CHAPTER 2

IC DESIGN TOOLS

The key to fast-turnaround integrated circuit design is the emergence of powerful tools that make it possible for the designer to be effectively assisted by computers. For example, the availability of interactive graphics terminals allows him to enter ideas into a design system quickly. Layout languages are being developed that permit the entry of designs at a high level of abstraction. Computer generated displays and plots of the mask levels of an IC are important to close the man-machine loop and are indispensable in the final debugging phase. There are additional ways that the computer can assist the designer. A "mechanical" check for design rule violations helps eliminate potential problem spots in an IC design. Circuit simulators can be used to predict the performance of critical parts of the design and logic simulators can be useful in testing the correctness and timing of the overall chip.

In all these interactions with the computer, there is an underlying database that contains some description of the integrated circuit. This may be a circuit description, a layout topology description, or a description of the geometry of the masks required to fabricate the IC. The database may be more or less sophisticated, and designers can interact with it at different levels. At the lowest level it simply stores data in some internal format, with no "intelligent" processing. In this case the designer may have to enter his design as a low-level geometrical description, and the only thing that can be done with the information stored is display it on various output devices. At the other extreme, very sophisticated design tools may one day accept a high-level description of a particular system and compile a complete layout. Some of the IC designer's principal tasks are:

- Entering design geometry
- Outputting design geometry (plotting, printing)
- Documenting the design
- Checking for design rule violations
- Checking for logical errors
- Simulating the behavior of a design
- Testing the actual integrated circuit.

This chapter provides an overview over the types of tools available and under development for these various tasks, starting with the two elementary operations: inputting and outputting mask geometries. It should be pointed out that an ASCII keyboard and a lineprinter attached to a general purpose time sharing computer can support a set of design tools that has proven adequate for IC designs of substantial size. In general it is worthwhile to expend significant effort to develop your design tools. Even for an investment of several weeks, the return on your time will be quite high if you intend to do more than just one small design. Our experience has shown that in generating one's first designs it is not unreasonable to spend half of the time in enhancing (or creating) software tools.

2.1 Entering Your Design

The lowest level for entering the description of an IC is a point by point description of the geometry on each mask level. In the absence of any sophisticated design aids a detailed drawing of the design can be done with colored pencils on graph paper. From there the coordinates of the corners of all polygonal shapes can be read off and typed into a program that constructs some internal representation of the design for later plotting. A *digitizing table* is a means to input the coordinates of vertices directly from a drawing without the need for the explicit typing of numbers. A pointing device such as a stylus or a crosshair is placed on the crucial points in the drawing and the position on the digitizing table is automatically converted to a corresponding set of coordinates.

A *low-level* description of mask geometries that is used by IC designers at several universities is the Caltech Intermediate Form (CIF), described in Chapter 7. CIF has simple language constructs to specify elementary shapes such as rectangles, polygons, flashes, and wires. Since CIF is intended to be machine-generated, rather than human-generated (although it can and has been successfully used by designers without better tools), it is rather tedious to use the language to enter designs by hand. One alternative to entering CIF directly is to write a short program to provide an interactive input environment similar to a text editor. Such an input program could prompt for the next entry, provide suitable default values, warn of format errors, permit convenient iteration of cells, provide user selectable reference points and relative coordinates, and handle the management of the generated files. An example of an input interface, written in the language Pascal, has been developed at the University of California at Berkeley [Krause 1979].

A more efficient input device for layout geometries is an *interactive graphics editor*, engineered for the special needs of IC layout. The necessary hardware includes a pointing device, used in drawing the layout, and a graphics display to show the design as it is being entered. Ideally the system should be as easy to use as a pencil and paper when producing a first rough sketch, yet it can contain features to help the designer avoid mistakes or be more productive. For instance, it may generate layouts on a specified grid, with all edges by default parallel to the coordinate axes and all interconnection paths of a presettable default width (for example the Icarus system [Fairbairn & Rowson 1978]). Another approach is to allow the designer to work on a loose grid [Williams 1977]. This facilitates the easy repositioning of items, since only relative positions are indicated on the screen. Once the layout is completed, a program fills out features to default dimensions and compacts the geometric layout into a final mask specification. In either approach it should be easy to define parts of the display as a named cell that can be called at a later time to be inserted into different places in the design – if necessary, with transformations (rotation, scaling, mirroring) applied.

2.2 Hardcopy Output

Even when a system with interactive graphics terminals is available, the ability to produce hardcopy checkplots is essential for documentation and error checking. Hardcopy output allows the designer to paste together an overall view of a complicated design with enough resolution for error checking, marking design changes, and inserting comments. Good checkplots must show several mask levels simultaneously in such a manner that it is clear exactly which levels contain an active feature at any specific point. Such checkplots are most easily read if all mask levels give an impression of being semi-transparent. Filled-in color features have proven very effective and easy to read, however, color output devices are still quite expensive.

Inexpensive checkplots of low resolution can be obtained from an ordinary line printer using different characters to represent different layers and separate or overstruck characters to show overlapping layers [Gibson & Nance 1976, Larsen 1978]. A few hundred lines of code are sufficient to produce such a plot from any CIF file.

