A Chip Assembler

by

Gary M. Tarolli

VLSI Advanced Development
Digital Equipment Corporation
Maynard, Massachusetts

presented at the International Workshop:

Problem Areas in
Digital Systems Description
and Design Tools

May 9-10, 1980
La Baule, France

sponsored by IFIP, Working group 10.2

SSP FILE #3882

INTRODUCTION

Most companies experience exponential growth in the cost of designing Very Large Scale Integration (VLSI) circuits as the number of transistors per chip increases (Figure 1). VLSI describes circuits consisting of more than 50,000 transistors on a single chip. Regular, structured design styles decrease design time and cost. To reduce further the rise in design cost, we turn over to computers the task of managing the information associated with integrated circuits.

Current techniques for specifying layout geometries are analogous to machine language coding in the early days of computers. Most current layout tools deal in absolute coordinates, with little or no parameterization. The Chip Assembler, a new chip design tool, deals with function blocks that are positionally-independent layout structures. The Chip Assembler can assemble these blocks together to hierarchically form a chip.

For the same reason that assemblers, compilers, operating systems, and structured coding conventions appeared, so are chip assemblers, silicon compilers, and regularly structured design styles appearing today. The average number of lines of microcode produced in one man year is about 1000. The numbers for Assembler and FORTRAN coding are also about 1000. But 1000 lines of a high level language does more than 1000 lines of a low level language.

What we are trying to do is provide the chip designer with a higher level language. Layout designers currently draw about 1000 gates per year. They digitize layouts and circuit schematics with little or no parameterization. If instead of 1000 gates per year, they put together 1000 datapaths per year, then they would be much more productive. The Chip Assembler offers a higher-level language to make such productivity possible.

For example, if you wanted to connect two lists of 16 connectors together on a graphic workstation, you would have to individually route the 16 wires from one list to the other (Figure 0). This may take 100 or more commands. In the Chip Assembler one command performs this function. In addition to routing, the Chip Assembler can also perform type checking on the connections it made. This support yields a much higher level of productivity.

REGULAR STRUCTURED DESIGN

Structured designs are those in which the physical hierarchy is the same as the structural hierarchy. The physical hierarchy represents the geometric symbol nesting as shown in Figures 2-5. The structural hierarchy could also be called the functional hierarchy and contains such items as ALUs, Datapaths, Programmable Logic Arrays (PLAs), and Register Files (Figure 6). A chip is structured when these two hierarchies are identical.

Since the hierarchies are identical, well defined interfaces can be imposed between interacting blocks. This simplifies the design and conveniently partitions it so that different engineering teams can work independently on different sections of the design. The Chip Assembler supports structured VLSI design by taking advantage of the fact that the physical and structural hierarchies are identical. It integrates the structural and physical data for a function block into one database. This solves the problems that arise when you have two databases that describe the same chip. One problem is verifying that the physical database represents the same chip that the structural database describes. Another problem is dealing with two discreet databases instead of one: create, update, delete operations become more complex.

In regular designs, the ratio of placed transistors to drawn transistors is very high. Thus a regular chip of 100,000 transistors may only have 5,000 drawn transistors: a ratio of 20:1. We are currently trying to achieve ratios between 50:1 and 75:1. By following some guidelines, we can achieve ratios in this range. We plan the wiring stategy first, then design the logic beneath the wires. We reserve low resist mask layers like metal for data, control, and power buses; and run these orthogonal to each other. We use structures like PLAs and finite state machines, which are highly parameterized and can be computer generated. A PLA is a programmable logic array consisting of an AND plane and an OR plane. Computer programs translate logic equations into PLA layout geometries. Repeated geometries like Register Files or wide datapaths also increase this ratio. Finally, we use microcode stored in ROMs to supply the necessary control functions.

Carver Mead pioneered regular structured design philosophy at the California Institute of Technology (Caltech). Researchers at Caltech developed tools to support this design philosophy. LAP is a program that allows users to specify mask geometries by placing SIMULA code into the LAP program. The syntax of the SIMULA code required to define the masks is simple enough that non-programmers can write it. Since SIMULA is a high level programming language, experienced users familiar with SIMULA were able to construct parameterized cells and layout. Simultaneously, Dave Johannsen developed a Silicon Compiler called Bristle Blocks. Bristle Blocks takes a high level description of a datapath and controlling PLA and automatically generates a correct mask layout for the chip. These tools and ideas paved the way for the Chip Assembler.

CURRENT STATUS

The current version of the Chip Assembler resembles a small disk based operating system. Operating systems perform operations on disk files like CREATE, APPEND, DELETE, COPY, RENAME, LIST a file on the terminal, HELP, and list a DIRECTORY of files. The Chip Assembler does the same for function blocks. It allows you to create, define, append, copy, list, and list directories of function blocks. It also has an extensive built-in HELP facility.

The function block is the basic building block of the Chip Assembler. A function block is a rectangular block that is either an assembly of other function blocks, or an atom. Atoms are blocks that cannot be broken down into smaller parts. These could either be computer generated, hand drawn, or compacted from a symbolic layout database. The Chip Assembler translates data formats from other tools like Applicon and Calma graphic stations into its internal geometric representation.

The Chip Assembler is written in SIMULA, a high level object oriented programming language based on ALGOL 60. It runs on a DECSYSTEM 20 computer under the TOPS-20 operating system. The Chip Assembler will be converted to run on VAX-11 computers under VMS.

Designers run the Chip Assembler at a low cost Designer Workstation consisting of a VT52 video terminal connected to a DECSYSTEM 20 timesharing computer. The terminal line has a PDP 11/04 computer and Hewlett-Packard 4 pen plotter connected in series between the VT52 and the DECSYSTEM 20 computer (Figure 7). The PDP 11/04 serves as the graphic processor to the color monitor. Both the HP plotter and the PDP 11/04 scan terminal communications for their special PLOTON character sequence. Until they receive their PLOTON instruction, they transmit characters through themselves behaving like a wire. Once they receive their PLOTON instruction, they interpret all ASCII characters as graphic plotting instructions until they receive their PLOTOFF command. This station provides the designer with immediate visual feedback which is essential for efficient layout designing.

