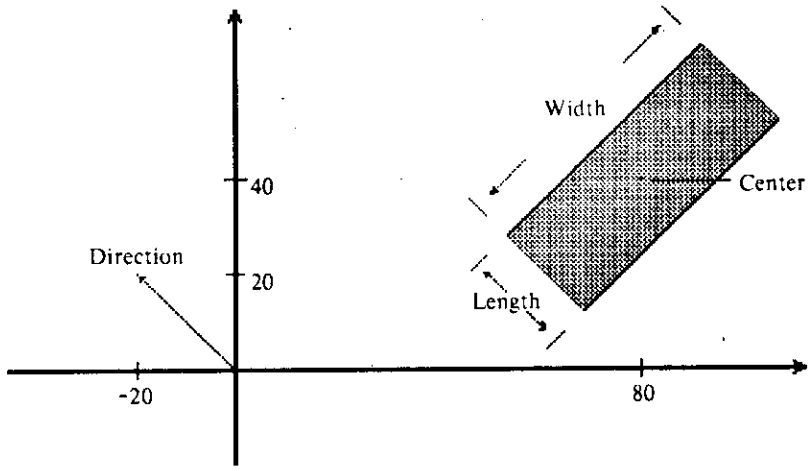


Fig. 19a. Box Representation in Intermediate Form

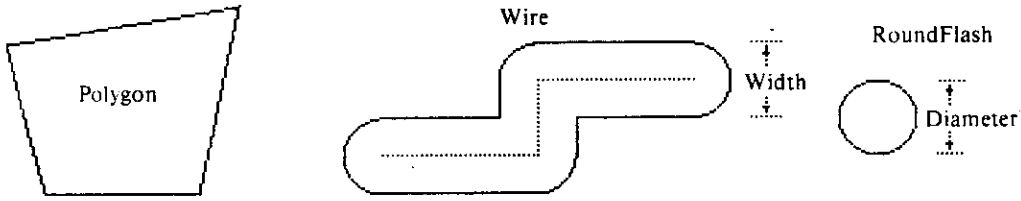


Fig. 19b. Other Items in the Intermediate Form

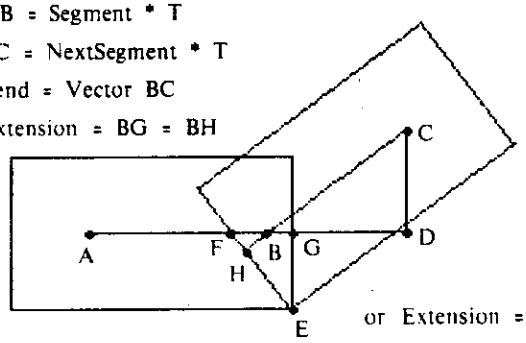
T transforms Segment to the +X axis

AB = Segment * T

BC = NextSegment * T

Bend = Vector BC

Extension = BG = BH



Similar triangles BCD, EFG, BFH

BC:CD:DB :: EF:FG:GE :: BF:FH:HB

FG = FB + BG

= BH * (BC/DB) + BG

= (1 + BC/DB) * BG

BG = FG / (1 + BC/DB)

= GE * (BC/DB) / (1 + BC/DB)

= GE * (CD / (DB + BC))

or Extension = w/2 * Y[Bend] / (LENGTH[Bend] + X[Bend])

fig. 19c. Converting Wires to Boxes

(cif19.press)

The Multi-Project Chip

Insight into integrated system design is most quickly gained by actually carrying through to completion several LSI design projects, each of increasing scope. A large, complex VLSI system could be quickly and successfully developed by designers able to easily implement and test prototypes of its subsystems. The separate subsystems can be implemented, tested, debugged, and then merged together to produce the overall system layout. However, such activities are only practical if a scheme exists for carrying out implementation with minimum turnaround time and low procedural overhead per project.

In this section we describe procedures for organizing and implementating many small projects by merging their layouts onto one *multi-project chip.* Then each designer of a small project or subsystem need not carry the entire procedural burden involved in maskmaking and fabrication. We also include a collection of practical tips and hints that may prove useful to those undertaking their first projects or organizing their first multi project chips. While the details in this section are specific to present maskmaking and fabrication technology, they nevertheless give a feeling for the sort of things that must be done to implement projects in general. In a later section we discuss how multiple project implementation might be done in the future.

Figure 20 contains a photomicrograph of a Caltech class project chip containing 15 separate student projects. The individual projects were simply merged together onto one typically sized chip layout, approximately 3 mm by 4 mm, and implemented simultaneously as one chip type. Most of these projects are prototypes of digital subsystems designed using the methodology of this text. By implementing a small "slice" of a prototype subsystem array, one can verify that its design, layout, and implementation are correct, and measure its power and delay characteristics as yielded by the particular fabrication process, thus gaining almost as much information as would be obtained by implementing the full array.

Following fabrication, the wafers containing such multi project chips are scribed, diced, and then divided up among the participants. The typical minimum fabrication run is about 10 wafers, each ~7.5 to 10 cm in diameter. Thus even a minimum run provides a few thousand chips, and each participant ends up with many chips. Participants may then each package