Color line-drawing plotters provide output of much higher information density. While a drawing of the polygonal outlines alone may be relatively hard to read, the readability can be improved if internal areas of the shapes on each layer are crosshatched in the appropriate color. Unfortunately this results in very long plotting times. Furthermore, intersecting shapes within the same layer, which during mask generation get merged into a single polygonal shape, may show all the internal boundaries as well, which makes these outline drawings rather confusing.

Electrostatic dot raster plotters (such as those offered by Versatec or Gould) generate high resolution, filled-in output in a matter of minutes. The various mask levels can be distinguished by different gray-pattern shading or by different stipple patterns. These stipple patterns should be properly selected so that any individual mask can be seen in the presence of any arbitrary combination of other mask levels. The outline of the structure of such a plotting program is presented in Section 7.3.

2.3 High-Level Descriptions

So far we have only dealt with the design at its lowest level description, the geometry of each mask level. In many instances the efficiency of the designer could be significantly enhanced if he or she could communicate with the machine at a higher level. A first step in that direction is to use a hierarchical organization in the design, subdividing the overall problem into more easily handled subproblems. In IC design the elements of the hierarchy are called *cells* or *symbols.* An *instance* or "use" of a cell can stand alone or be embedded as a subpart of another cell. This means that the contents of the cell are inserted into the design at certain points in much the same way that procedure calls are embedded within other procedures in an ordinary programming language. Such a hierarchical approach reduces the amount of information that has to be handled and stored explicitly, and is the key to modular debugging.

The next step is to provide a library of such cells with suitable parameters, for example a general adder cell that is called with a parameter specifying the width of the data path, or a PLA that takes as an input the array specifying the positions of all connection points in the two NOR planes. In the same spirit a set of generic cells could be called in conjunction with a *technology specification file*, which then returns a cell with the proper linewidths and registration tolerances for that particular technology.

Rather than specifying a design through geometrical shapes or parameterized layout cells, it would be preferable to specify the designers intent at an even higher level, for example by the equations for a block of boolean logic or by the state diagram for a finite state machine. For this purpose a description language and a compiler that produces a corresponding layout are required. In particular the PLA mentioned above could be called by the set of its logic functions, and the compiler would take care of the minimization and call the proper number and type of internal and peripheral cells to make up the PLA.

This brings up the point that for any design there exist a number of different *representations*: register diagram, circuit, layout topology, mask geometry, behavioral description or descriptive text. An advanced design system might contain the information for all or some subset of relevant representations in an integrated data base, so that the designer can view his design in a number of ways. Ideally this data base would automatically reflect changes to one representation in all the others.

A somewhat less ambitious system may have one master design representation that reflects the intent of the designer and in which modifications can be introduced. Other representations are then slaved to this master and could be automatically updated, interactively or by a separate compilation phase. A feasible design system of the near future may permit the designer to specify his designs at the register transfer level in a language like ISP [Bell & Newell 1971]. From that description general, parameterized cells, possibly represented by their topology in stick diagram form, are called from a library and fleshed out to the proper dimensions based on a design rule or technology file. The resulting blocks may be further adjusted in size for the best fit [Johannsen 1979] and then be submitted to an automatic routing program that wires the blocks together. After that, actual mask geometries can be generated, from which the values of parasitic circuit elements can be calculated. These values together with connectivity information stored in the data base form the input for a simulation program. To check the completed design, the output from this simulation is compared to the behavioral description of the overall design at the register transfer level.

The next higher level of input specification should include features such as user-definable subroutines, parameterization of objects and conditional branching. With such features the layout language has the power of a general purpose programming language plus all the specifically built-in features that make it suitable for layout purposes. In particular, the parameterization of objects permits one to base the position, size or shape of a geometrical feature on the position or size of some other feature. Thus, for example, objects could then be moved around while signal paths or wires remain connected to the proper points. Furthermore, the position of interconnection points could adjust themselves to match the corresponding locations in adjacent cells, and power supply lines could automatically be widened so that they can handle the total current required by the connected cells. Layout languages can, in principle, be implemented in any computer language (see the description of ICLIC in [Stone 1978]).

In conclusion, the complexity of VLSI designs forces us to develop the necessary tools to communicate with the machine at a much higher level than was traditionaly possible. Automatic compilation of various representations will dramatically reduce the occurrence of errors and keep design time within acceptable limits.

2.4 Design Rule Checking

[contributed by Wayne Wilner, Xerox PARC]

Where automated design systems are not available and the mask geometries are generated by a human designer using low-level tools, the chance for errors and design rule violations is high. Thus, careful checking of the layout is mandatory. In the absence of any checking programs, the designer checks his layout by inspecting the hardcopy output plots. While this may sound like an impossible task for designs with more than a few hundred features, there are a few things that come to the aid of the designer. Humans possess tremendous pattern recognition power. Each mask level of a modular, densely packed IC has a surprising amount of structure and regularity, and design rule violations such as lines that are too narrow or too close together stand out clearly. Of course, any intentionally introduced regularity, such as the use of arrays of the same cell, also facilitate checking. It may be astonishing, but even people who do not understand the details of a particular design can readily detect certain violations on a checkplot. To check rules that concern more than one level, it is advisable to jointly plot the levels involved for easy inspection. Combinations of particular importance are poly-diffusion-cut and metal-cut. In general every design should be carefully inspected by at least one other impartial designer.