The Chip Assembler produces a variety of output. Textually-annotated plots of the geometry are displayed on the color raster scan monitor or hardcopy pen plotter (Figure 8). These give the designers a medium through which to discuss the chip, as well as a means of seeing the chip's geometric characteristics early in the design.

In addition to geometric plots, the Chip Assembler also plots tree graphs showing the chip's functional and physical hierarchy (Figure 9). This aids the designer by showing which function blocks are subblocks of other function blocks.

Also produced are Chip Assembler database files recording the function block's data. When desired, a CIF 2.0 (Caltech Intermediate Form) file is produced and converted to CALMA database format. From the CALMA database, you can run Design Rules Checkers, Pattern Generation programs, and plotting programs that drive large bed plotters.

CHIP ASSEMBLER IN DETAIL

The Chip Assembler is an interactive, menu-driven data collection program that assists the designer in specifying the layout and electrical properties of a chip. The designer specifies all details of a chip in a simple syntax composed of common English words. The Chip Assembler collects and stores this data for the designer in its database.

To design a chip, the designer specifies rectangular blocks and then assembles these together to form larger blocks. The designer specifies the order of assembly and the orientation of placement. Blocks are placed in one of the following directions: North, East, South, or West. In any of the directions, there are two edges on which to line up the blocks. For North and South assemblies, you can line up the East or West edges of the blocks; for East and West assemblies, you can line up the North or South edges of the blocks.

After the designer has instructed the Chip Assembler to assemble the blocks, the Chip Assembler connects each block-pair together, creating the new layout and electrical data for the assembly. The Chip Assembler checks the bristles on each pair of edges that are to be connected. If the bristles abut, then the Chip Assembler physically abuts the blocks. Otherwise, the Chip Assembler river routes between the two interfaces (Figure 10).

When abutting blocks, the Chip Assembler performs a type check on each bristle pair joined. It verifies that the bristles are on the same mask layer and of compatible types. It reports errors such as connecting two inputs together, or routing POWER to GROUND.

When routing occurs during block assembly, the Chip Assembler insists that all bristles be of the same mask layer. This constraint arises from the fact that the river routing algorithm works on only one layer. If and when this changes, we will remove this restriction. However, in our designs, this is not a difficult constraint to satisfy. The Chip Assembler also checks that the mask layer it is routing on is a valid routing layer. This prohibits routing on a contact cut or implant layer.

After you have assembled a block, you can look at any or all of its data. You can also view the block on either the color monitor display or the HP color plotter. When viewing a block, you can specify precisely which branches of the subblock tree you wish to view. So if you have a block composed of a left side and a right side, you can have the right side only show you its exterior, while the left side plots itself in full detail. This can save much time when viewing arrays of blocks.

FUNCTION BLOCKS

The function block is the basic entity of the Chip Assembler. Each block supplies one chip function and contains the following data :

1) name

2) transistor count

3) list of subblocks

4) list of geometric shapes (CIF 2.0)

5) list of circuit simulator primitives

6) power consumption data

7) North, East, South, and West bristle lists

The four bristle lists contain the block's interfaces. Each bristle list contains any number of "bristles". Each bristle has the following data:

1) name

2) type and direction

3) color (mask layer)

4) width

5) capacitance

6) delays (estimated, minimum/maximum accepted, computed ...)

This data is analogous to the data a sophisticated compiler keeps about variables and procedures. The SIMULA compiler that we use does type checking on all variables and procedures, even across EXTERNAL files and procedures. This saves much debugging time. Usually, when the program compiles, it runs. The Chip Assembler does an analogous task when it assembles blocks together. It performs type checking for each bristle pair that it connects together. As we add more data to bristles, the Chip Assembler can perform more checks and thus verify that chip is specified correctly.

Designers input function blocks from the terminal, command file, or CALMA or APPLICON databases. You can also append data to an already existing function block. The Chip Assembler allows you to display any part of a function block's data, or it can store/retrieve function blocks from database files. Once you have a fully described function block, you can generate a list of the nodes with their connections, circuit simulator input files, and CIF 2.0 mask descriptions.

WORK TO BE DONE

Much work has yet to be completed in the Chip Assembler, but a sound foundation exists. Most of this work will be done during the summer months and completed for evaluation in September. The areas of investigation include:

1) design of pads and specialized/generalized routing required for them

2) design of a general purpose parameterized PLA and/or finite state machine in our current NMOS process

3) complex routing between distant blocks on the chip

4) another look at stretchable cells - we have temporarily put this function aside as our layout designers prefer to match all pitches by hand to achieve maximum density

5) include interconnect verification in the Chip Assembler and/or interfaces to existing interconnect verification programs

6) collect and produce logic simulator data files

7) define other VLSI building blocks

8) produce composite chip documentation from brief textual descriptions of each function block

DREAM

Although we are currently using a specific chip to drive the design of the Chip Assembler, I foresee many uses for it. The most exciting area is custom application engines. Custom application engines are not general purpose CPUs, but a special piece of silicon that implements a customized function. These engines perform their tasks orders of magnitude faster than general purpose computers.

As chips and chip sets get more expensive to design, can we really afford to continue building expensive general purpose machines to be expensively programmed at a later time? Perhaps a better question is "Do we really want to?". If our goal is to increase the price/performance of our machines, how can we best do this? Should we build an entire mainframe computer on a single chip? I don't think so. I think we can get an order of magnitude increase in performance without designing a faster model of an existing machine. By designing an application driven machine to do the task we want and nothing more, we can remove all the levels of interpretation that an application program goes through in its execution on a general purpose computer. And we can design special hardware to accomplish the task with more parallelism.

Today we are already seeing the first generation of these custom application chips. There are A/D converters, multipliers, FFT, a LISP machine, and graphic processors on a chip. We would like to see these chips perform the function of some of our longest running programs. For example, researchers are implementing PC board routing in VLSI. Maybe we could design a chip to do DRC checking, Differential Equation solving, or FORTRAN compilation.