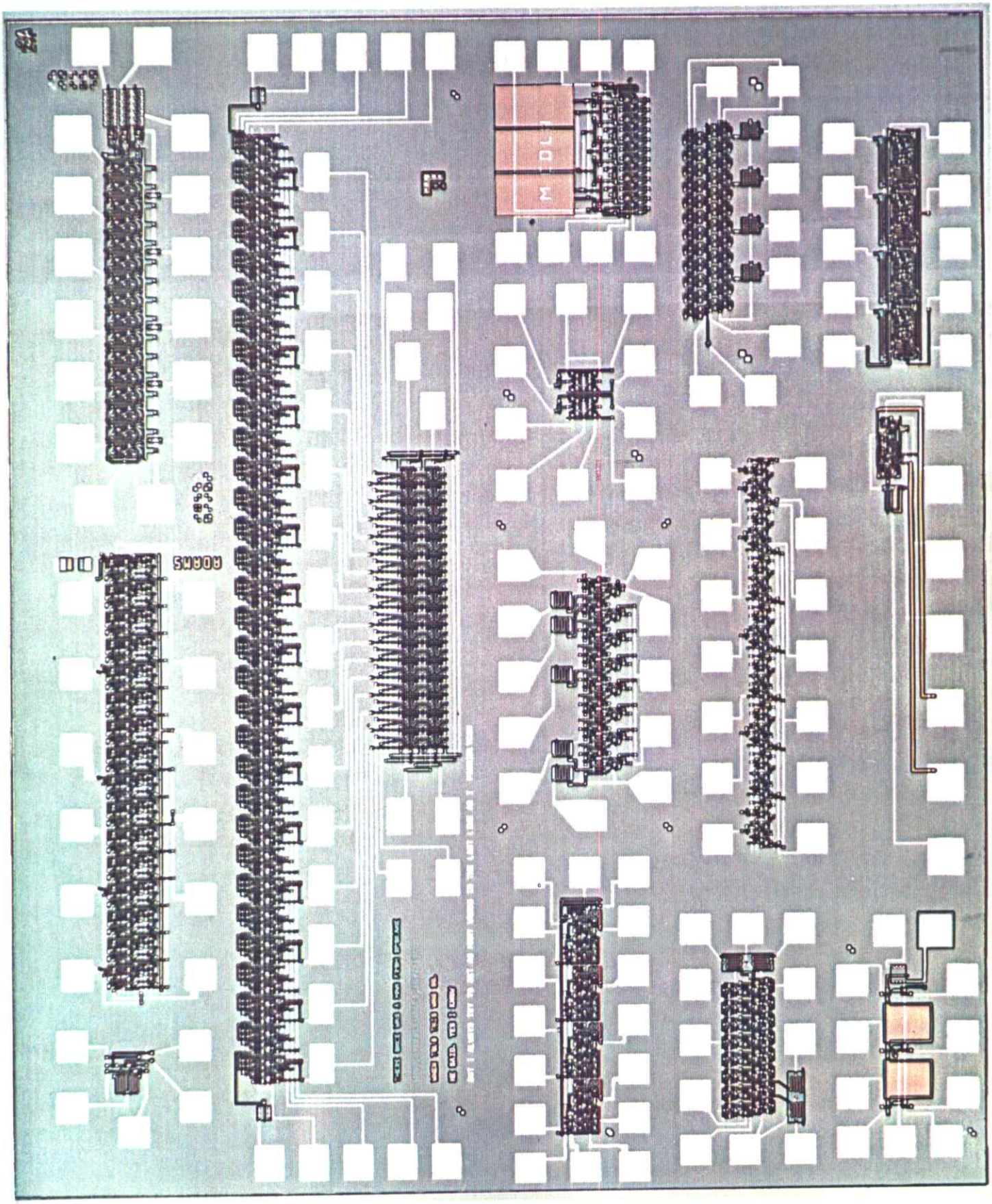


Fig. 20. Photomicrograph of a Caltech Class Project Chip

their chips, bonding the package leads to the contact pads of their individual project. Since most such projects are relatively small in area, yields are unusually high: if a project's design and layout have been done correctly, most of the corresponding chips will work.

Organizing a multi-project chip involves: (i) creating the layout of a *starting frame*, into which the various projects are to be merged, (ii) gathering, relocating, and merging the project layouts into the starting frame to create one design file and generating from this the PG files for the overall project chip, and (iii) documenting various parameters and specifications to be used during maskmaking and fabrication.

The starting frame contains all the auxiliary portions of the chip layout: scribe lines, alignment marks, line width testers (critical dimension marks), and test patterns. The starting frame may contain fiducial marks on each mask level if these are not to be placed by the mask house, and in some cases may contain a parity mark on each level to mark the appropriate reticle side and orientation during step and repeat reduction. A tip: placing a mask level name or symbol somewhere within the chip's scribe line boundary on each level helps prevent the fatal error of level interchange at some time during project merging, maskmaking, or fabrication.

The contents of this starting frame must be carefully worked out to meet the requirements and constraints of the chosen mask house and fab line. The important factor of turnaround time for the entire mask and fab sequence may be reduced to some extent by repeatedly using a relatively standard starting frame which then becomes familiar to all those involved. Some typical values for the time involved: 2 to 3 weeks for maskmaking, and then 3 to 4 weeks for fabrication, longer if large work queues exist at the mask or fab firms.

When a multi-project chip is scheduled, a tentative chip partition for each project can be negotiated among the participants. Project design and layout can then proceed, with iterations on the space allocation being done right up till the final merging. The gathering and merging of project layout files into one design file is simplified if they are in a *common intermediate form*. Projects may then be relocated to their respective partitions of the chip, displayed, plotted, or otherwise checked, using minimum and consistent software operating upon manageable sized files. When the project chip appears correctly organized, pattern generator (PG) files are produced and written on a mag tape to be sent to the mask house.

An alternative to the merging of projects at the intermediate form level, is the relocation

and merging of their PG files. However, the PG files for major designs, containing fully instantiated artwork, become unwieldy in size even at today's complexity. The PG file merging scheme is workable for projects of small to moderate size, and does provide a contingency plan for including projects having alien intermediate forms. If designs are relocated and merged at the PG level, additional software should be provided for displaying or plotting the chip at that level, so that merging errors may be spotted. A tip: it is a good idea in any case to have some bounds checking to prevent stray items of one project from clobbering another.

A thought: the interface between design groups and mask houses would be cleaner if design files in a *common intermediate form*, such as CIF, rather than PG files were used to transmit designs to the patterning process. Files would be much smaller. The use of data links would be eased. The process to convert and sort design files into PG files, involving patterning mechanism dependent optimization, would be appropriately located: in association with the particular patterning mechanism.

Examples of Multi-Project Chips:

The above concepts and some further possibilities may be clarified by examining the details of some specific examples. Figure 21 illustrates a collaborative Xerox PARC/Caltech multi-project chip set [organized by D. Fairbairn, D. Johannsen, R. Lyon, J. Rowson, S. Trimberger]. The figure was produced as a software blowback from the PG file, of the metal level of this chip set. Projects in the set ranged in scope from the test of a few cells of an experimental, low power shift register [C. Sequin, U. C. Berkeley, and R. Lyon, Xerox PARC], up to a complete content addressible cache memory system [D. Fairbairn].

Although several of the projects in the set are fairly large, all were individually designed to yield chip sizes packagable in standard 40 pin packages, which can hold chips up to ~ 7 mm square. The pattern generator at the intended mask house was a GCA/D.W.Mann 3600, and the photorepeater was a Mann 3696. Together, these can produce 10x reticles having field sizes as large as 10 cm square, and can reduce, step, and repeat these at a maximum of 10mm x,y intervals onto masks. Therefore, the 3600/3696 can provide masks for square chips up to 10 mm (10,000μ) on a side. A 10mm square chip can hold the patterns of several normally sized chips. By including *interior scribe lines* in the starting frame, as indicated in figure 21, one reticle set can be patterned on the Mann 3600 to contain a number of