The mechanical nature of design rules allows some checking to be performed by computer. Starting from a CIF file, the various shapes in the same mask level that touch or overlap are merged. Subsequently the positions of suitably oriented edge vectors of all polygonal shapes can be compared to one another to check for adequate distances. For large layouts a suitable ordering and grouping of all features is necessary to prevent the compute time from increasing with the square of the number of features.

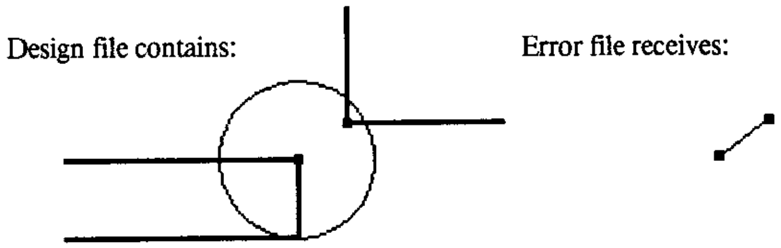
Error-checking programs embody the rules for a particular process and examine CIF files or pattern generation tapes for violations, reporting each instance in terms of coordinates or patterns, along with the nature of the violation. Design rules typically assign minimum distances to:

dimensions of features, such as breadth of runs or size of contact cuts;

spacing between features in the same layer, such as distance between runs;

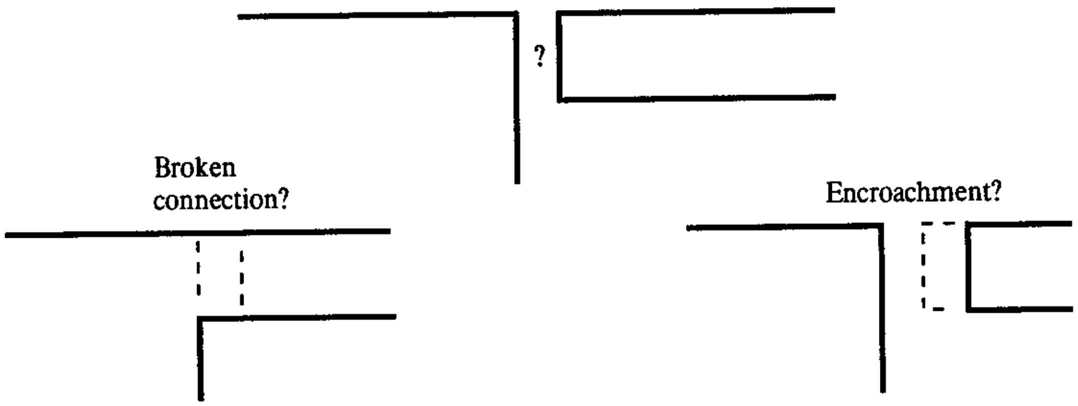
spacing between features in different layers, such as overlap of metal and contact windows.

For example, suppose unconnected areas of polysilicon must be 2λ apart. In the diagram below, a circle of radius 2λ centered at the upper right corner of the left-hand area reveals that the right-hand area is too close.



This design rule violation may be reported in terms of a line segment, that is, two points, one at the periphery of each area, and their (insufficient) separation. It is a non-trivial problem to present violations to the designer in the most convenient way. The output from such a program could simply be a list of the positions and levels of all pairs of vectors violating a particular design rule. Alternatively an extra error "mask" level, which can be superimposed on the regular layout plots to indicate the location and type of violation, may be easier to read.

An inherent limitation of design rules comes from their pertaining solely to the lowest level of detail. Consider the following diagram. Two areas are separated by less than their minimum spacing.



It is clearly a design rule violation, but is it a broken connection or is it an encroachment? The designer will have to decide and fix it appropriately.

In laboratories that experiment with different processes, the critical distances may vary from month to month. In experiments with custom circuits, the objective may be to find how exceptions to the design rules can be exploited. Therefore, while the types of rules may be rigidly bound into a checking program, specific distances should be parameterized. The difficulty with such an approach is that design rules used in an industrial environment are often rather complicated, involving conditions such as: "polysilicon must extend at least four microns beyond diffusion, unless the gate dimensions are small, where six microns are required". It is an unsolved problem to mechanically create a program that can efficiently verify geometrical constraints of varying nature. For the much simpler design rules used throughout *Introduction to VLSI Systems* [Mead & Conway 1980] the problem is tractable.

