1988

A Unix based VLSI design workstation.

Alger W. K. Yeung
University of Windsor

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

A Unix Based VLSI Design Workstation

by

Alger W.K. Yeung

A Thesis
Submitted to the Faculty of Graduate Studies through the Department of Electrical Engineering in Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science at the
University of Windsor

Windsor, Ontario
January, 1988

ISBN 0-315-43783-9

Abstract

As VLSI technology evolves the complexity of the design process is contantly increasing. The existence of a powerful, easy-to-use, and flexible VLSI design workstation is essential if one is to successfully design a complex VLSI circuit. This thesis deals with the design of a VLSI workstation based on the DEC VAXstation II/GPX minicomputer operating in an Ultrix (Unix) and X-Window environment.

A special silicon compiler called Memory Oriented Silicon Compiler (**MOSC**) has been developed by the author and is available on the workstation together with a number of utility programs and two public domain programs. The public domain programs, Electric and Relax2, have been modified and tailored to interact with MOSC in the Ultrix and X-Window environment. A full-custom hierarchical VLSI design methodology has also been identified in order to ensure that the VLSI design tools can be used effectively. A number of circuit designs have been carried out to illustrate the capabilities of the workstation. The MOSC silicon compiler is an application specific CAD tool. It was developed in order to afford the designer a simple manner for exploiting Residue Number System concepts and implementing pipelined memory oriented structures for digital signal processing applications. The MOSC program is capable of synthesizing 5-bit modular adders, subtractors, multipliers and constant operators in an integrated manner with the Electric design system. The utility programs developed include plotting routines, drivers for graphic

devices, and a file format translator that allows the exchange of files with other VLSI design software available in the VLSI Research Laboratory.

The software developed and subsequently implemented on the VLSI design workstation has proven to be powerful and easy-to-use tool for designing VLSI circuits.

Acknowledgments

The author would like to acknowledge the guidance and suppo.. provided by Dr. W.C. Miller and Dr. G.A. Jullien. The ideas and suggestions by other member of the VLSI Research Group were also greatly appreciated.

Table of Contents

List of Figures

List of Tables

CHAPTER ONE

Introduction

As the design and fabrication technology associated with Very Large Scale Integration (VLSI) circuits advances, the complexity of VLSI circuit design is constantly increasing. Each year more complex architectures involving a larger number of transistors can be implemented in a smaller silicon area. Accordingly circuit designers must then rely more and more on computer-aided design (CAD) tools. Many CAD tools have been developed to simplify the design tasks during the past decade and consequently the design time required for a successful design has been reduced from years to months or weeks. However, as the complexity of VLSI circuits increases viable CAD tools must be much more sophisticated. In some cases, the tools are difficult and inflexible to use.

For successful VLSI circuit design, the actual design morphology is only half the battle, and the other half is learning how to use the CAD tools effectively and in a straight forward manner. Therefore, a powerful, easy-to-use and flexible CAD tool for VLSI design is needed to aid in the solution of complex design tasks.

The objectives of this research are to create a fully functional Unix based VLSI design workstation, to identify a full-custom hierarchical VLSI design methodology, and to develop a special class of silicon compiler for high speed digital signal processing (DSP) applications. The workstation is based on the DEC VAXstation II/GPX

minicomputer located in the VLSI Research Laboratory at the University of Windsor.

The development of the fully functional VLSI design workstation has been influenced by a number of reasons. First of all, Ultrix, the Digital version of the Unix operating system, is selected to run on the workstation because the VLSI research community in most other universities uses Unix as the most popular operating system. In order to communicate with and be compatible with other universities a Unix based facility for VLSI design must be installed in our research laboratory. In addition, we wish to be able to use the large number of public domain VLSI design software programs developed in a Unix environment that are available.

As shown in Figure 1.1 the present VLSI design facilities at the University of Windsor are centered around Phoenix Data System software running on a VAX-11/750, a standalone Daisy ChipMaster workstation and the Applicon VLSI design system operating on a VAX-11/785. Presently most of circuit mask layouts are done using the Daisy's MAX layout editor, and then the design verification is performed using the Phoenix Data System software. Hence, a lot of time is typically spent on converting data formats and transferring files between the two machines. A completely integrated workstation with a layout system and a circuit verification capability is needed to improve the VLSI designer's productivity and use the computing resources more effectively.