By going this route, we are no longer limited to a machine that executes instructions sequentially, or even a tree/matrix machine that does so. With VLSI we can do exactly what we want, in the manner that is best suited to the data. This is much the same reason why we use an object oriented programming language like SIMULA.

One image processing task I ran across required the summation of two images. To perform this, you add the 4096 integers that represent one image's intensity (one integer per pixel) to the second image's 4096 integers. With VLSI we can afford to design a chip(s) with 4096 adders to do this in one machine cycle instead of 4096 or more cycles.

In applications like image summation, where we need to manipulate 4096 integers, we run into pin out problems if we try to put 4096 adders on a chip and feed them data from a main memory. A better approach is to take a bit slice approach and design a chip that handles 8 integers completely - it not only stores the data, but also performs operations on it. This means that we need 512 of these chips for the total engine. This chip would contain both the memory and processor combined, and thus eliminate the pin out problem.

VLSI application machines require "Smart Memories", which are memories combined with local processors. The data must be very close to the data manipulator. If this is not the case, communication severely limits the performance of the machine, both in speed and wiring complexity.

Application machines require custom chips that are currently very expensive to design. With the advent of tools like the Chip Assembler, the design of these chips can be done quickly and cheaply. Since the chips are designed rapidly, they are not as dense as if they were totally hand drawn. But they need not be maximum density. We can gladly sacrifice chip area for time and cost of design. With a dramatic reduction in cost of custom chips, a whole new market is created for VLSI.

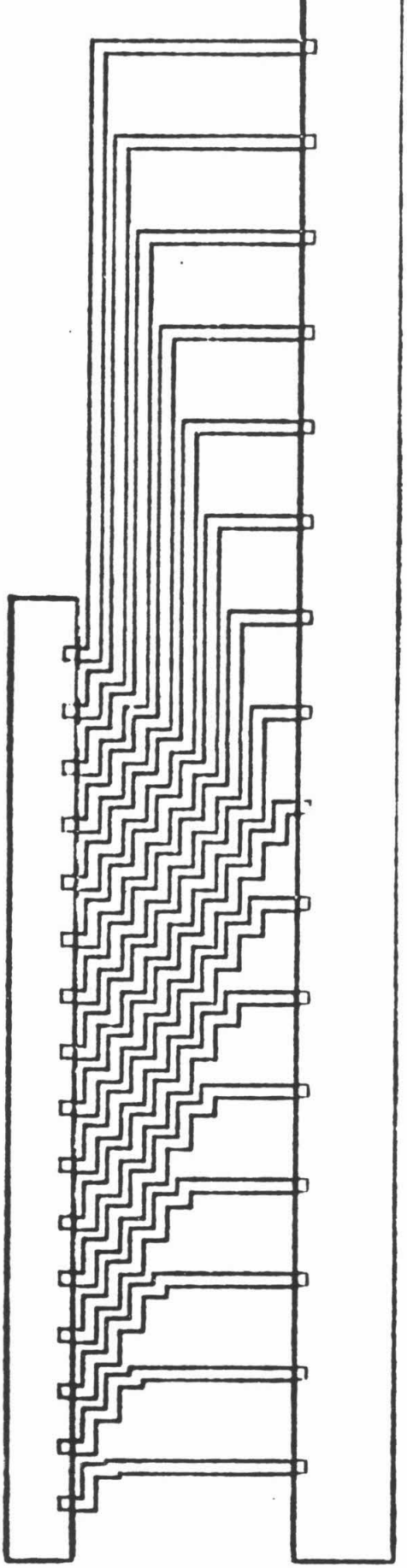


Figure 0.

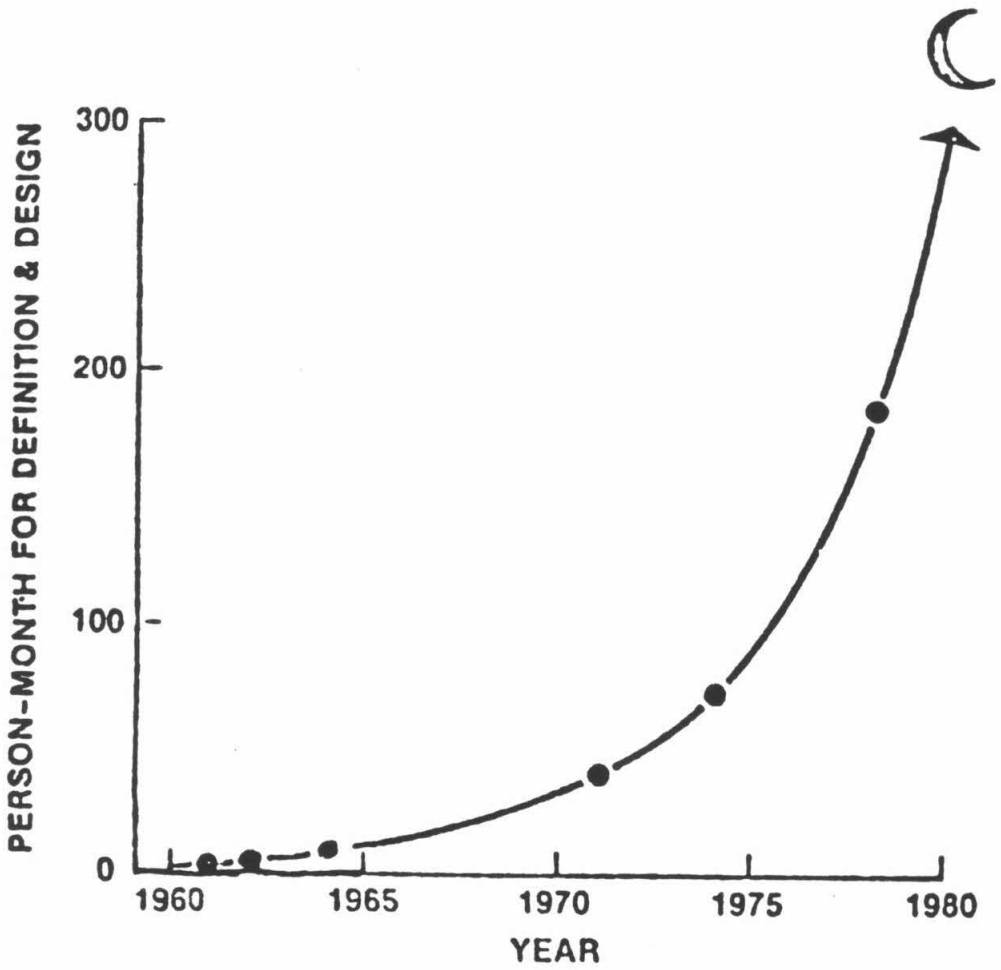


Figure 1.