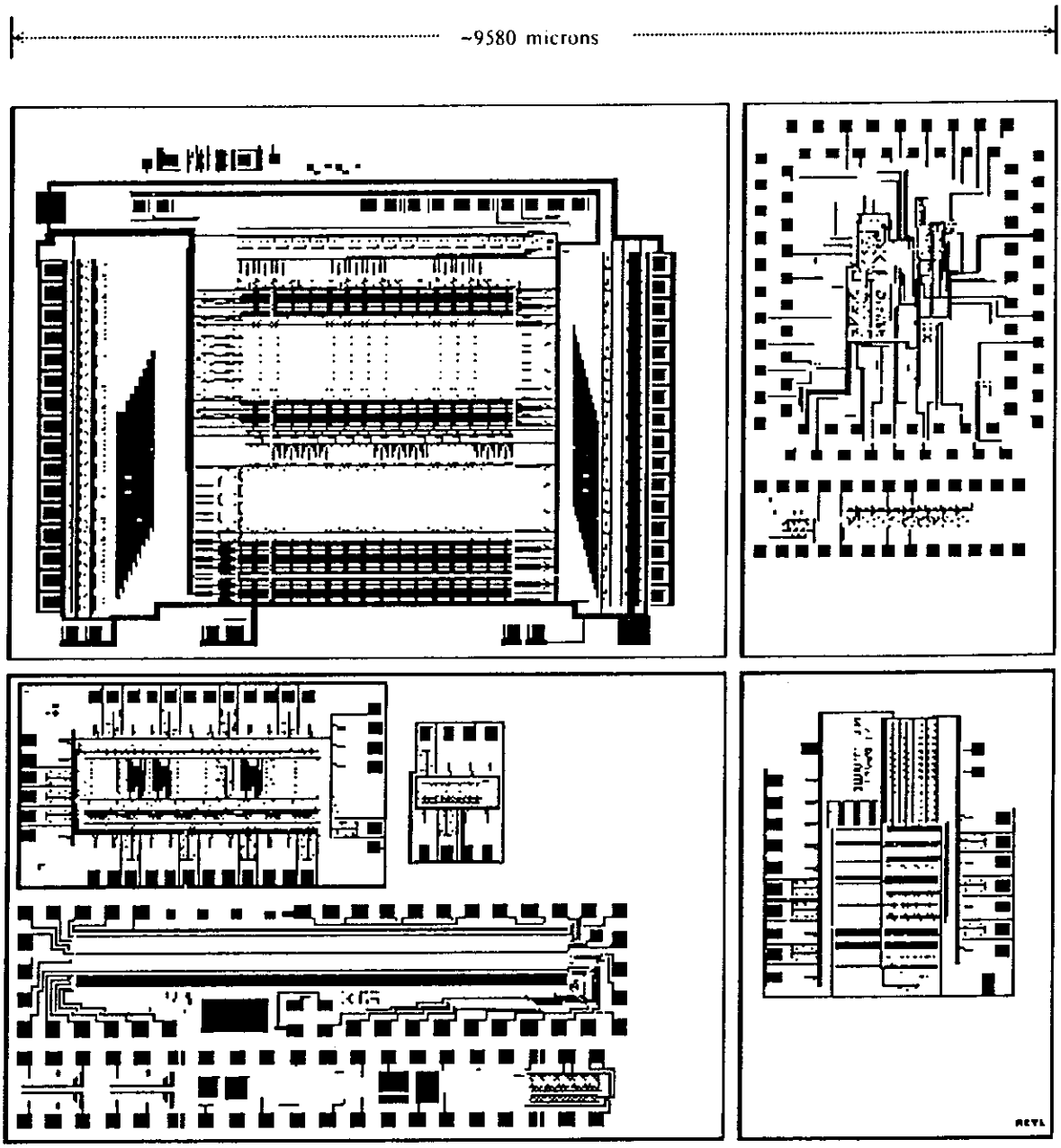


Fig. 21. Collaborative Xerox PARC/Caltech Multi-Project Chip

[Software Blowback from the PG File of the Metal Layer]

different chips, each of which may contain more than one project. When masks are made, each reticle is photorepeated at intervals in x,y corresponding to its outer dimensions minus some scribe line overlap. In the example in figure 21, the x,y stepping distances were both ~9700 microns. Fabricated wafers are scribed and diced on all scribe lines, including the interior ones, to yield chips of typical sizes. One of the projects, on the lower left chip in figure 21, is an experimental charge coupled device array [R. Davies]. The CCDs rode along on this chip set to obtain working masks for use in a completely different process technology (triple poly) from the standard nMOS the other projects used.

Figure 22 provides a higher magnification PG file software blowback of the region near the center of the left scribe line of the chip set. Alignment marks and line width testers (C/D's) were placed in this region, as noted in the figure. Software blowbacks of individual mask levels, more closely resembling the reticles and masks than would a composite design checkplot of all levels, are useful in conveying such location information to the mask and fab houses. Parity marks were not needed on the reticles for this project chip set. Fiducial marks were placed on the reticles by the mask house. Since the software converting the design files to PG files had just been constructed prior to organizing this chip set, reticle blowbacks were requested before proceeding further with maskmaking to verify that everything through pattern generation had worked correctly.

Some other practical details: Participants in the chip set shared some of the commonly used layout items normally required in any project. Examples were input contact pads with attached "lightning arrestor" circuits to protect the input MOSFET gates, and output drivers snaked around and attached to output pads. Even at current device sizes, pads occupy a large fraction of the chip area for large collections of projects, and participants tend to make the pads as small as their bonding skill allows. A square pad ~75μm on a side is a rather small bonding target, and 125μm on a side is easier for the novice to hit. Perhaps ~100μm square pads separated by ~75μm is a good compromise, and these should be at least 25μm from any other metal lines to avoid shorting the lines when bonding.

The scribe lines on this chip set were laid out as 140μm wide cuts down to 160μm wide paths on the diffusion level, to provide lanes free of oxide for scribing or sawing. Metal paths 30μm wide were then laid out straddling the boundaries of these scribe lines, to provide electrical contact from the substrate to the metal during the etching of the metal layer. Since all the projects on this chip set were prototype designs, and were not intended to be placed in extended use, the chips were not overglassed. Eliminating the overglassing

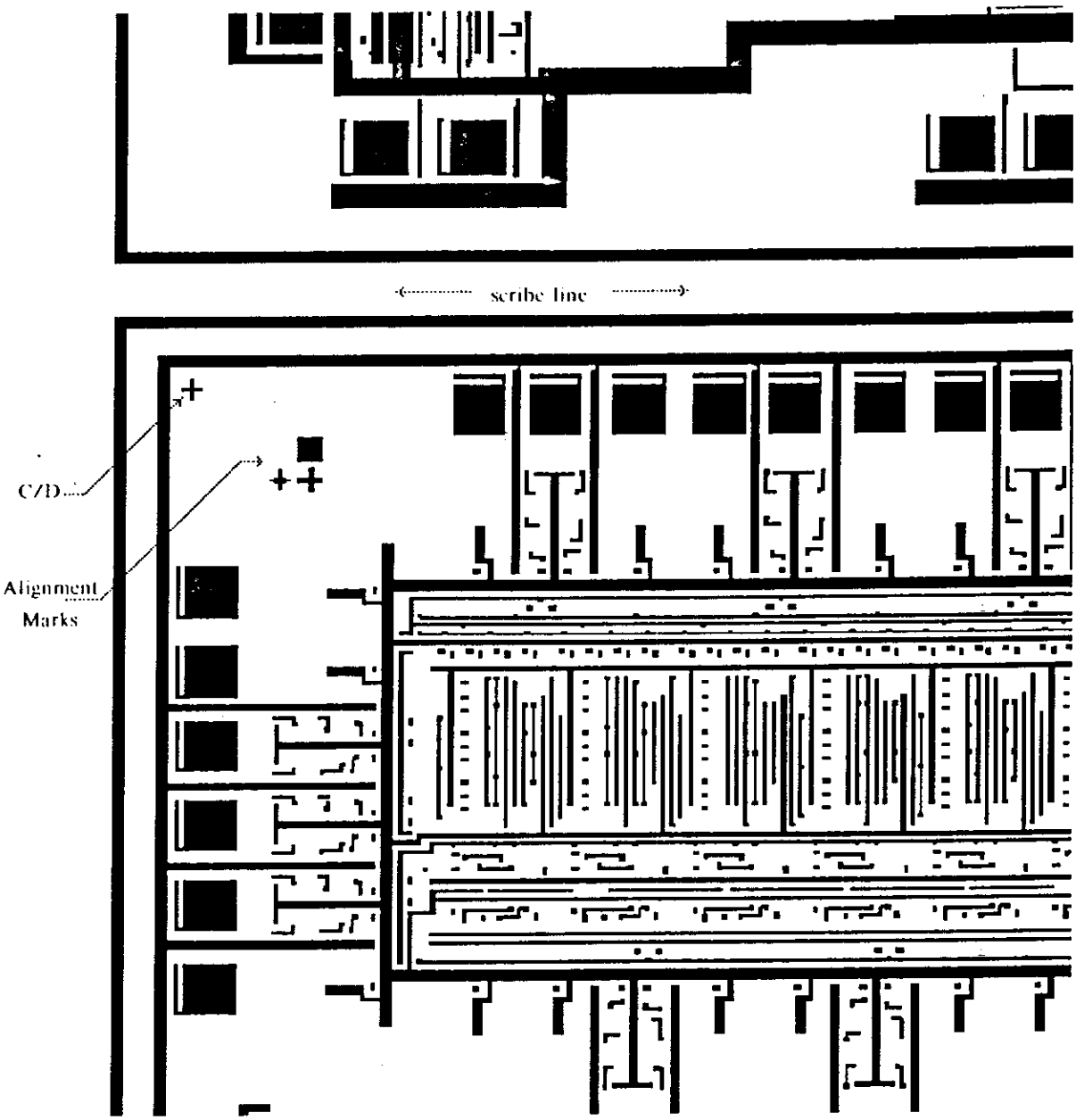


Fig. 22. Close-Up View of Figure 21 Region Containing Mask & Fab Information

meant that a mask level for defining cuts through overglassing over the contact pads and scribe lines was not needed, reducing maskmaking costs. On the other hand, the chip set included a mask level to pattern the thin gate oxide, to provide buried contacts between diffusion and poly that do not require metal coverage as does the butting contact. Such buried contacts enable more compact layouts, but are subject to a rather complex set of design rules, require an extra mask level, and sometimes reduce yield and reliability.