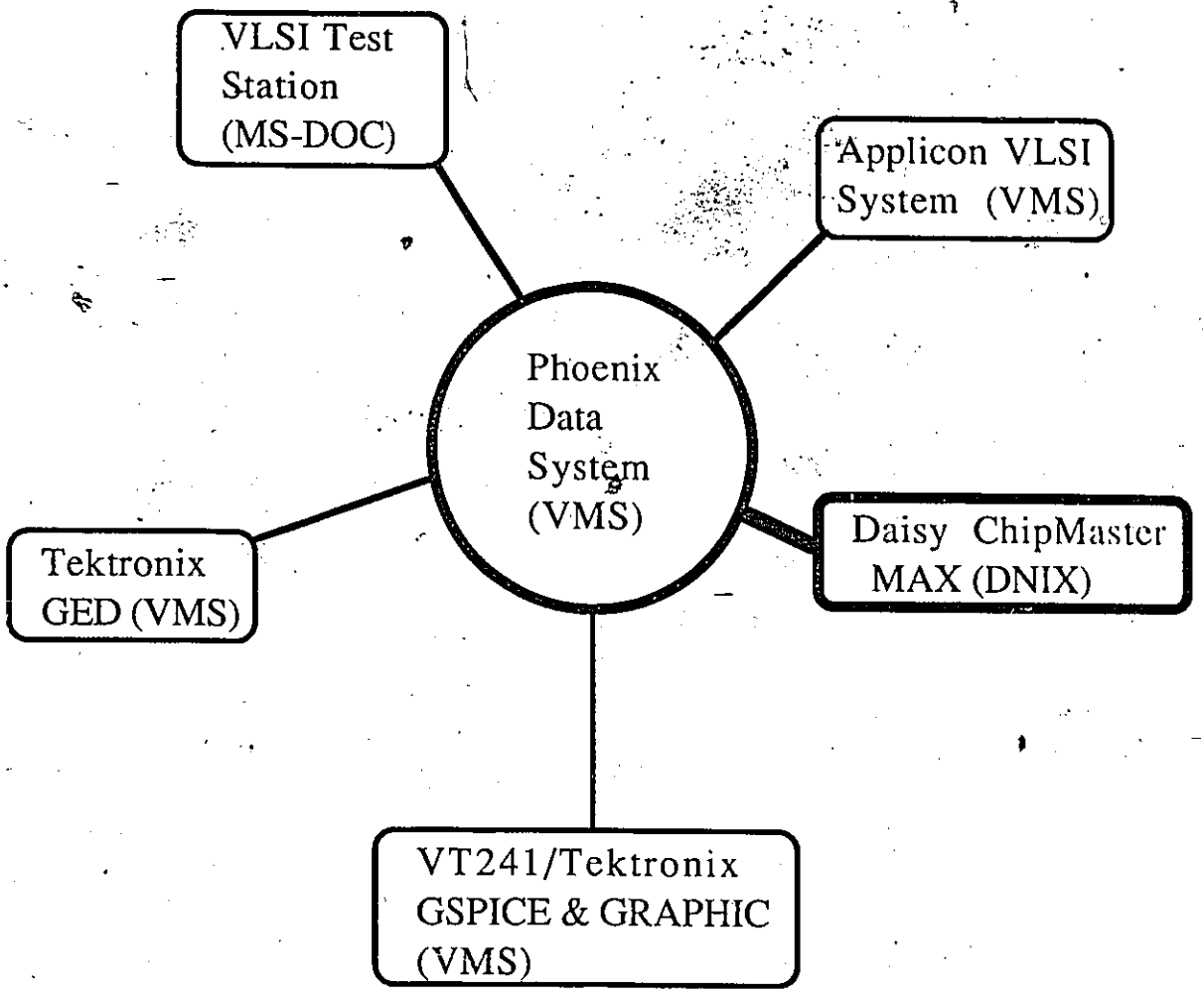


Figure 1.1: VLSI design facilities in the VLSI Research Laboratory

It was also desired that the VLSI workstation have a special silicon compiler that allows one to synthesis Residue Number System (RNS) arithmetic operators directly from design specification. A large portion of the research efforts carried out by members in the Signal and System Group has been concentrated on high-speed DSP applications based on RNS structures. As part of this research a number of basic memory cells based on the RNS concepts have been designed and fabricated successfully. In order to expand this

research orientation it is very useful to carry out the design of RNS arithmetic operators with the aid of a silicon compiler and to integrate the silicon compiler with the VLSI design workstation software.

The research plan was divided into three paths, as shown in Figure 1.2. The percentage shown beside each box represents the relative amount of effort expended in that area. The first path is mainly concerned with the Ultrix operating system, the windowing facility, and public domain software programs. The second path concentrates on the creation of a silicon compiler for the RNS operators. The third path leads to the identification of a VLSI design methodology and to determination of which CAD tools are required at each design stage. We must know the problems or difficulties involving the interrelations between design procedures, and the availability of and need for CAD tools must be determined before we can provide a viable VLSI design workstation. In addition to the methodology and tools problem, a data format translator must be developed in order to allow the workstation to communicate with other VLSI design tools.

Now let us look at the hardware elements of VLSI design workstation first followed by software details. The design workstation is physically based on the DEC VAXstation II/GPX workstation which supports a wide range of applications. The GPX workstation gains its performance advantage from a powerful graphics coprocessor that offloads text and graphics computation from the CPU. It also offers a double-buffered video memory closely