(from Gordon Moore, CALTECH CONFERENCE ON VLSI, January 1979)

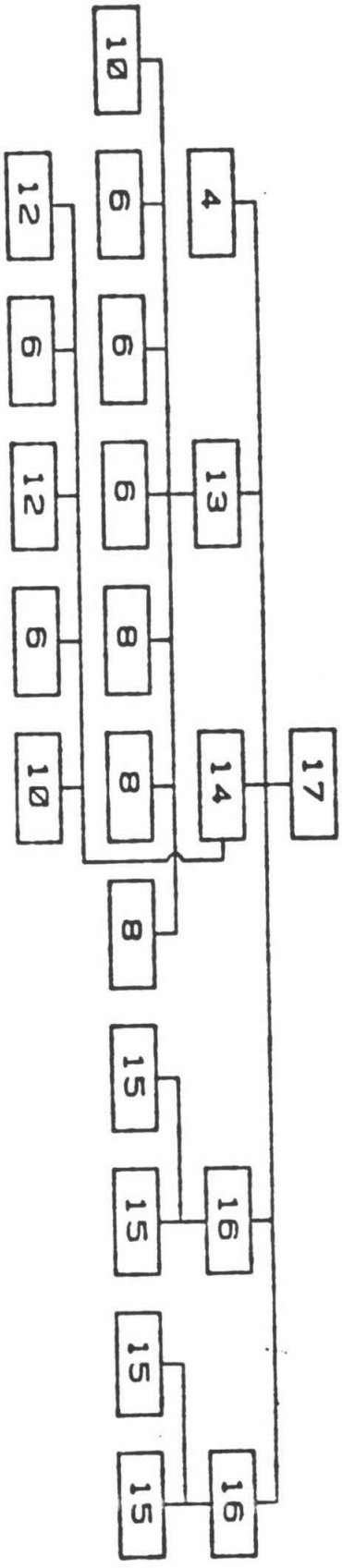