Deleting the overglassing process step also made it possible to electrically probe interior points on the chips during testing, probing small metal test pads included in the layouts. Such pads must be placed with care, however, because they hang relatively large capacitances onto circuitry and slow it down. Note that test pad probing requires special jigs and a stereo microscope, and that it is only possible to directly probe the metal layer. Testing uncovered chips may also require reduced light levels. The operation of dynamic circuits, i.e. those which use a pass transistor input into a gate having no other electrical connection, can be severely affected by light. Light induces leakage currents in the n-p junction between source and drain regions and the substrate. At room temperature, charge stored on dynamic nodes can be retained for many milliseconds in the absence of light. However, in normal room light the retention time is reduced to tens of microseconds. Thus care should be taken to avoid high light levels when long clocking periods are used. Dynamic memory chips are packaged in opaque black packages because of this effect.

A software blowback of the metal mask PG file of another project set, organized at Caltech, is shown in figure 23. The total area of this multi-project chip set is ~ 1 cm^2. It is subdivided into four major sections: The lower right quadrant contains the OM2 Data Engine described in Chapter 5, layed out using λ = 2.5μm. The upper right quadrant contains a 16 by 16 bit multiplier with on-board accumulator [by Rod Masumoto, Caltech], also using λ = 2.5μm. The lower left quadrant contains a subsystem, laid out using λ = 2.9μm, which converts output from one port of a computer memory into the red, green, and blue analog signals for driving a color TV monitor. The upper left quadrant contains 28 projects, mostly from students in an LSI Systems course at Caltech. Other small projects are located along the left edge of the multiplier, and in the unused area within the TV subsystem project. The source material for this project chip set was generated on three different computer systems, in two different languages. Check plotting and viewing were done on three other systems. In addition to the Caltech projects, this chip set contains projects from Carnegie-Mellon University, Washington University (St. Louis), University of California, Irvine, and the Jet Propulsion Laboratory. Approximately 500,000 pattern generator

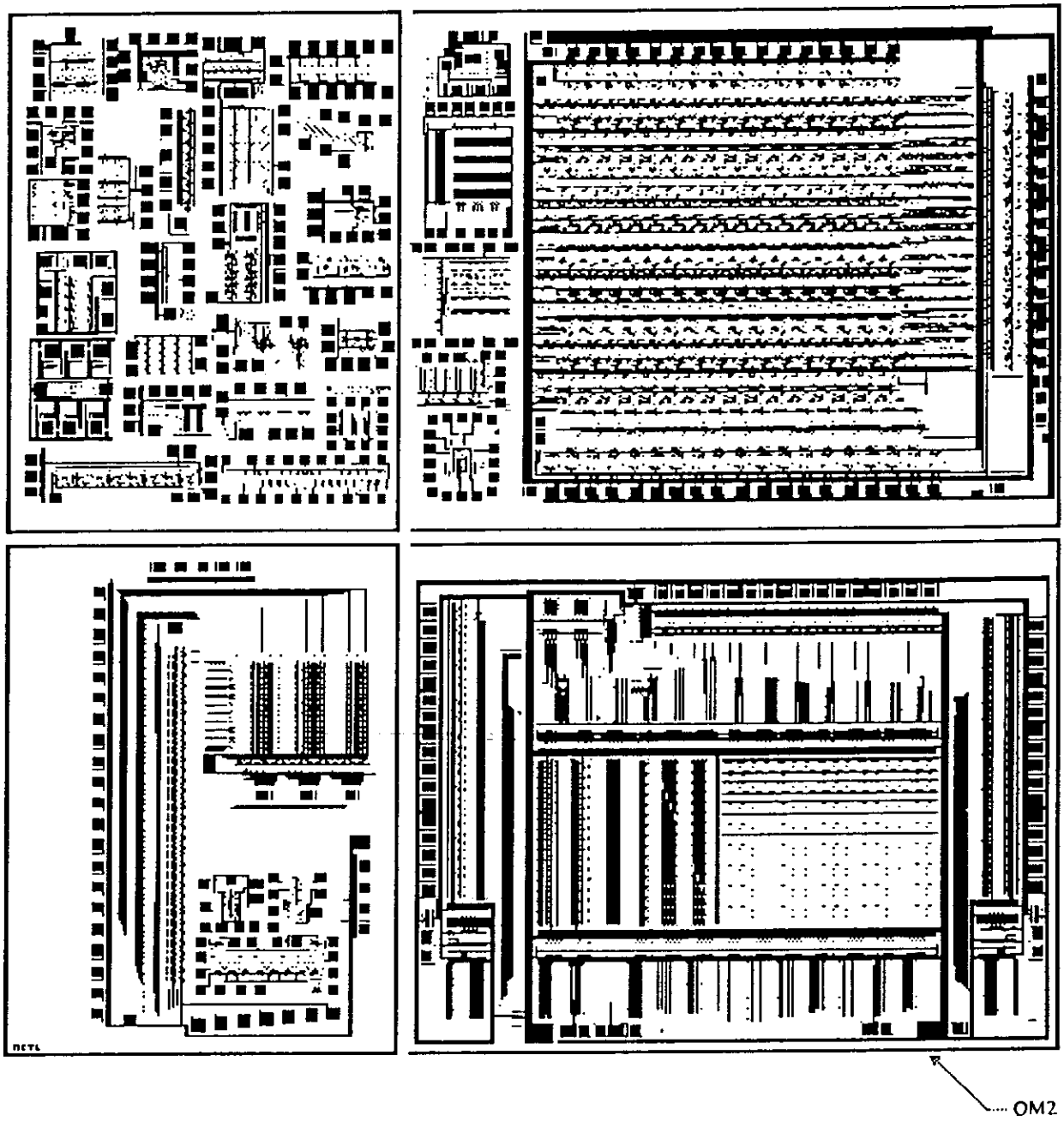


Fig. 23. Multi-Project Chip Set Organized at Caltech

rectangles were required to pattern the reticles for the five mask levels used in this project set. Conversion from intermediate form to PG files required ~10 CPU hours on the Caltech DECsystem 20.

The masks for the multi-project chip sets shown in figures 21 and 23 were produced by Silicon Valley mask houses from PG tapes, accompanied by PG file software blowbacks showing the locations of auxiliary layout items to be used during implementation, and by spec sheets containing a list of mask and fab specifications and parameters. These spec sheets contain two types of information:

(i) that which the mask house will need for reading the PG tape, generating the reticles, and stepping the master masks. This includes whether dimensions are in Metric or English units, whether fiducials and parity marks have been laid out or are to be placed by the mask house, desired reticle magnification (usually 10X, sometimes 5X), the x,y step and repeat distances, the type and magnification of reticle blowbacks desired, and whether maskmaking beyond reticle generation is to be contingent upon blowback inspection. This information is independent of the chosen fab line.