2.5 Checking for Other Errors

Many fatal errors in a design do not exhibit themselves as violations of design rules. Consider logic errors, state machines that are initialized to terminal states, or transistors that are wired incorrectly, but within the given design rules. Consider an array of cells that are supposed to abut exactly, but are either separated or overlapping; if the space between them is larger than the minimum spacing for all layers, design rules may be observed while the array is grossly in error. Consider the placement and continuity of busses. These errors are representative of flaws for which the designer is singularly responsible.

Many such errors can be spotted on relevant subsets of layers. A plot of metal and contacts can reveal errors in continuity which would otherwise be lost in the details of a full plot. Alternatively, a plot of poly, diffusion, contacts and implants enables one to check their important overlaps. This technique is very effective if the plots are large, clean, and of high contrast.

Hardest to find are connections that are almost correct such as a connection to the Q-bar output instead the Q output of a flipflop, or a connection to the wrong bit in a wide bus. In particular, wide-ranging interconnections between cells are error prone since they are easily lost in the jumble of a large checkplot. Errors of that kind must be detected by logical checks or simulation. In manual debugging, such errors are often found if the actual signal path is traced with a colored pencil on your hardcopy output.

Automatic compilation of a layout from previously defined and debugged cells will drastically reduce the number of errors. But even with such a structured approach, rather basic errors may occur if adjacent cells are either improperly placed or if the individual cells were not specifically designed to tolerate arbitrary neighbors without creating a design rule violation. In manual debugging, cell and array boundaries and all interconnection points should be checked with particular care.

2.6 Simulation as an IC Design Tool

[contributed by Richard Lyon, Xerox PARC]

Simulation is a design technique widely used in a variety of engineering disciplines. When it is too difficult to verify the correctness of a design by inspection, by proof, or by test, simulation may help. Simulation allows the designer to test a design before building it, by modelling in detail the components from which the design is built, and by computing their interactions under various conditions. Simulation is useful at many levels in integrated circuit and system design; system-level, register-transfer-level, logic-level, and circuit-level simulators are useful at various stages of the IC design process. A related activity is the design of IC fabrication processes, which can benefit from process simulation; the simulation of process variations may become more important as VLSI approaches the physical limits of device sizes, where the set of devices used by the system designer must be carefully matched to the technology.

Unfortunately, not many generally useful simulators are readily available. Even when such a program is available to run on your computer, the problem of preparing data in a form suitable to the simulator can be formidable. It is easy to write a register-transfer-level simulator, for example, but the part that would make it useful, an automatic link from the design language to the simulator input language, is much harder to implement. The lack of commonality of design methods in the digital system design field have resulted in a delay in the availability of such programs. In the circuit design field, on the other hand, the method of design has traditionally been standardized to drawing by hand on paper the interconnection of standard types of lumped circuit elements. From here it is logical to assume hand translation to the language of a circuit simulator. For this reason, circuit simulators are widely available in standard languages (Fortran IV). Two such simulators, somewhat tailored for IC simulation, are SPICE from U. C. Berkeley, and MSINC from Stanford. Their input languages are similar, and one example should serve to illustrate both.

As an example, we have simulated the output pad driver called *PadOut*, which was designed in *Icarus* (Integrated Circuit ARtwork Utility System, an interactive layout design system) according to the Mead and Conway design rules, with lambda equal to 3 microns. This is a driver intended to interface NMOS chips to other popular logic families, at speeds and voltages comparable to TTL. It uses push-pull enhancement-mode output drivers, driven in turn by *super-buffers* (see [Mead & Conway 1980] chapter 1). The fanouts are generally somewhat higher than the theoretical optimum of *e*, to reduce space and power at the expense of speed. Figure 2.6.1 is the Icarus layout picture of PadOut; notice that the output transistors are both wrapped around the pad. The schematic diagram is shown in Figure 2.6.2; it includes node numbers and element names which are needed for translation to the simulator input language.