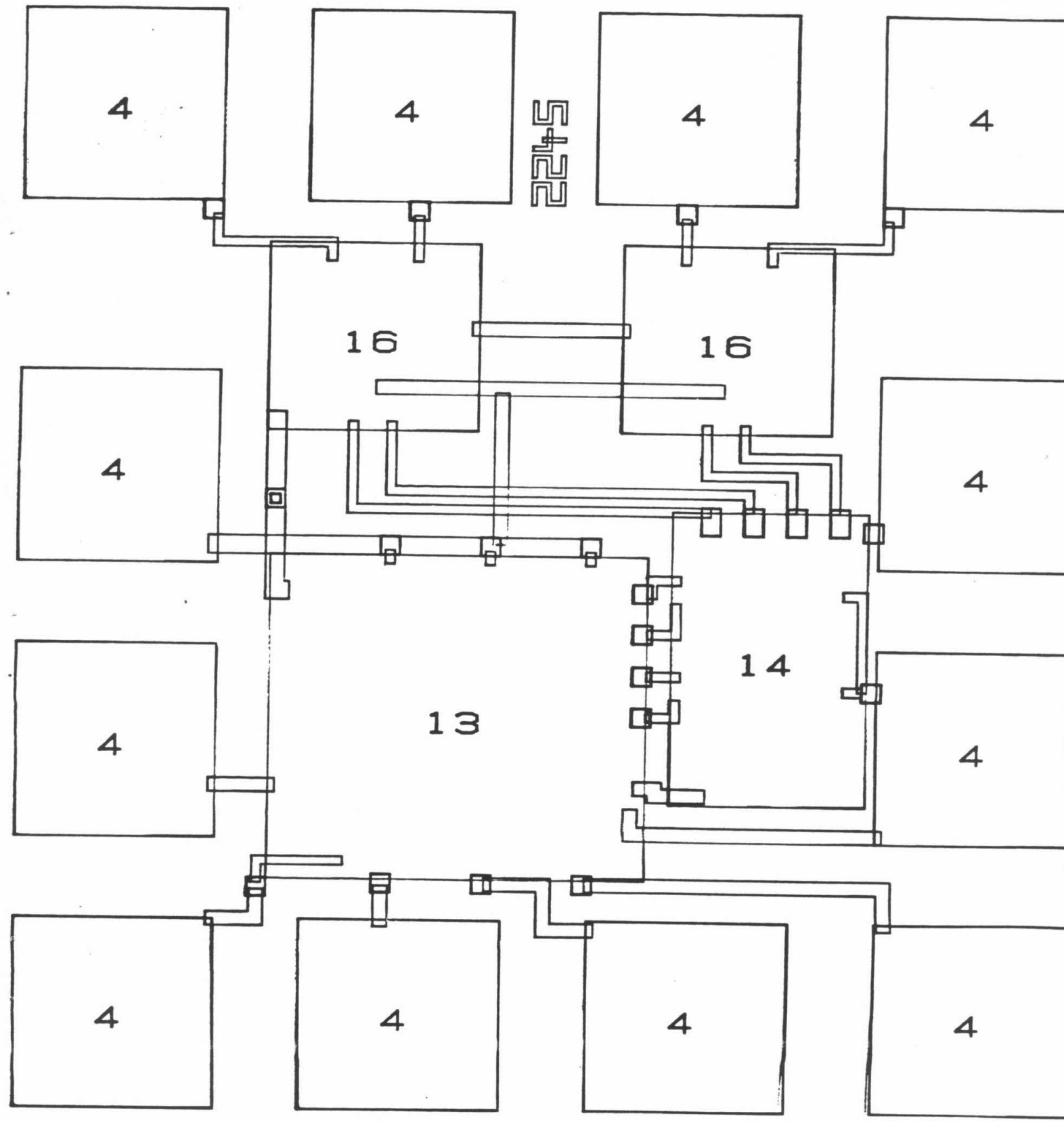
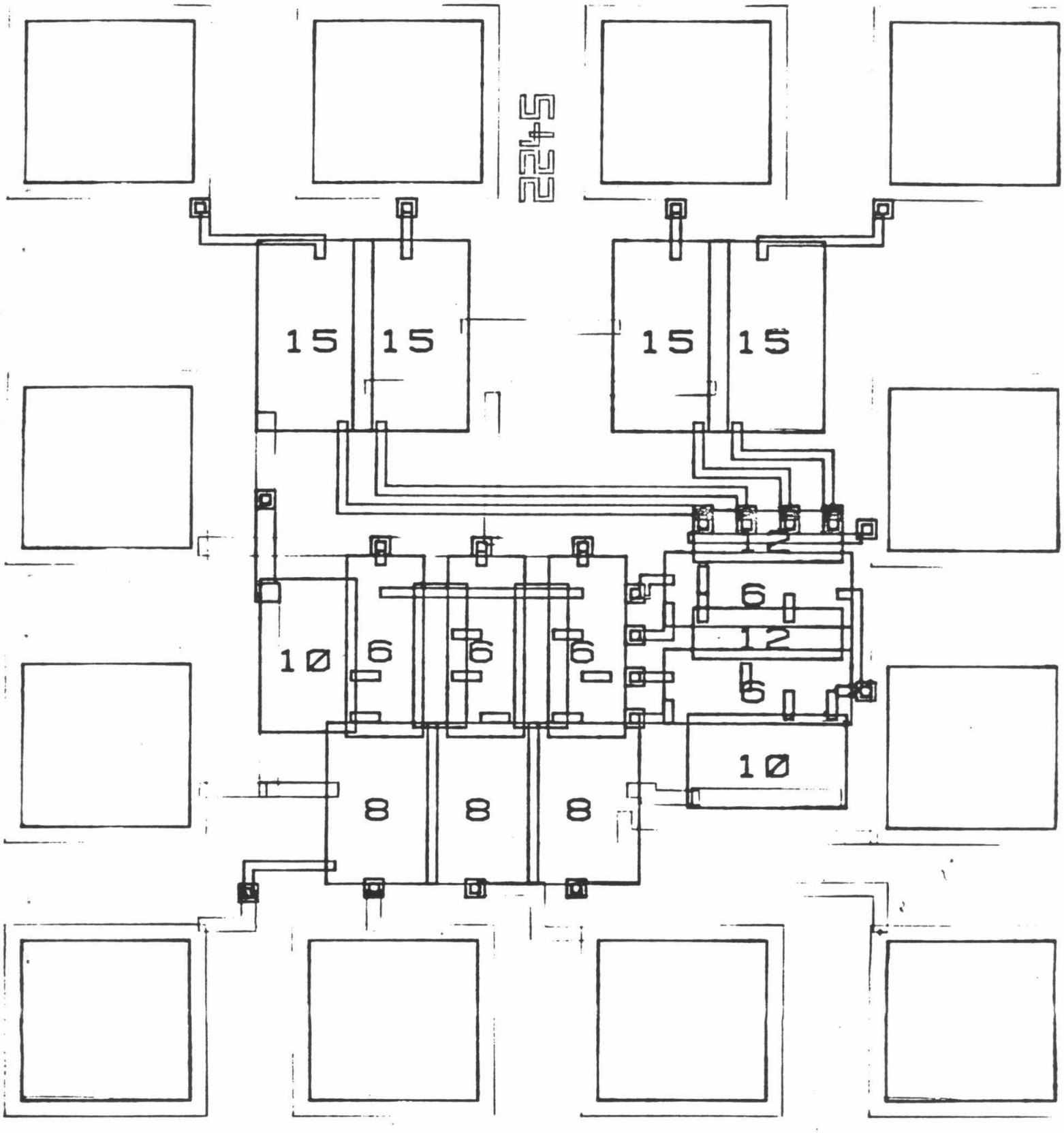
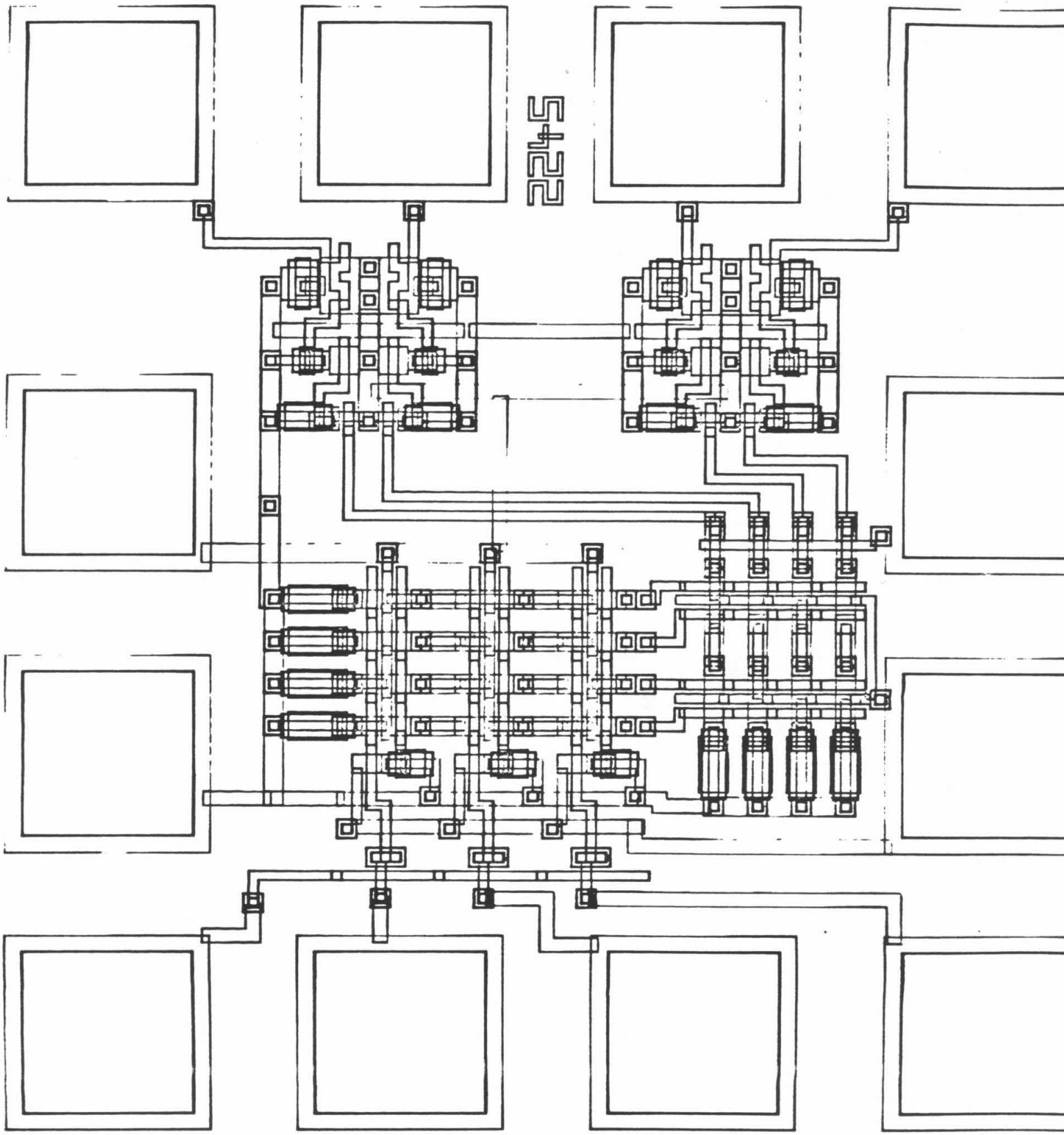