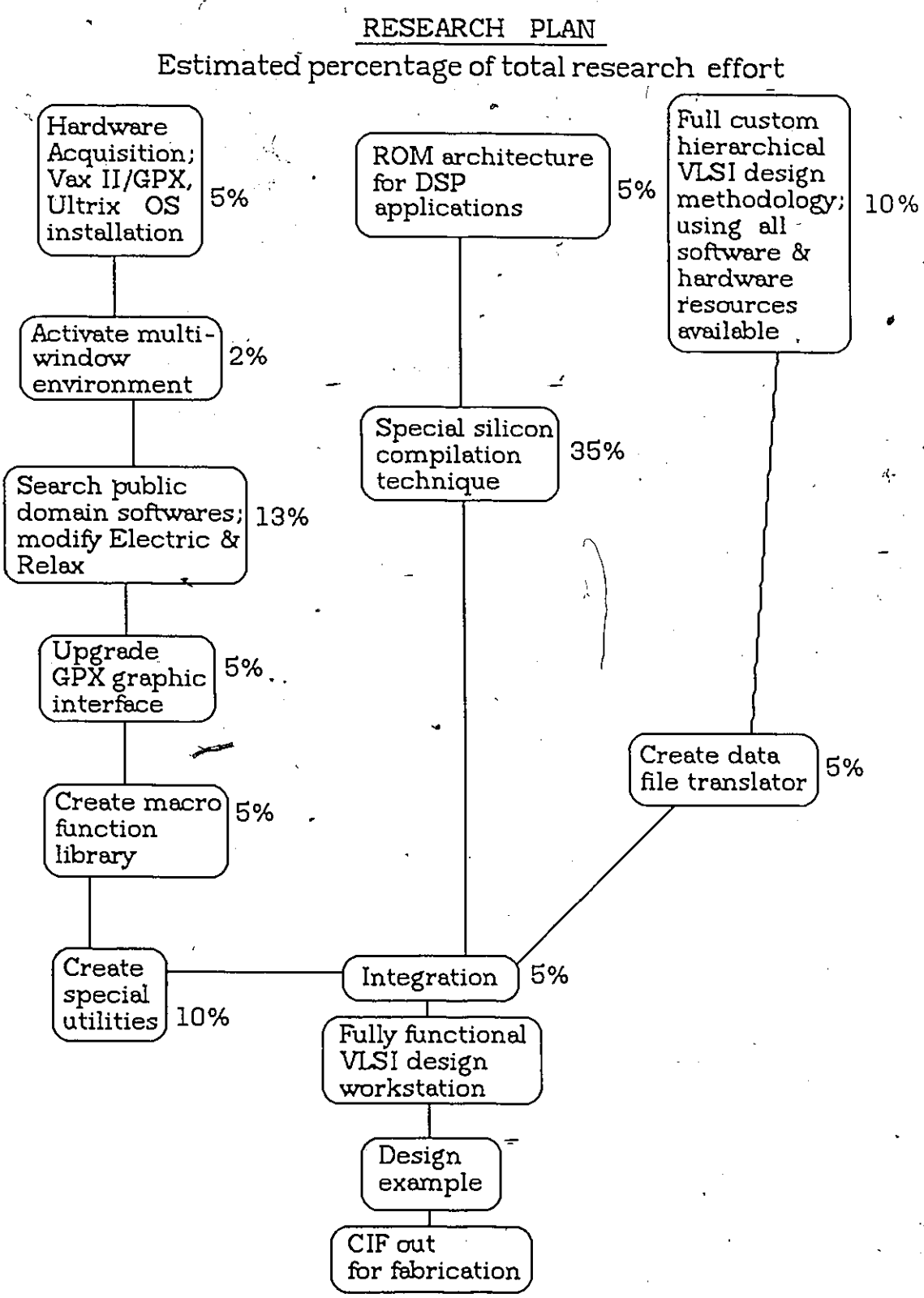


Figure 1.2: Research activities involved in the project.

coupled to the coprocessor, which translates into faster text and graphics drawing speeds. The coprocessor can independently access display list instructions to manipulate graphical objects, further freeing the CPU to run the application software. The resolution (or pixels) of the 19-inch colour monitor is 1024 * 864. 256 colours selected out of a possible 16 million colours can be displayed simultaneously. The GPX workstation is currently equipped with 9 Mbytes of main memory, 216 Mbytes of disk storage and a 95-Mbyte tape drive.

A Unix based computing environment has been implemented on the GPX workstation. Ultrix, Digital's Unix operating system, is compatible with AT&T's system V while maintaining all Berkeley 4.2 BSD commands, system calls and library functions. Ultrix version2.0 has been successfully installed in the GPX workstation. Ultrix also features the X-Window system which is emerging as a windowing standard for the Unix workstation community. The X-Window system has also been loaded in the workstation. A startup command file has been written to initialize the windowing environment, and to define various combinations of keys on the keyboard and mouse buttons to invoke a number of window commands. The X-Window system allows one to resize, restack, move, and iconify windows. It also provides pop-up menus for creating new windows, changing window colours, or starting applications. It can be easily customized, giving users complete control over their own window environment. A more detail description on how to use the operating system and the X-Window facility is included in Appendix I. Besides the windowing environment, a communication system is also provided. The

workstation is linked to other main frame computers by an Ethernet realization of a local area networks. A mail facility and remote file manipulation across heterogeneous operating systems are setup properly for mail and file transfer. A manual has been written for users to use the facility effectively and is also included in Appendix I.

One of the research goals was to investigate the performance of public domain software and determine which software could be useful to the project. After a number of available software packages were evaluated and **Electric**, a complete electrical design system developed at the Fairchild Laboratory was chosen as having the potential to run on the GPX workstation effectively. Since Electric was primarily developedto run on a Sun workstation and an AED frame buffer terminal operating under the Berkeley BSD 4.2 system, a lot of modifications had to be made. A discussion on how to modify and compile programs is presented in Appendix II. After a long process of development Electric design system has been successfully integrated as part of the application software running on the GPX workstation.

In summary, the software developed for the GPX workstation falls into three areas. The first and major software project is the realization of a silicon compiler for RNS oriented applications. The second major project is the modification of the public domain Electric package and its integration into the workstation environment. The third major project area was concerned with the development of a number of special utility programs, such as file conversion programs, plot routines, and graphic drivers for a number of output devices. All

these programs were implemented on the GPX workstation and then fine-tuned to provide a powerful productivity tool for VLSI design.

CHAPTER TWO

Full-Custom Hierarchical VLSI Design Methodology

A typical VLSI design methodology described in the literature [WeEs85] consists of a behavioral description, a structural description and a physical description. Each of these descriptions is further subdivided into a number of design options that may be selected for a particular design. These three levels of description are adequate for a general integrated circuit design environment. However, in order to take full advantages of a given process technology and to simplify the steps associated with complex circuit design, a full-custom hierarchical VLSI design methodology has to be adopted.

An effective design methodology alone is not enough for designing complex VLSI circuitry. A number of VLSI computer-aided design (CAD) tools is also required to reduce the complexity of the design process. A VLSI circuit is a complex maze of polygons and lines that form paths that in turn combine to effect an overall function. For successful circuit design CAD tools are needed to analyze these paths and to ensure that they are correct. In reality, a circuit design method is influenced by the CAD tools available to an IC designer. A full-custom VLSI design methodology will be presented in detail in this chapter. The functions of various CAD tools in different design phases are also explained. In addition, a silicon compiler, which ultimately generates the physical mask layout

description directly from behavioral description of a design, will also be described to show where it fits into the design methodology.

A. Morphology of VLSI Design

A VLSI design process can be largely simplified by employing the use of a hierarchical structure. The use of hierarchy involves dividing a circuit (module) into simpler subcircuits (submodules) and then repeating this operation on the submodules until the complexity of the submodules is at a comprehensible level of detail. This concept is similar to the one used in developing a complex software program where large programs are split into smaller and smaller sections until simple subroutine, with well-defined functions and interfaces, such as passing parameters, can be written. In other words, the major reason of using hierarchical description is to keep the amount of detailed information the designer must work at each stage in the VLSI design process at a minimum.

Once the submodules have been identified, modularity for the submodules can then be considered. If the modularity of submodules is well formed, the interaction with other submodules can be easily achieved. For instance, the physical interface of each submodule that indicates name, position, layer type, size and signal type of external interconnections must be well defined so that no extra connections are needed to connect the submodules when the cells are placed together.

The use of hierarchy and modularity simplifies the problem complexity and enhances the design strategy. The hierarchy and modularity are parts of the hierarchical VLSI design methodology which is identified in Figure 2.1.