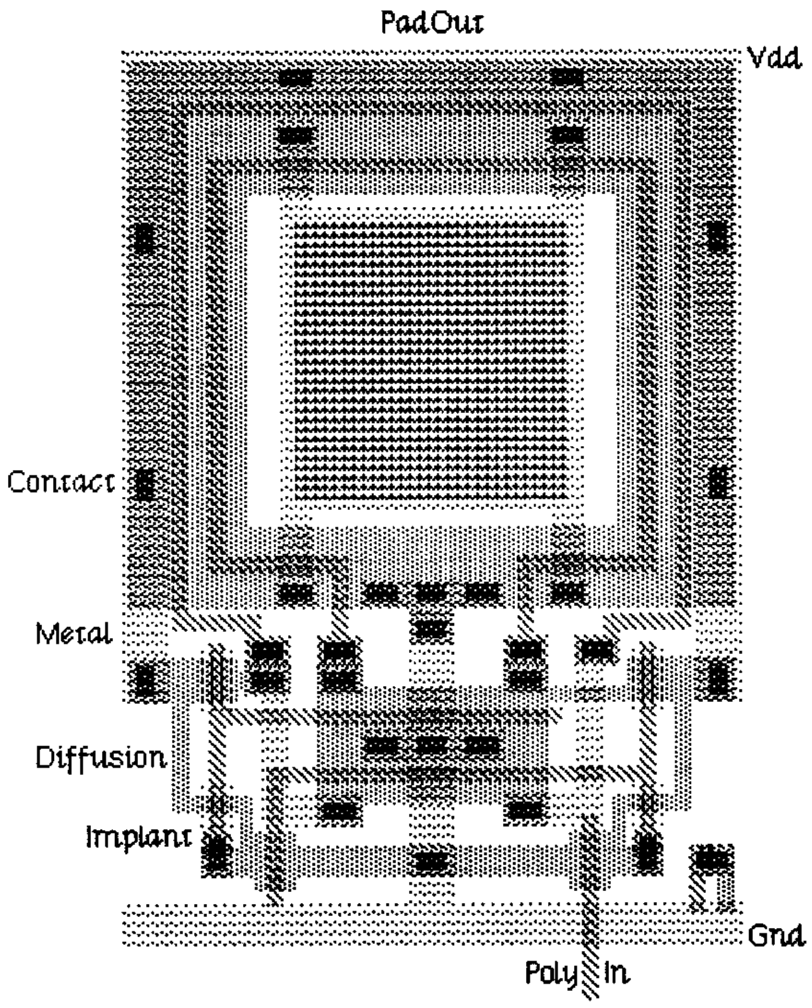


Figure 2.6.1. Icarus Layout of "PadOut"

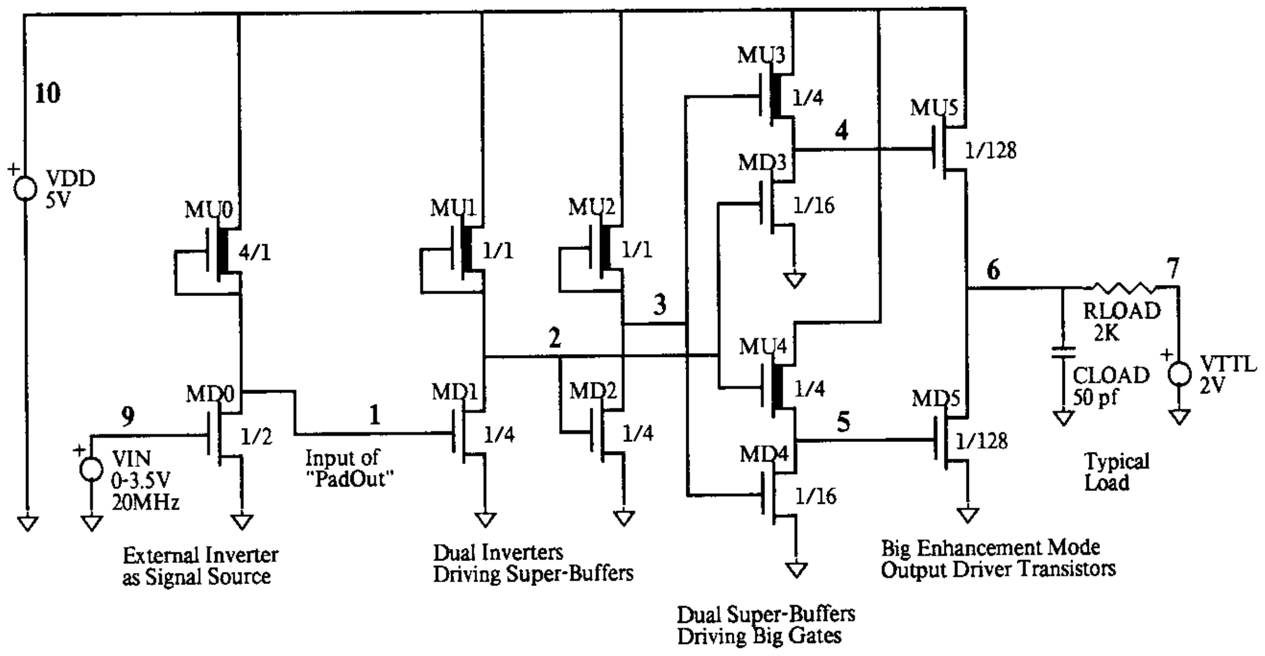


Figure 2.6.2.