(ii) that which is specific to the fab line, or lines, on which the wafers will be fabricated. Examples here are the number, size, and type of working plates desired, and the photographic polarity of the working plates, i.e. whether they are a positive or negative image of the PG pattern. The polarity of the working plates depends on the process step and on whether positive or negative resist is used. In addition, it is customary to specify how much, if any, the lines in the image will be expanded or contracted to compensate for growth or shrinkage of regions due to the process. This so-called "pulling" of line widths in maskmaking may begin as far back as at pattern generation. Thus, while the patterning and fabrication processes are design and layout independent, they are usually coupled, and masks made for a run on one fab line are not necessarily useable elsewhere.

Maskmaking and patterning technology will remain in a state of transition for years to come. The present shift is from contact printing with working plates to projection alignment using original master masks. These two alternatives are illustrated in figure 24. From the system designer's point of view, at the interface to the mask and fab firms, they present no essential differences, requiring perhaps slightly different specs, and yielding different intermediate artifacts. In the next section we discuss the future evolution of these technologies, presenting several implementation schemes likely to become commonplace over

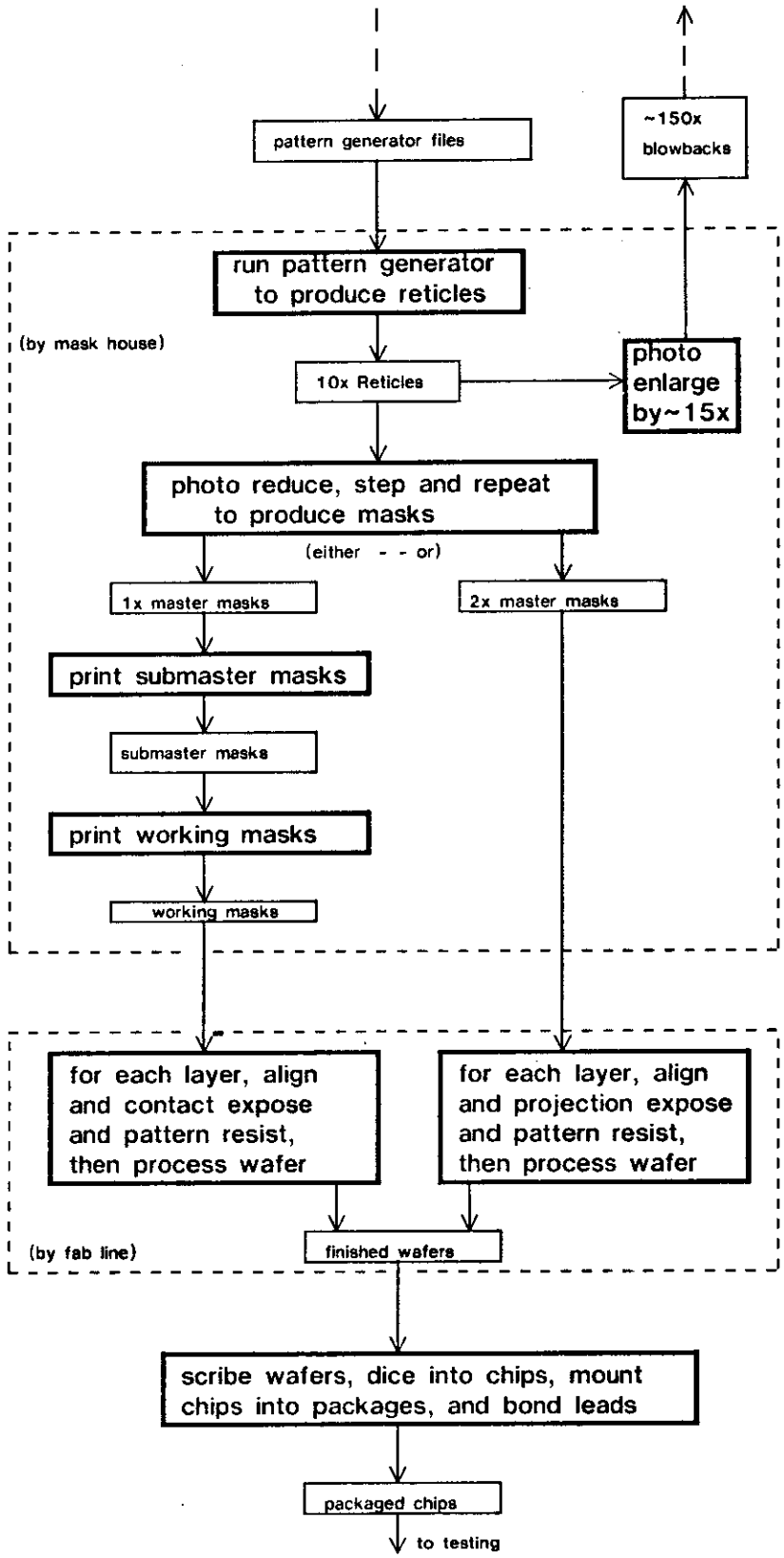


Fig. 24. Present Maskmaking and Fabrication: Two Alternatives

(prespf.sil)

the next decade. These schemes will enable fabrication of systems much denser and faster than present ones. However, the basic concepts of the design methodology will still apply. Remembering our film processing analogy, we will have "finer grain" and "faster" film available as time passes. However, the basic art of photography remains.

Patterning and Fabrication in the Future

As λ is scaled down toward its minimum value, ultimately limited by the physics of semiconductors to about 0.1μm, it will become feasible to implement single chip, maximum density VLSI systems of enormous functional power. Patterning and fabrication at such small values of λ requires that certain fundamental problems be overcome[4]. In this section we will discuss alternative solutions to two of the major problems: At values of λ of ~2 μm, a problem of *runout* is encountered, causing successive patterning steps to misalign over large regions of the wafers. This problem is solved by using less than full wafer exposure. At values of λ under 0.5 μm, the wave length of light used in photolithography is too long to allow sufficient patterning resolution. This problem is solved by using non-optical lithography, exposing the resist with electron beams or x-rays.

Historically, silicon wafers have been patterned using full wafer exposure, i.e. using masks which covered the entire surface of the wafer. The pattern for one layer of one chip is stepped and repeated during the fabrication of the mask itself, so that the mask contains the patterns for a large array of chips. During the fabrication of each successive layer on the wafer, that layer's mask is aligned at two points with the pattern already on the wafer, and the entire wafer then exposed through the mask. In the future, as feature sizes are scaled down, full wafer exposure will not likely be possible for reasons developed in this section.

The earliest integrated circuits, circa 1960, were fabricated using wafers of 2.5 cm diameter, and typical chips were 1 to 2 mm, with a minimum feature size of ~25μm. In 1978, production wafers are 7.5 to 10 cm, typical commercially manufactured LSI chips are 5 mm, and minimum feature size is ~5μ. The concurrent development of ever finer features sizes and larger wafer sizes has placed an increasingly severe strain on the process of full wafer exposure. The reasons lie in the physics of wafer distortion.

When a wafer is heated to a high temperature, it expands by an amount determined by the

thermal coefficient of expansion of silicon. A bare wafer will contract exactly the same amount upon cooling, and will therefore remain exactly the same size. Suppose, however, that a layer of SiO_2 is grown on the wafer when it is at the high temperature. The thermal coefficient of expansion of SiO_2 is approximately 1/10 that of silicon. As the wafer is cooled, the silicon will shrink at a rate much greater than that of the SiO_2. Normally the resulting wafer will not be flat, but convex on the SiO_2 side. If the wafer is cooled slowly enough, it is possible to "relieve" the stress induced by the difference in thermal contraction. Wafers in which such stress relief has been achieved are nearly flat but are, of necessity, a different size than they were originally[7a,b].