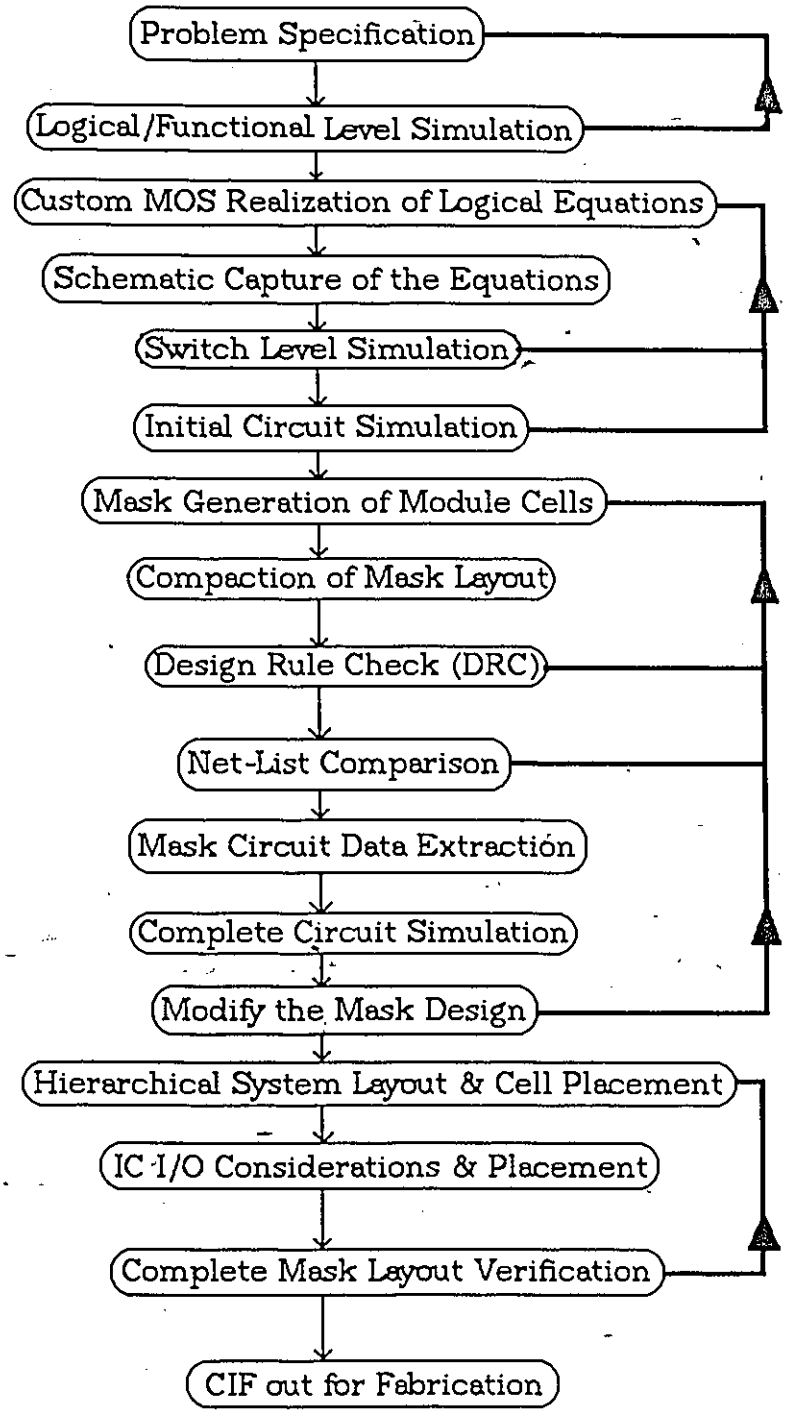


Figure 2.1: Full-custom hierarchical VLSI design methodology

The VLSI design is a continuous trade-off process to achieve

proper results through a number of iteration processes as shown in the Figure 2.1. Different VLSI design tools are used in each design phase to reduce the complexity of the design and assure the designer of a working product so that a diversity of VLSI design tools can be vital to the success of a VLSI circuit design project.

In order to explain the design methodology effectively, a simple 4-bit ripple carry adder circuit is used to illustrate the design procedures and the function of the various available on the VLSI design workstation.

In the first step of the design process, the problem specifications must be clearly stated and divided into smaller problems. This step is usually done by a designer. The hierarchy of VLSI design is employed in this step. For example, when designing a 4-bit combinational adder we have two 4-bit inputs, labeled A and B respectively, one 4-bit output labeled SUM, and 1-bit output labeled CARRY. The adder can represented as a black box with two inputs and two outputs, as shown in Figure 2.2. Now we have to simplify the black box so that the problem specifications can resolved into smaller problems. Since we are designing a 4-bit adder, we can divide the black box into four smaller black boxes as shown in Figure 2.3. Each of these subdivisions has three 1-bit inputs and two 1-bit outputs. The simplified design problem is now how to design a 1-bit adder.

Once the problem has been clearly identified, an initial logical or functional level simulator is built to simulate whether the initial design concept will meet design specifications. In logical level simulation, circuit elements are modeled as simple switches or gates connected by wires. The logical level simulator usually simulates the