Diagram of "PadOut" Output Pad Driver with element names and node numbers for simulation with SPICE.

```
PAD DRIVER SIMULATION
* RF LYON -- JULY 13, 1978
*
VDD 10 0 DC 5VOLTS
VTTL 7 0 DC 2VOLTS
VIN 9 0 PULSE 3.5VOLTS 0VOLTS 2NS 2NS 2NS 23NS 50NS
*
MD0   1 9 0 0 ENH W=12E-4 L=06E-4 AS=144E-8 AD=144E-8
MU0  10 1 1 0 DEP W=06E-4 L=24E-4 AS=144E-8 AD=144E-8
MD1   2 1 0 0 ENH W=24E-4 L=06E-4 AS=144E-8 AD=144E-8
MU1  10 2 2 0 DEP W=06E-4 L=06E-4 AS=144E-8 AD=144E-8
MD2   3 2 0 0 ENH W=24E-4 L=06E-4 AS=144E-8 AD=144E-8
MU2  10 3 3 0 DEP W=06E-4 L=06E-4 AS=144E-8 AD=144E-8
MD3   4 2 0 0 ENH W=96E-4 L=06E-4 AS=600E-8 AD=600E-8
MU3  10 3 4 0 DEP W=24E-4 L=06E-4 AS=144E-8 AD=144E-8
MD4   5 3 0 0 ENH W=96E-4 L=06E-4 AS=600E-8 AD=600E-8
MU4  10 2 5 0 DEP W=24E-4 L=06E-4 AS=144E-8 AD=144E-8
MD5   6 5 0 0 ENH W=768E-4 L=6E-4 AS=4000E-8 AD=4000E-8
MU5  10 4 6 0 ENH W=768E-4 L=6E-4 AS=4000E-8 AD=4000E-8
CLOAD  6 0   50P
RLOAD  6 7   2K
*
.MODEL ENH NMOS (NGATE=1E20 TPS=1 XJ=1E-4
+ CGD=4E-12 CGS=4E-12 CGB=2E-12 TOX=95E-7
+ NSS=-22E10 NSUB=8E14 )
.MODEL DEP NMOS (NGATE=1E20 TPS=1 XJ=1E-4
+ CGD=4E-12 CGS=4E-12 CGB=2E-12 TOX=95E-7
+ NSS=80E10 NSUB=8E14 )
*
.TRAN 1.0NS 80NS
.PLOT TRAN V(1) V(2) V(3) V(4) V(5) V(6) (0,8)
.WIDTH OUT=72
.END
```

Figure 2.6.3. SPICE Input Deck for *PadOut*

The simulator SPICE was used at Xerox PARC, on the MAXC2 computer, which has no floating-point hardware; therefore, the execution of the Fortran program was blindingly slow. Figure 2.6.3 shows the *input deck*, an ASCII text file. The SPICE program, like most widely available programs, was written for the card-reader/line-printer/batch-computing environment which is found at the typical university computing center. Therefore, be careful of input formats; only 72 columns of 80-column cards are used – long lines use continuation marks in column 1, as in Fortran. The documentation is sparse, but keep in mind that you should not do anything you could not do on a keypunch, such as lower case letters. See *User's Guide to SPICE* by E. Cohen and D. O. Pederson, from U. C. Berkeley Dept. of Electrical Engineering and Computer Science.

In the input listing, each line is called a *card*. The first line is the *title card*, and lines starting with * are *comment cards*. Each *element card* names a component (the first letter of the name determines the element type, such as M for MOSFET), tells what nodes it is connected to (in order, such as drain, gate, source, substrate), and gives a few parameters (such as width and length in centimeters). There are also *model cards* and *control cards*, which will not be described here, but can be seen in the listing.

We have described in the element cards the circuit of Figure 2.6.2 (some of the parameters are estimates, such as AS and AD, areas of source and drain). The first inverter is not part of PadOut, but represents a typical signal source, which is in turn driven by a 3.5 volt, 20 Mhz square wave generator with 2 nsec rise and fall times.

The output file produced by SPICE from the input shown was too long to include here. The most interesting part of it is shown in Figure 2.6.4, the graph of the time response of the various nodes, which is plotted line-printer style by typing the node numbers in appropriate columns. To make it readable, take a bunch of colored markers and draw in the curves for the nodes of interest. You will see that the response from node 1 to node 6 is noninverting, with t_{PLH} of 13 nsec and t_{PHL} of 9 nsec, measured at a 2 volt threshold (or more nearly symmetrical at 11 nsec if measured somewhere below 1 volt).

Is PadOut really this fast? Probably not on most processes; the model cards used here have estimates of the Spice model parameters which were felt to be realistic, but which gave results that are probably too optimistic for most typical 1978 processes. The inverter-pair delay from node 1 to node 3 is seen to be 6 nsec, where the inverter ratios are $k=4$ and the fanouts are $f=5$ (actually 6 for the first inverter). The delay estimate according to [Mead & Conway 1980] is then $(k+1)f\tau=25\tau=6$ nsec, so we may conclude that we have simulated a process with $\tau=0.24$ nsec (transit time), which certainly is optimistic. The actual performance of PadOut will have to be determined by test, and will depend on where it is fabricated; some lines would be three times slower than this simulation.