Figure 2.


4
4
225
4
4
16
16
4
4
14
13
4
4
4
4
4
4


22+5
15
15
15
15
12
6
12
6
10
6
6
6
6
10
8
8
8


224

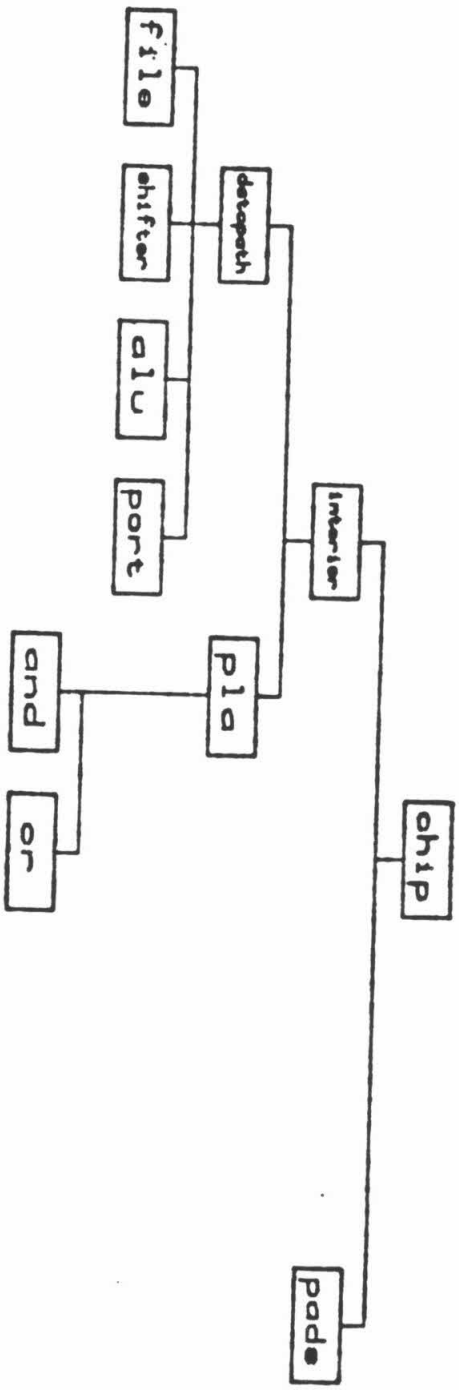

file
shifter
datapath
alu
port
interior
and
pla
or
chip
pads

Figure 6.