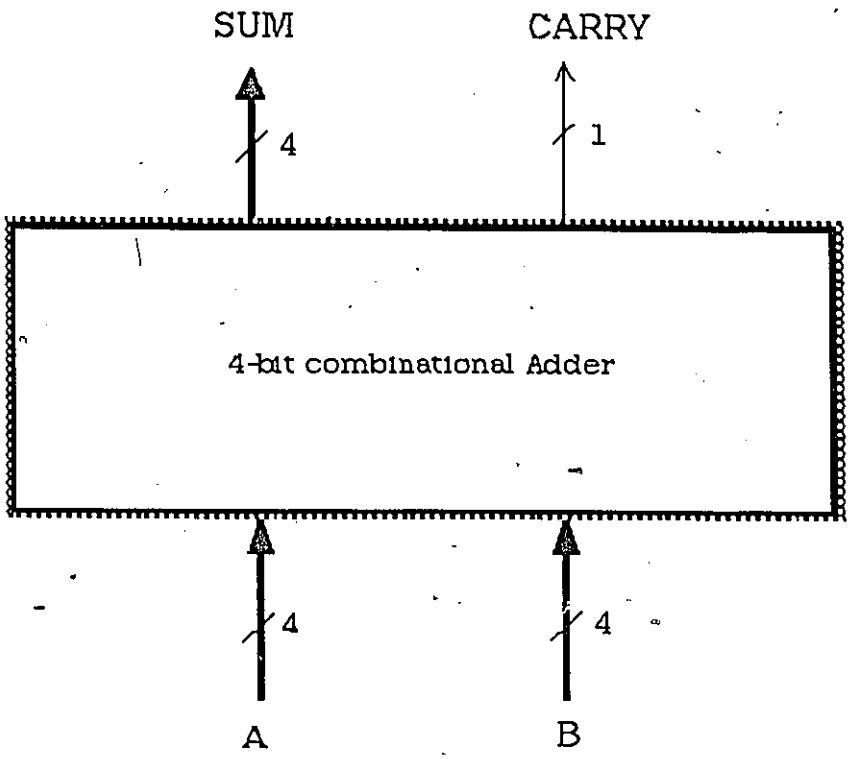


FIGURE 2.2: The 4-bit Adder at block level

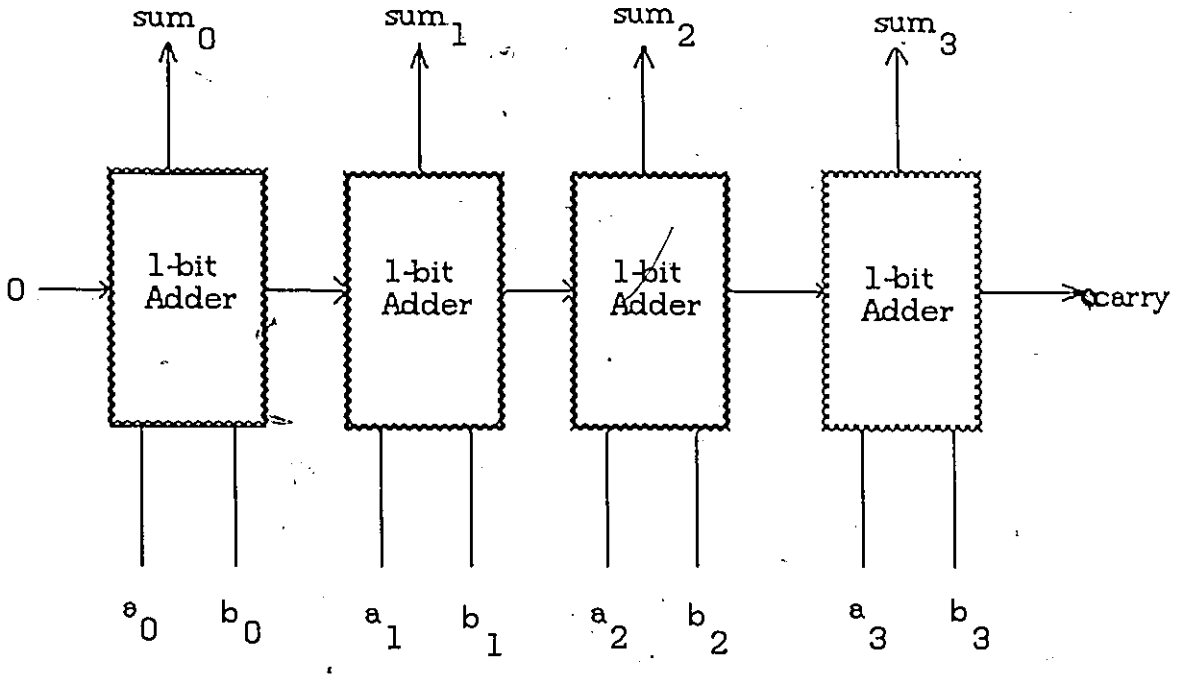


Figure 2.3: The 4-bit adder at sub-block level

circuit at the device level or at the gate level. In most complex cases, the logical simulator is dependent upon the problem so that it is usually created by the designer. If the results of logical simulation are not satisfactory the designer may have to make some trade-off decisions with the problem specifications in order to achieve adequate results. For the simple example of the 4-bit adder, a logical level simulation is not necessary. A truth table or functional description of the adder can be easily obtained and is shown in Table 2.1.

C	A	B	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.1: Truth Table of a 1-bit Adder

In most cases, Boolean equations can be derived from the results of the logical level simulation. In the case of the adder example, the Boolean equations derived from the Table 2.1 are shown as follows:

$$SUM = ABC + A(BC)' + C(AB)' + B(AC)'$$

$$CARRY = AB + C(A+B)$$

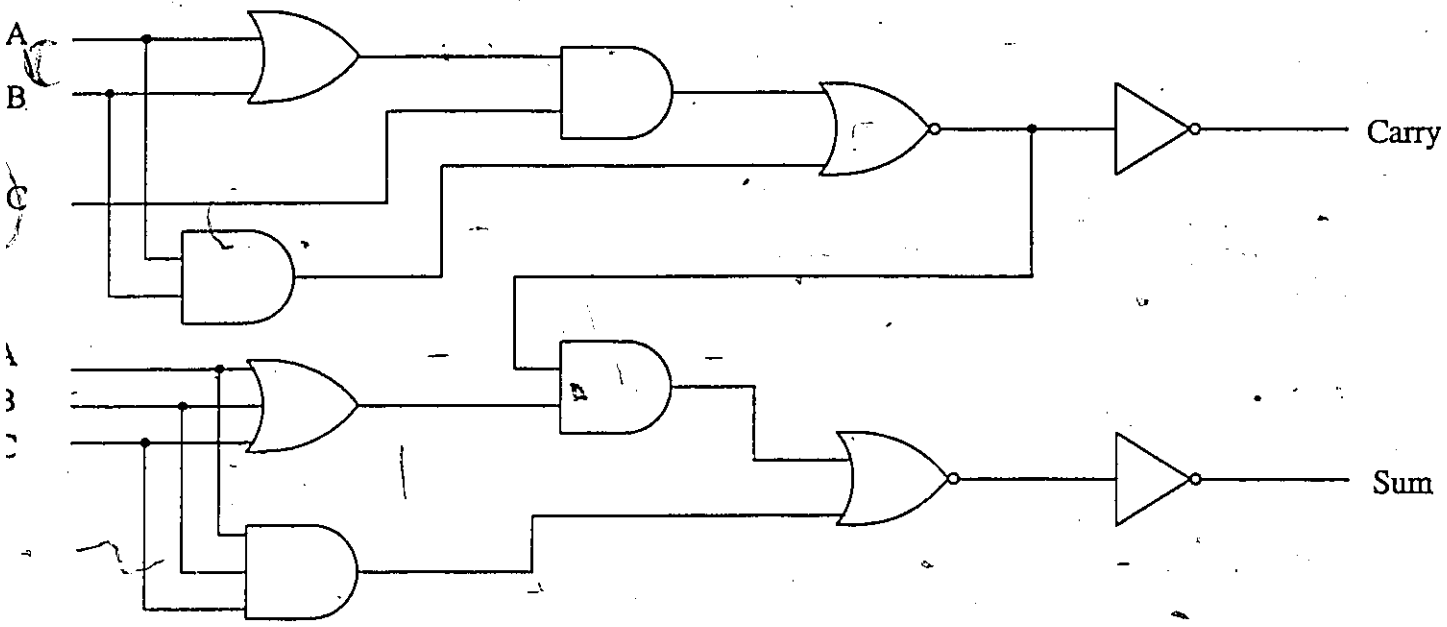
The next task to be performed by the designer is a custom CMOS realization of the logical equations. The CMOS realization is initially captured in a schematic form which is usually created using a schematic editor. The schematic form is used to store the major logical components or MOS transistors and their connectivity. The

process of creating a schematic is sometimes called schematic capture.

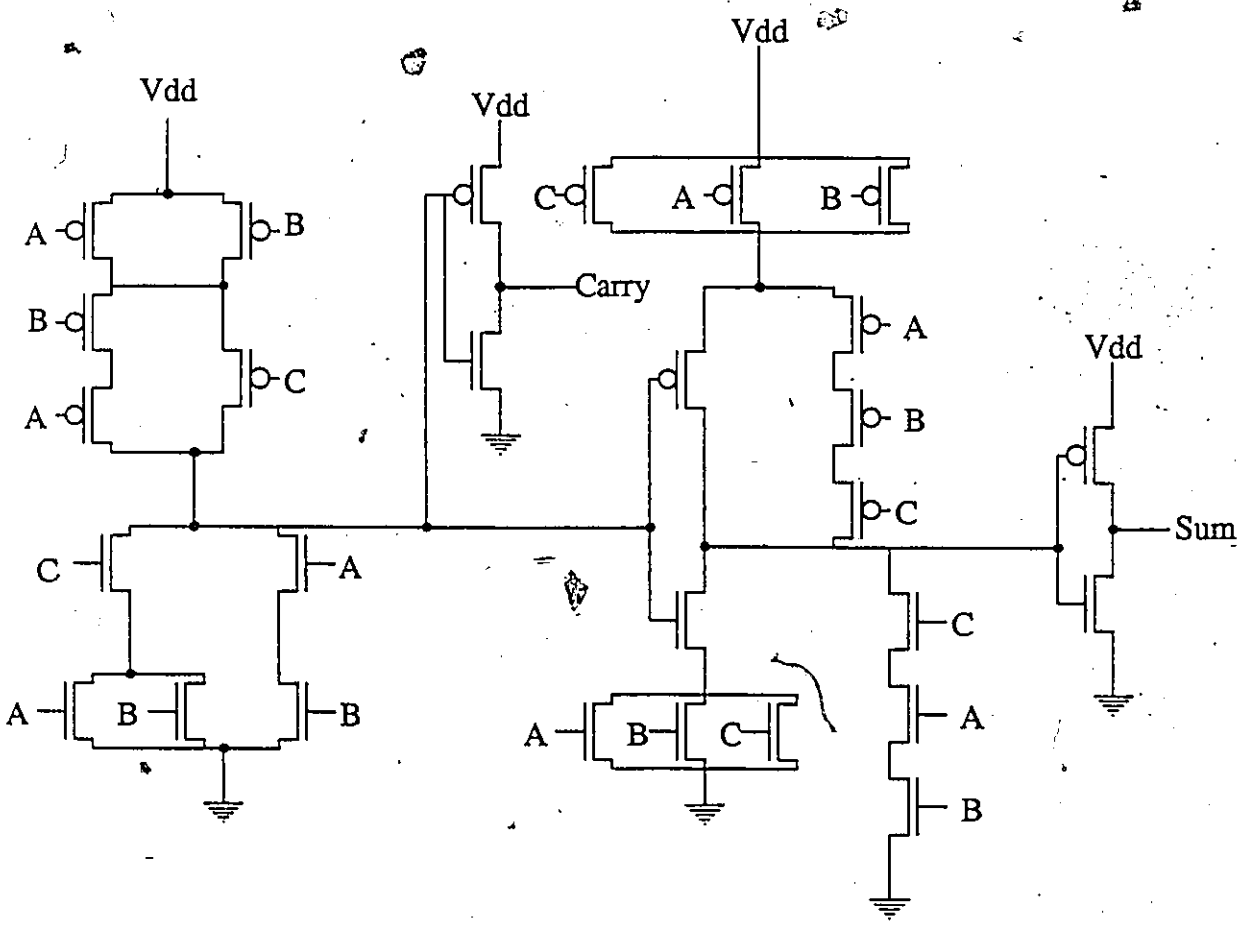
The designer may use the schematic information to ensure that the mask layout to be created correctly corresponds to the desired circuit in the later design phases. The designer may schematically capture the logical equations at the device or gate level and the MOS realization at the transistor level. In the case of the 4-bit adder, the gate level and transistor level schematic captures shown in Figure 2.4 are created by the schematic editor available on the VLSI design workstation. The schematic editor in the workstation is invoked by using the **technology use logic** command. This editor can draw all major logical devices, including gates, flip-flops, user-defined black box, all types of MOS transistor and various meters. The connectivity of the logical components is stored in a circuit netlist.