Circuit simulation can be very useful to the integrated circuit/system designer if it is applied to those problems that require it, but should not be relied on to verify the correctness of a complicated system design. In digital system design with a consistent design philosophy, it is usually possible to

```
           0.000E-01                0.000E+00                4.000E+00                6.000E+00
              - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* 0.000E-01   .61  X                   .                        .           X            .
* 1.000E-09   .61  X                   .                        .           X            .
* 2.000E-09   .61  X                   .                        .           X            .
* 3.000E-09   .X   X                   .                        .            X           .
* 4.000E-09   .6 1 X                   .                        .           52           .
* 5.000E-09   .6   X    1              .                        .            X           .
* 6.000E-09   .6   X          1        .                        .           X            .
* 7.000E-09   .6   X                  1.                        .          25            .
* 8.000E-09   .6    X                  .   1                    .       2   5            .
* 9.000E-09   .6    X                  .        1               . 2         5            .
* 1.000E-08   .6     X                 .            1      2    .           5            .
* 1.100E-08   .6       X               .          2     1       .           5            .
* 1.200E-08   .6           4  3        . 2                1     .           5            .
* 1.300E-08   .6                 24   3.                   1    .         5              .
* 1.400E-08   . 6         2            .    4    3          1   .   5                    .
* 1.500E-08   .  6     2               .             4   53   1 .                        .
* 1.600E-08   .     62                 .   5                  431                        .
* 1.700E-08   .      2  6     5        .                        .31  4                   .
* 1.800E-08   .      2 5     6         .                        .  31     4              .
* 1.900E-08   .      X           6     .                        .     X      4           .
* 2.000E-08   .     25               6 .                        .      13       4        .
* 2.100E-08   .     X                  . 6                      .       1 3      4       .
* 2.200E-08   .     X                  .    6                   .        1 3      4      .
* 2.300E-08   .     X                  .      6                 .        1 3      4      .
* 2.400E-08   .     X                  .       6                .         1 3    4       .
* 2.500E-08   .     X                  .         6              .         1 3    4       .
* 2.600E-08   .     X                  .          6             .          13   4        .
* 2.700E-08   .     X                  .           6            .          13   4        .
* 2.800E-08   .    26                  .            6           .           X  4         .
* 2.900E-08   .    X                   .            6     1     .           3  4         .
* 3.000E-08   .     5   1 2            .             6          .            3 4         .
* 3.100E-08   .  1    5              2 .             6          .           3 4          .
* 3.200E-08   .  1       5             .       2     6          .      34                .
* 3.300E-08   . 1           5          .             62      3 4.                        .
* 3.400E-08   . 1                  5   .      3    4 6      2   .                        .
* 3.500E-08   . 1            3        4.      5    6           2.                        .
* 3.600E-08   . 1     3     4          .        6      5        .2                       .
* 3.700E-08   . 1    3 4               .    6                5  .  2                     .
* 3.800E-08   . 1   43                 6                        5     2                  .
* 3.900E-08   . 1   X             6    .                        .5      2                .
* 4.000E-08   . 1   X        6         .                        .  5     2               .
* 4.100E-08   . 1  43    6             .                        .   5     2              .
* 4.200E-08   . 1  43 6                .                        .    5     2             .
* 4.300E-08   . 1  4X                  .                        .     5     2            .
* 4.400E-08   . 1 643                  .                        .      5    2            .
* 4.500E-08   . 16 X                   .                        .       5   2            .
* 4.600E-08   . X  X                   .                        .        5  2            .
* 4.700E-08   . X  X                   .                        .         5 2            .
* 4.800E-08   .61  X                   .                        .         5  2           .
* 4.900E-08   .61  X                   .                        .          5 2           .
* 5.000E-08   .61  X                   .                        .          5 2           .
* 5.100E-08   .61  X                   .                        .          5 2           .
* 5.200E-08   .61  X                   .                        .           52           .
* 5.300E-08   .61  X                   .                        .           52           .
* 5.400E-08   .6  1X                   .                        .           52           .
* 5.500E-08   .6   X     1             .                        .           52           .
* 5.600E-08   .6   X             1     .                        .           X            .
* 5.700E-08   .6   X                   1                        .         2 5            .
* 5.800E-08   .6    X                  .     1                  .     2     5            .
* 5.900E-08   .6    X                  .          1             2           5            .
* 6.000E-08   .6      43               .             1  2       .           5            .
* 6.100E-08   .6         43            .        2       1       .           5            .
* 6.200E-08   .6              4  3     2                  1     .          5             .
* 6.300E-08   .6                2    4 . 3                  1   .        5               .
* 6.400E-08   . 6         2            .       4    3         1 . 5                      .
* 6.500E-08   .   6   2                .              5 4   3  1.                        .
* 6.600E-08   .      X                 5                        X1                       .
* 6.700E-08   .      2   65            .                        . 31  4                  .
* 6.800E-08   .      25       6        .                        .    31    4             .
* 6.900E-08   .      X             6   .                        .      13      4         .
* 7.000E-08   .     X                 6.                        .       1 3      4       .
* 7.100E-08   .     X                  .   6                    .        1 3      4      .
* 7.200E-08   .     X                  .     6                  .         1 3      4     .
* 7.300E-08   .     X                  .       6                .         1  3     4     .
```

Figure 2.6.4.
SPICE Output

identify the critical parts of the design (for example the longest chain of pass transistors, the new RAM cell, or the node with the highest fanout); in this way, critical parts can be identified for simulation (see [Mead & Conway 1980] Chapters 1 and 7 for information on critical timing; see Chapter 4 for more on simulation and testing). Of course, even simulation will not verify that the design will run fast enough if the simulation parameters and models do not realistically reflect the process used to make the circuit.