It might seem that subsequent masks could be scaled to just match the wafer distortion introduced up to the appropriate point in the process. Unfortunately no such correction can be introduced without a knowledge of the pattern of SiO_2 on the wafer. During cooling, dislocations are induced in the underlying silicon crystal at the edges of openings in the oxide pattern. Hence, the magnitude and direction of wafer distortion is dependent in complex ways upon the thickness and distribution of SiO_2 on the surface and upon the details of the thermal cycle. While it is in principle possible to compute a geometric correction for each pattern to be produced, it is clearly not possible to apply one correction for all possible patterns. Misalignment between subsequent layers due to distortion of this type is often referred to as *runout*. Runout due to wafer distortion is today the largest single contributor to misalignment between masking steps. Attempts to use finer feature sizes, which require more precise alignment, on larger wafer sizes, which induce larger distortions, seem doomed to failure unless full wafer exposure is abandoned.

Two attractive alternatives to full wafer exposure are now being explored: (i) electron beam exposure, and (ii) exposure using step and repeat of the chip pattern directly on the wafer.

A scanning electron beam system can be used to expose resist material, and is also capable of sensing a previous pattern on the surface of a wafer. The beam can initially scan an area covering the alignment marks of a particular chip. Information gained from this sensing operation can be used to compute the local distortion, and the chip can be exposed in nearly perfect alignment using these computed values. The process can be repeated for each chip on the wafer, until all have been exposed.

This technique has several virtues. No masks are required. A digital description of the chip can be exposed directly onto a silicon wafer. A different chip can be placed at each chip

location, and this opens up the possibility of greatly extending the multi-project chip concept. However there are also limitations. Data is transferred serially. Even at the highest data rates which can be conveniently generated, a long time is required to expose each chip. More fundamentally, the physics of electron beam interactions places severe restrictions on the minimum practical feature size attainable. When a beam of electrons enters a resist-coated wafer, scattering occurs both in the resist and in the wafer. This backscattering contributes a partial exposure at points up to a few microns away from the original point of beam impingement, and has a number of implications:

(i) The exposure, or spatial distribution of energy dissipation, varies with depth in the resist. Thus resist cross section is not readily controllable.

(ii) Exposure at any particular point depends on all patterns exposed within a few microns. This is known as the "cooperative exposure" or "proximity" effect and necessitates pattern-dependent exposure corrections[8].

(iii) Exposure latitude becomes narrower as the spatial period of a pattern is reduced. This is illustrated in figure 25, which shows the rise in background level exposure as a function of lateral distance for four different spatial periods: (a) 2μm, (b) 1μm, (c) 0.5μm, (d) 0.3μm. The beam diameter is 250 angstrom units, the energy 10keV, the resist thickness 0.4μm. The consequences of this background rise are particularly troublesome for high-speed, low-contrast resists. Experimental results show somewhat greater line broadening than predicted by the model[9].

For the above reasons, the writing time and the difficulty of exposing desired geometries increase rapidly for linewidths below about 0.5 micron[9].

An immediate prospect for achieving feature sizes of 1-2μm with large wafers is offered by stepping the chip pattern directly on the wafer rather than on a mask. This technique avoids the serial nature of the electron beam writing by exposing an entire chip at once. Using good optical systems it has been possible for many years to produce patterns with feature sizes in the range 1 to 2μm. Recent progress in the design of optical projection systems may even make 1/2 to 3/4 micron line width patterns over several millimeter diameter areas practical[10]. Techniques are known for using light to achieve alignments to a small fraction of a wavelength. Recently, an interferometric optical alignment technique has demonstrated an alignment precision of 0.02 micron and should be capable of a

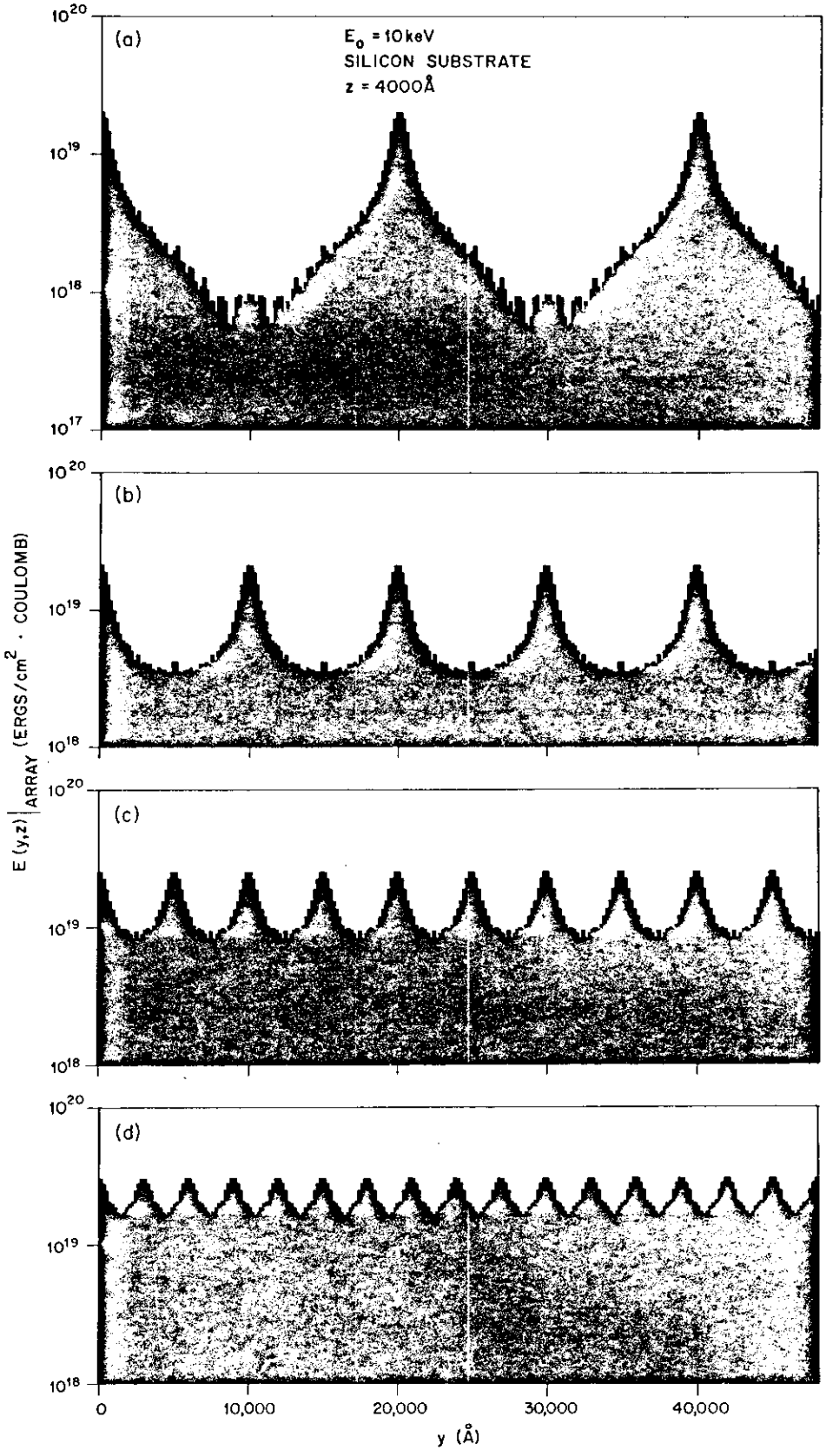


Figure 25. Electron Beam Exposure of Resist on Silicon.
Monte Carlo Calculation of Exposure Level at a Silicon-Resist Interface, as a Function of Lateral Distance, for Four Spatial Periods.
[contributed by H. I. Smith, Lincoln Laboratory, M.I.T.]

reregistration uncertainty less than 0.01 micron[11]. It would seem that devices of ultimately small dimensions (0.25μm) could be fabricated using optical alignment. It must be stressed that a realignment to the underlying pattern *must* be done at each chip location to achieve the real potential of the technique.