Once the netlist has been captured, switch level [Brya81] and initial circuit simulations can be performed to ensure the correct correspondence between the custom MOS realization and simulation results. These simulators mainly use a transistor as the primitive or switch element. In the VLSI design workstation, five switch level simulation interfaces for ESIM, RSIM, RNL, CADAT and MOSSIM simulators are supported. These simulators use only the (0, 1, X) states. For each set of input vectors, the circuit is simulated to determine a steady-state level at the output nodes.

On the other hand, the initial circuit level simulation [Nage75] determines the analog waveforms at particular nodes without considering any capacitance or resistance. The circuit elements are modelled as transistors with a number of parameters. The values of



(a)



(b)

Figure 2.4: (a)Gate-level Schematic Capture of 1-bit Adder
(b)Transistor-level Schematic Capture of 1-bit Adder

the parameters are determined by the technology process and the geometrical properties of the transistors. The initial circuit level simulation combines the information from the netlist and the user-defined device model to provide more detail about the waveforms at the nodes. One of the most widely used circuit level simulators in VLSI design is SPICE and this package is also supported on the VLSI design workstation. The SPICE deck information for the circuit can be extracted from the circuit schematic which was created in the previous design phase. If the simulation results turn out to be unsatisfactory, the designer may have to go back and modify the MOS realization. This process is repeated until satisfied results are obtained.

For the example of the adder, a SPICE input file is created by the SPICE Deck Extractor, and a circuit simulation has been performed and the simulation results are shown in Appendix III. The updated CMOS transistor model information is also automatically included in the input file. If a power source, ground node, input signals and output signals are completely specified in the schematic capture, a complete SPICE input file can be created directly without any human interaction. Two programs, GSPICE and GRAPHIC, have been written to enhance the SPICE graphic capabilities. These two programs can significantly speed up the whole simulation process. A detailed description on how to use the programs together with a program listing of GRAPHIC are included in Appendix IV.

Once the initial simulation results are within a satisfactory range the mask layout generation of each module cell can be undertaken. The modularity of the modules shall be considered here

in order to simplify the hierarchical system layout in a later design phase. For example, the power (VDD) line may be located at the top of the module cell and the ground (VSS) line at the bottom of the module. The inputs may come from the left side of the module and the outputs from the right side of the module. If the layout of the circuit mask is done in this fashion there is no need to create extra connection wires between modules when the module is replicated to form a chain.

There are two types of layout approaches in manually creating a mask layout, namely icon type and polygon type. For the icon type layout, the designer is concerned with the mask layout mostly at the transistor and transistor interconnection level since a MOS transistor is a primitive element and has its own symbolic representation or icon. In contrast, for a polygon type layout system the designer has to create every single polygon. Hence, the use of icon oriented layout is a much faster approach in creating a mask layout. The connectivity of a mask layout created by an icon layout method is also easier to be established and maintained. However, one of the major advantages of using a polygon based layout method is that a VLSI circuit can be realized in a minimum area of silicon.

Both mask layout types are supported on the workstation. The native mask layout method on the workstation is the icon oriented one because connectivity information can be extracted quickly and this is important to the other CAD tools of the workstation. The polygon type layout capability is also enhanced with the aid of powerful macro functions that are described in Appendix V. For the

adder example a mask layout for a 1-bit adder is shown in Figure 2.5. The mask represents 28 transistors and their interconnections.

Once the initial mask layout has been created, the designer may optimize the size of the mask by using a compactor. The function of a compactor is to remove unnecessary space from a design by moving every layer of the mask to its closest permissible distance. A two-dimensional compactor is available on the workstation used to compact a mask layout in both the horizontal and vertical directions.

The compacted mask layout of a design must then be checked thoroughly to guarantee that there are no design-rule violations. This process is usually called the Design Rule Check (DRC) and is generally concerned with the physical placement of the layout. The DRC works in accordance with a number of layout restrictions which are in turn dependent on the fabrication process. The layout restrictions ensure that the manufactured circuit will connect as desired with no short-circuits or open paths. On the workstation, the DRC can be operated in an incremental and batch mode. If the DRC is operated incrementally, it checks each change as it is made to the circuit mask. If any design-rule violations are found, the DRC will point out exactly where and what the violations are. All the design-rule errors must be corrected completely before going on to the next design step.

After all design rule violations have been corrected, a net list comparison can be performed to check the correctness of the interconnection of the mask layout. In contrast to the DRC that is usually concerned with the geometrical design rules the net list comparison examines the electrical connection rules set out by the