IC designers have relied on simulation as a design tool for years. When the performance of a part being designed is critical (as is typical in manufacturing for sale), and the production/test turnaround is slow (also typical in the IC manufacturing business), circuit simulation is a necessity. However, in the topological design phase of a digital system, circuit simulation is not really helpful. By following strict design conventions and by employing relaxed design rules and conservative clocking schemes, it may be possible to design complete systems on a chip with only minimal use of circuit simulation. And, if turnaround is fast, an actual measurement may be a better way to determine performance than simulation is. A truly useful tool in this context would be a simple *logic-level* or *switch-level simulator*, working directly from the actual mask information [Bryant 1979]. Such a simulator, if able to handle the whole design at once, would be an invaluable help in finding logic errors and misrouted interconnections. The real task for the near future is to integrate simulation tools with design languages and layout programs.

2.7 Designing for Testability

Chip testing is an issue that should not be postponed until the wafers are delivered. While there exist sophisticated debuggers for pieces of software, there are no equivalent tools for chips. No hard and fast rules exist for designing a chip so that it can be effectively tested. One can only apply common sense and heed a few caveats.

A clean, modular design is a big asset in the testing phase. If the system is composed of a number of blocks, it may be possible to isolate each one as a separate project, complete with its own input and output pads. Thus, for instatnce, one could independently test the memory and the finite state control portions of a system. While the blocks themselves are not likely to be much use in their unbundled state, if each block is tested separately one need only correctly connect them together to construct a working system. The same principle applies to key cells in a repetitive design. If a novel memory cell is being tested, a single cell should be provided in isolation. The designer can then verify that the fundamental cells work (or not), even if the complete array doesn't. Testing small or moderately sized pieces of a system has the additional advantage that yield considerations are less significant.

Once the blocks are connected together it is still useful to have access to internal state information. One way to solve the problem is to provide internal test points (pads), with the intention of probing them after the chip is packaged. This approach requires sophisticated probing equipment and is quite risky. The circuit and bonding wires surrounding the probe points are easily damaged if the probe shifts. Probe cards for such an arrangement are expensive, and it may not even be possible to construct such cards unless the layout of the probe points is carefully considered. A more appropriate solution is for the designer to provide standard output pads and drivers that monitor key nodes in the system. These pads are bonded in the usual fashion and allow easy access to the internal signals. Since there may be a number of signals of interest, the pad count may be reduced by using a single output pad driven by a shift register that is parallel-loaded.

Carefully designed input stimuli in conjunction with thoughtful circuit design can go a long way toward qualifying an IC. For example, in testing an ALU, a particular output response following the execution of a certain sequence of instructions may assure the designer that registers x, y, and z and data bus q are all functioning correctly. If each internal section of a system can be tested in this way, one can be confident in the correctness of the chip. In the event that the project does not work at all, process test patterns (see Section 6.2) containing simple transistors and inverters can provide reassurance that a minimum level of process quality has been achieved.

The importance of including support circuitry on chip should not be overlooked. With a system that requires multiple clock phases and several lines of input and output, massive quantities of time can be consumed debugging the tangle of wires and auxiliary instruments that comprise the test set-up. Such set-ups are fragile, subject to noise pickup and a serious liability when trying to measure the performance of systems. Simple support circuitry, for example a two-phase non-overlapping clock generator, can reduce the confusion to more tolerable levels. This assumes that the designer is not debugging the support circuits along with his design.

References

[Bell & Newell 1971]
C. G. Bell and M. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, 1971.

[Bryant 1979]
R. E. Bryant, "MOSSIM: A logic-level simulator for MOS LSI", MIT Lab for Computer Science, September 1979.

[Fairbairn & Rowson 1978]
D. Fairbairn and J. Rowson, "ICARUS: An Interactive Integrated Circuit Layout System", *Proceedings of the 15th Annual Design Automation Conference*, June 1978.

[Gibson & Nance 1976]
D. Gibson and S. Nance, "SLIC – Symbolic Layout of Integrated Circuits", *Proceedings of the 13th Annual Design Automation Conference*, June 1976.

[Johannsen 1979]
D. Johannsen, "Bristle Blocks: A Silicon Compiler", *Proceedings of the Caltech Conference on VLSI*, January 1979.

[Krause 1979]
J. Krause, "CIF Interface", ERL Report, Dept. EECS, U.C. Berkeley, 1979.

[Larsen 1978]
R. P. Larsen, "Symbolic Layout System Speeds Mask Design for IC's", *Electronics*, Vol 51, No. 15, July 20, 1978.

[Locanthi 1978]
B. Locanthi, "A Simula Package for IC Layout", Display File #1862, Computer Science Department, California Institute of Technology.

[Mead & Conway 1980]
C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA., 1980.

[Stone 1978]
M. Stone, "IC Design Under ICL, Version 1.0.", SSP File #1336, Computer Science Department, California Institute of Technology, February 1978.

[Williams 1977]
J. Williams, "Sticks — A New Approach to LSI Design", Master's Thesis, Massachusetts Institute of Technology, June, 1977.