The step-and-align technique can be extended to ultimately small dimensions by substituting an x-ray source for the optical one, while retaining the automatic optical alignment system. X-rays require a very thin mask support, e.g. Mylar, upon which a heavy material such as gold or tungsten is used as the opaque pattern. Interactions of x-rays with matter tend to be isolated, local events. No back-scattering of the x-rays occurs, and electrons produced when an x-ray is absorbed are sufficiently low in energy that their range is limited to a small fraction of a micron. For this reason, patterns formed by x-rays in resist materials on silicon wafers are much cleaner and better defined than those attainable by any other known technique (see figure 26). X-rays of very high intensity can be efficiently obtained from the synchrotron radiation of an electron storage ring. The time required for exposing a chip with such a source is no more than that required at present using optical exposures. Both optical and x-ray techniques have the property that the total *exposure* time per wafer can be made independant of how much of the wafer is exposed at a step. Therefore, the only penalty in a step and align process is the time required for mechanical motion and alignment.

It appears that we have in hand all of the techniques for ultra fine line lithography, even on larger silicon wafers. Both electron beam and optical stepping work must, however, focus on local alignment as the crucial step in achieving high density, high performance LSI.

We now describe a production lithography system for ultimately small dimensions. A major component of the system is a 500 to 700 MeV electron storage ring, approximately 5 meters in diameter, shaped in the form of a many sided polygon. The electron beam within this storage ring is deflected at each vertex by a superconducting magnet. This deflection results in a centripetal acceleration of the electrons, and hence in an intense tangential emission of synchrotron radiation. The most important component of such radiation is soft x-rays in the 280 to 1000 eV quantum energy range (wavelengths of 0.004 to 0.001 μm). Such x-rays are ideal for exposing resist materials with line widths in the 0.1μm range[12,13].

One exposure station is fitted to each vertex of the storage ring. Each exposure station has an automatic optical alignment system for individual alignment of each chip[11]. Coarse

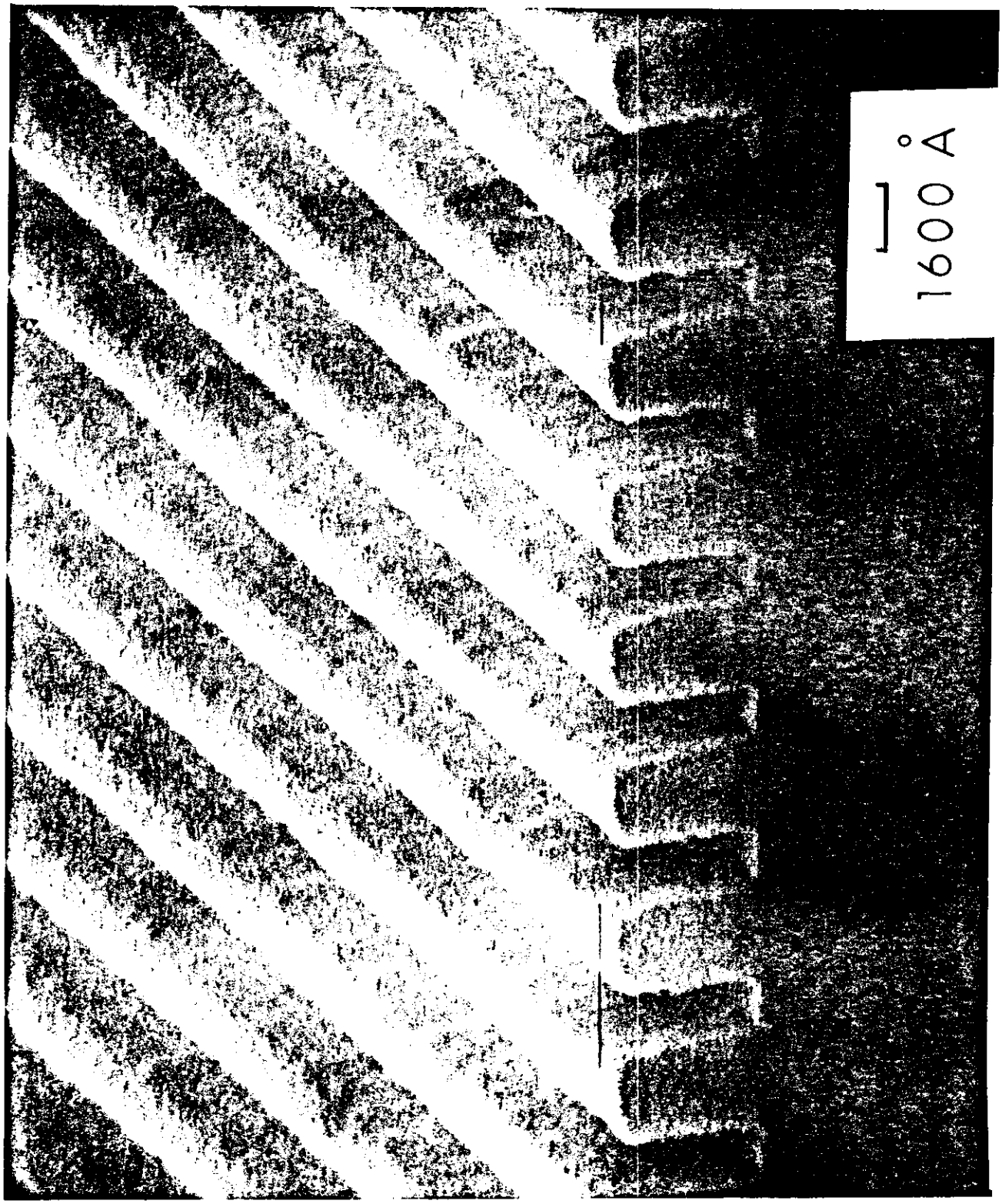


Figure 26. Resist on Silicon, Patterned by X-Ray Exposure.
[contributed by H. I. Smith, Lincoln Laboratory, M.I.T.]

alignment is controlled by a laser interferometer and the wafer brought into position by ordinary lead screws moving a conventional stepping stage such as those in current photorepeaters. Auxiliary alignment features are placed on each mask level within each chip. Misalignment of two such patterns on the wafer relative to those in the mask produces Moire patterns which are detected by photosensors and fed to a computer system. Piezoelectric transducers driven by the computer system bring the wafer into final alignment under the mask. Each exposure station in such a system is capable of aligning and exposing one layer of one chip every few seconds. Each chip may contain of the order of 10^7 devices, which is the equivalent of several *wafers* at today's scale.

An overview of the possible routes from design files to finished chips with sub-micron layout geometries is shown in figure 27. In the immediate future, alignments much better than those achievable today will be possible with the optical step and align technique (leftmost path in figure 27). In addition, this scheme eliminates the step and repeat process in mask making, enabling considerably shorter turnaround time. The rightmost path, direct electron beam writing on the wafer, promises the ultimate in short turnaround time. It can be viewed as using the fab area as a computer output device. For high volume manufacturing, at ultimately small dimensions, the center path as described above will most likely become the workhorse of the industry.