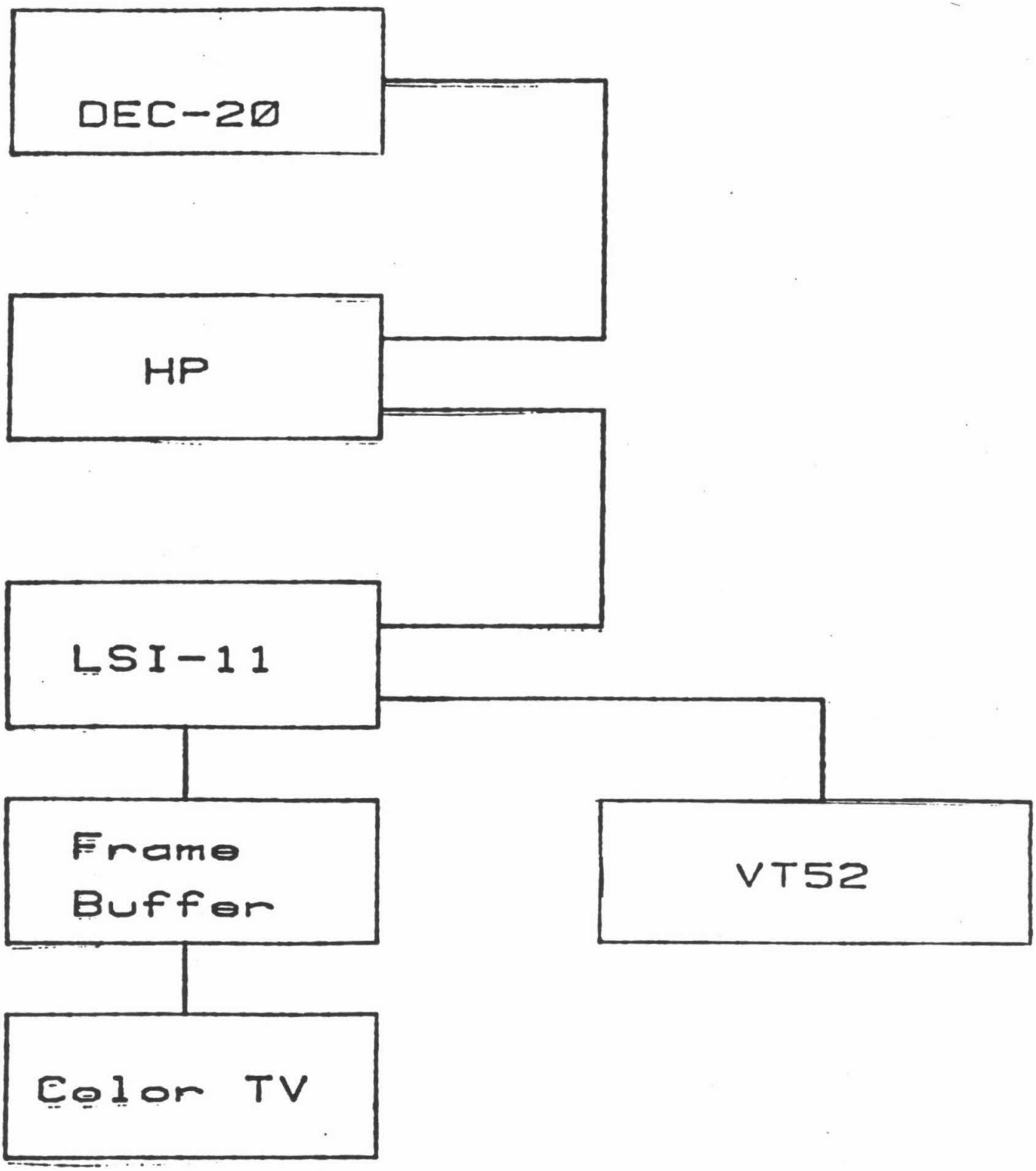


Figure 7.

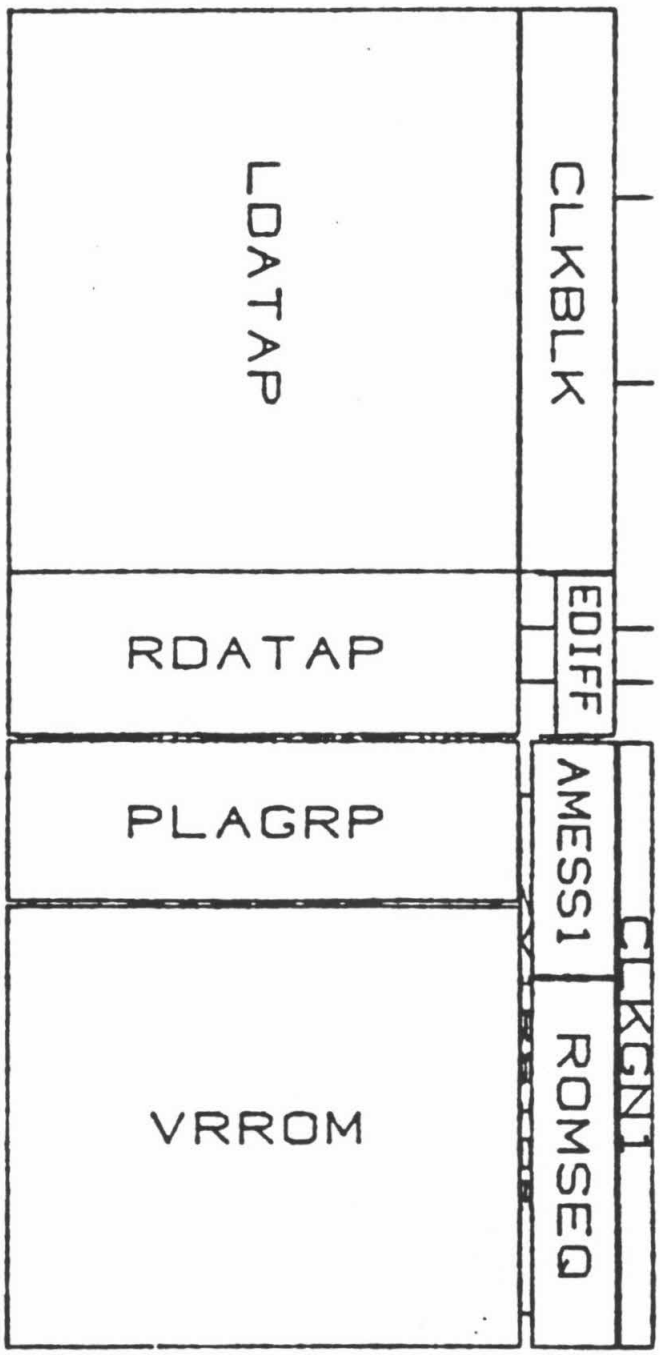

LDATAP
CLKBLK
RDATAP
EDIFF
PLAGRP
AMESS1
CLKGN1
VRROM
ROMSEQ

Figure 8,

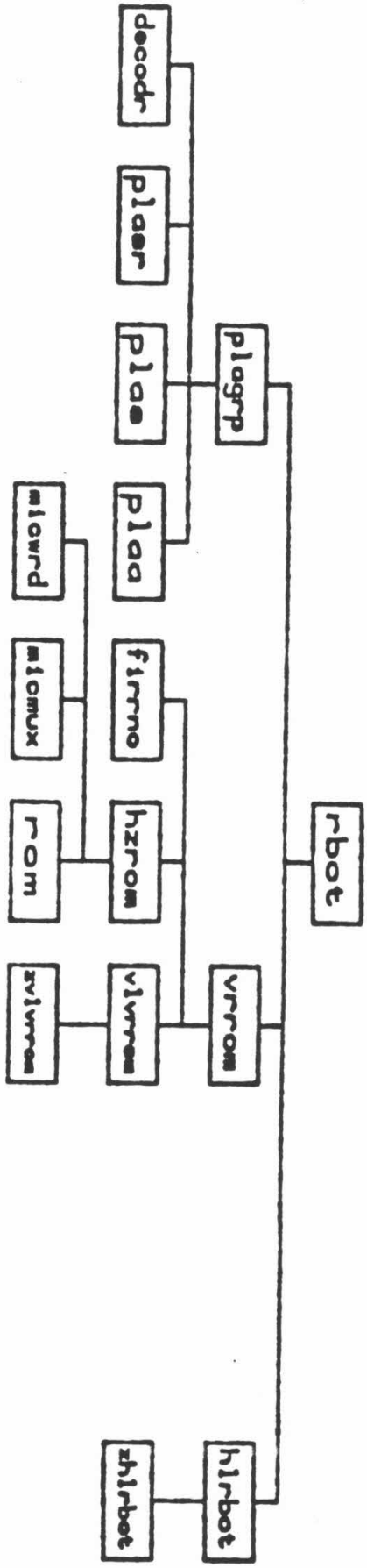

rboot
plogrp
decode
plaer
plae
plaa
mlomrd
mlomux
rom
xvimem
firmm
hzrom
vimem
vrrom
hlrboot
zhlrboot

Figure 9.

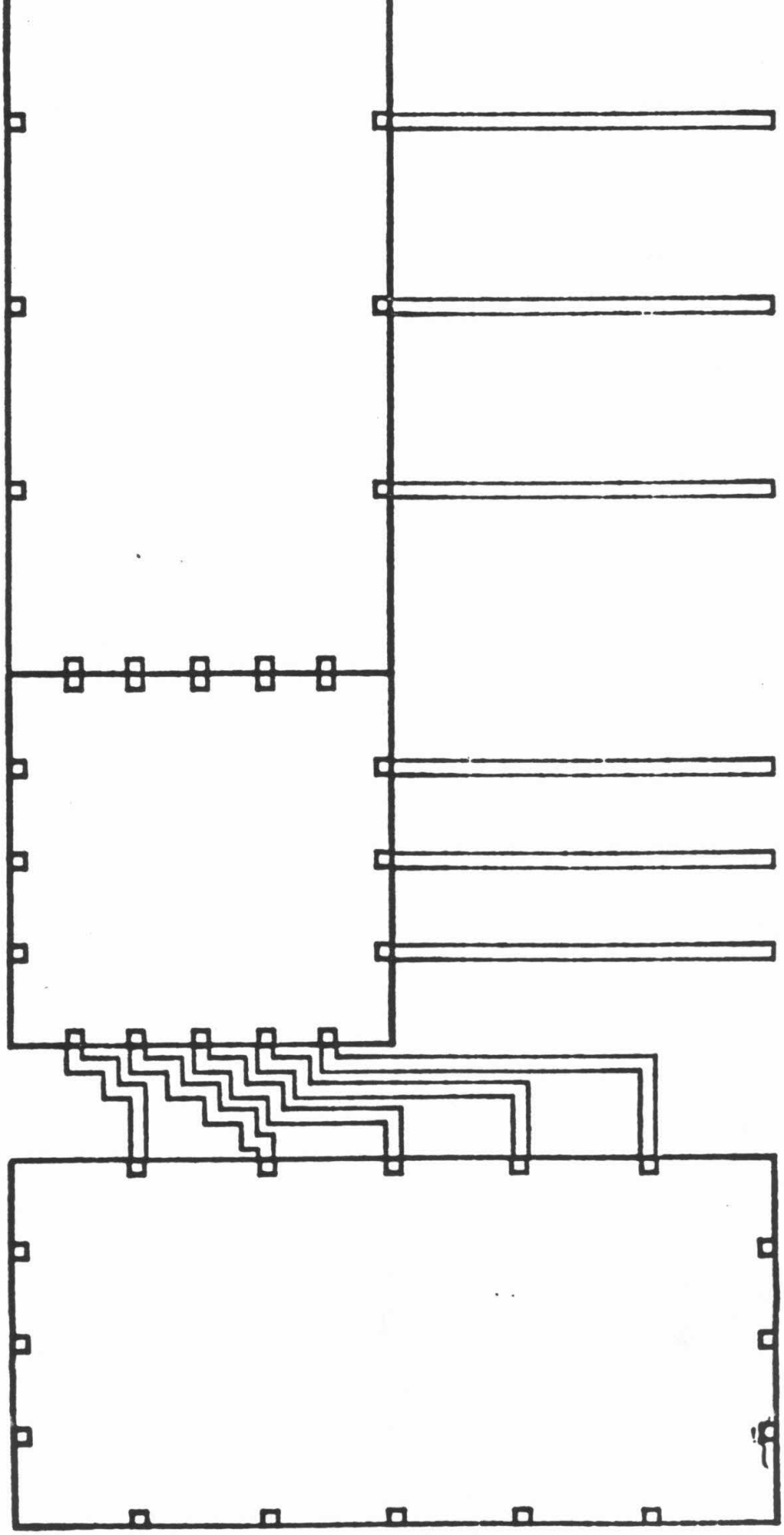


Figure 10.