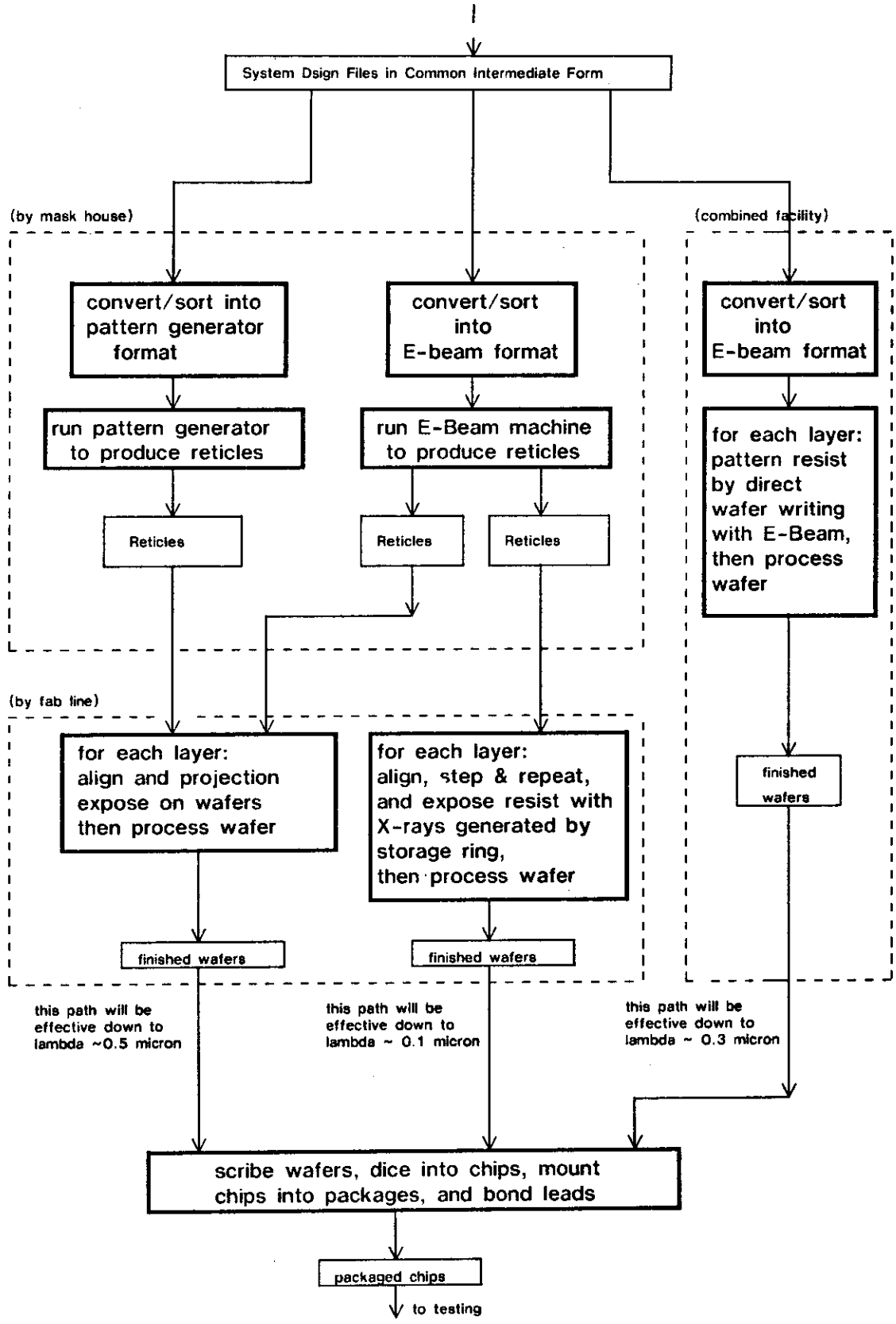


Fig.27 Some Future Patterning and Fabrication Alternatives

(futpf.sil)

Fully Integrated, Interactive Design Systems

{ in preparation }

- - - creating a data structure which allows the various levels of interactive processes to operate on the same data base - - - nodes, transistors, cells, and instances - - - operations on the data base - - - interactive logic transfer function tests - - - interactive circuit transfer function tests - - - interactive design rule checking - - - the filing problem - - -

System Simulation, Test Generation, and Testing

{ in preparation }

- - - system-level/register-transfer-level design description and simulation - - - testing the system design - - - practical strategies for structured VLSI system development - - - designing for testability - - - generation of test sequences - - - testing the chips - - -

References

1. D. G. Fairbairn, J. A. Rowson, "ICARUS: An Interactive Integrated Circuit Layout Program", submitted to 15th Design Automation Conf., IEEE, June 1978.

2. J. A. Rowson, "A Data Structure for Interactive Integrated Circuit Design", submitted to 15th Design Automation Conf., IEEE, June 1978.

3. A. C. Kay, "Microelectronics and the Personal Computer", Scientific American, Sept.1977, pp 210-228.

4. I. E. Sutherland, C. A. Mead, T. E. Everhart, "Basic Limitations in Microcircuit Fabrication Technology", ARPA Report R-1956-ARPA, November 1976.

5. C. A. Mead, "Ultra Fine Line Lithography", Display File #1179, Dec. 2, 1977, Department of Computer Science, California Institute of Technology.

6. GCA/D.W.Mann, "3600 Software Manual, Appendix B: 3600 Pattern Generator Mag Tape Formats", GCA Corporation, IC Systems Group, Santa Clara, Ca..

7a. I. A. Blech, E. S. Meieran, "Enhanced X-ray Diffraction from Substrate Crystals Containing Discontinuous Surface Films", J. App. Phys., vol. 38, pp. 2913-2919, June 1967.

7b. E. S. Meieran, I. A. Blech, "High Intensity Transmission X-ray Topography of Homogeneously Bent Crystals", J. App. Phys., vol. 43, pp. 265-269, Feb. 1972.

8. M. Parikh "Self Consistent Proximity Effect Correction Technique for Resist Exposure", to be published, J. Vac. Sci. Tech. Jan./Feb. 1978.

9. R. J. Hawryluk, H.I. Smith, A. Soares and A. M. Hawryluk, "Energy Dissipation in a Thin Polymer Film by Electron Beam Scattering: Experiment", J. Appl. Phys., vol. 46, pp.2528-2537, June 1975.

10. J. S. Wilczynski, "A Step and Repeat Camera for Direct Device Fabrication", Semicon East, Sept. 1977, Boston; M. Huques, M. Babolet "Lenses for microelectronics", and P. Tigreat, "Use in a Photodemagnifier", International Conference on Microlithography, Paris, June 1977.

11a. D. C. Flanders, H. I. Smith, S. Austin, "A New Interferometric Alignment Technique", App. Phys. Lett., vol. 31, pp. 426-428, Oct. 1977.

11b. S. Austin, H. I. Smith, D. C. Flanders, "Alignment of X-Ray Lithography Masks Using a New Interferometric Technique - Experimental Results", to be published, J. Vac. Sci. Tech., Jan./Feb. 1978.

12. B. Fay, et al.,"X-Ray Replication of Masks Using the Synchrotron Radiation Produced by the ACO Storage Ring", App. Phys. Lett, vol. 29, pp. 370-372, Sept. 1976.

13. E. Spiller, et al., "Application of Synchrotron Radiation to X-Ray Lithography", J. App. Phys., vol. 47, pp. 5450-, Dec. 1976.

14. N. Wirth, "What Can We Do about the Unnecessary Diversity of Notations for Syntactic Definitions?", Communications of the ACM, Nov. 1977.

Reading References

R1. J. J. Donovan, "Systems Programming", McGraw-Hill, 1972, an introductory text on this subject, provides practical information on the implementation and use of assemblers, macro-processors, compilers, loaders, operating systems, etc.

R2. W. M. Newman, R. F. Sproull, "Principles of Interactive Graphics", McGraw-Hill, 1973, is the classic text on interactive computer graphics, and is recommended reading for those interested in constructing interactive layout systems and fully integrated, display oriented, design systems.

R3. P. Freeman, "Software Systems Principles", Science Research Associates, Inc., 1975, presents the basics of a broad range of topics important in the building of software systems.