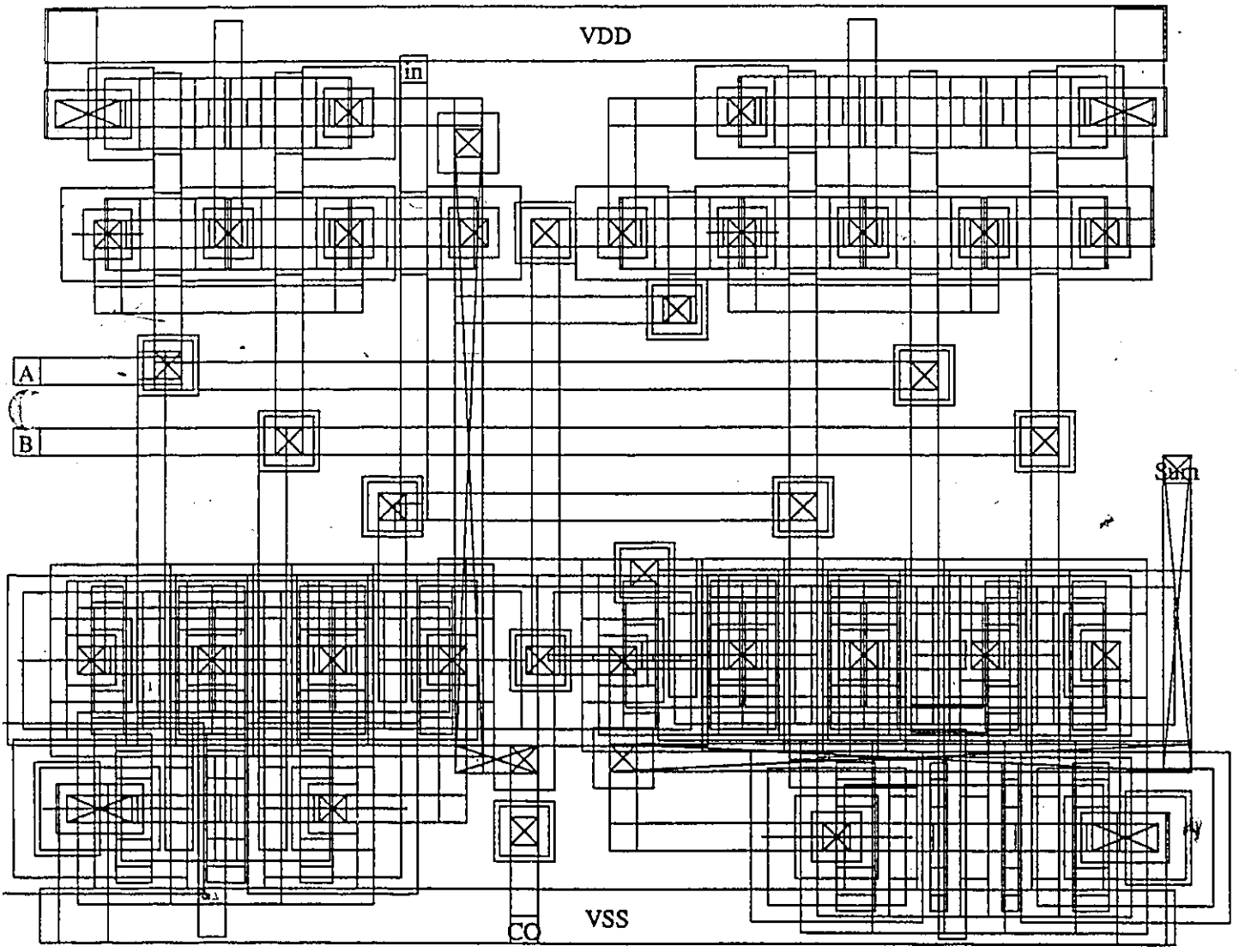


Figure 2.5: Mask layout of the 1-bit adder

designer. For the purpose of comparison, a circuit network must be derived from the compacted mask layout by a node extractor. In most cases, it is beneficial to know whether the circuit network is the desired network. Although simulation and other analysis can tell much about circuit correctness, only the designer knows the true topology, and hence would benefit from knowing whether the derived network is the same.

The mask layout is usually done in a way to correspond to the schematic description of the desired circuit. Since the mask layout is usually done manually an error in the interconnection of the mask circuit is unavoidable. It is much easier to understand and check the schematic network. Generally, the schematic network has known and correct topology while the network derived from the IC layout is less certain. The goal is, therefore, to compare these networks by associating the individual components and connections, If all of the parts associate the networks are the same, however, if there are unassociated parts this indicates how the networks differ. The Electric network software monitors all database activities and updates the connectivity information automatically. This software also provides connectivity information to the design-rule checker and various circuit simulators. In addition to maintaining correct network information, this network tool is also able to compare the schematic network and the mask level network.

Although the designer can determine that the mask level network is the desired circuit with the aid of DRC and netlist comparisons the performance of the mask circuit, such as, speed and power dissipation is not known yet. Therefore, a full analog

simulation must be performed. The transistor connectivity with all parasitic capacitances and resistances are extracted from the mask layout by a circuit extractor that is similar to the one used in capturing a netlist from a schematic description. In the workstation, the SPICE Deck Extractor can extract all the transistors and their connectivity with associated parasitic components and convert the extracted information into a SPICE input format file which can be read into the GSPICE program.

A complete analog simulation of the module cell at the mask level is then performed to provide fine-grain detail about the waveforms at nodes. The SPICE program is the best simulator available to perform the detail simulation. Generally, an analog simulator such as SPICE is used to check the performance and the critical path of a design. In the workstation, another circuit simulator, Relax2 [LeRu82], is also provided. It does not generate the simulation results as accurate as SPICE does, but it takes much less time to compute. The Relax2 package in the workstation serves as an alternative simulator for to the designer. The Relax2 simulator is based on the Waveform Relaxation method [LeRu82] and is an iterative method for analyzing nonlinear circuit systems in the time domain. The method, at each iteration, decomposes the system into several subsystems. Each of the subsystems is then analyzed for the entire given time interval. One of the limitations of Relax2 is that it cannot efficiently simulate a circuit with feedback. The simulation results generated by SPICE and Relax2 can be graphically displayed in the workstation by using a plotting program called PLOT. The PLOT program offers a number of options for users to select, such as,

multiple windows, window size and plotting range. A discussion on the operation of the program and the program listing are included in Appendix VI. Other programs for hardcopy devices are also discussed in this appendix.

If the simulated performance does not meet the design requirements, the designer may have to optimize the mask layout further. For example, the designer may modify the size of the transistors along the critical path to increase the throughput rate.

The design procedures, as mentioned above, are repeated for each submodule circuit. After all submodule cells have been correctly created and optimized the designer can use the mask editor to place the submodule cells together to form the complete circuit. In the workstation, the cell placement can be effectively handled because the connectivity between module cells is always maintained. The mask layout of the complete the 4-bit adder is shown in Figure 2.6. The modularity of the module cell can be examined. The CARRY input is coming from the left side and the CARRY output is leaving from the right side. Therefore, when the module cell is replicated to form an array of four, the CARRY output of one cell is overlaed with the CARRY input of next module cell so that no extra connections are needed.

A hierarchical verification for the complete mask circuit may be performed to ensure that no geometrical design rules or electrical design rules are violated. The next design step deals with input and output (I/O) considerations. For example, how many power pads and ground pads are needed. These I/O pads are usually stored in a standard cell library and the designer can take them from the library

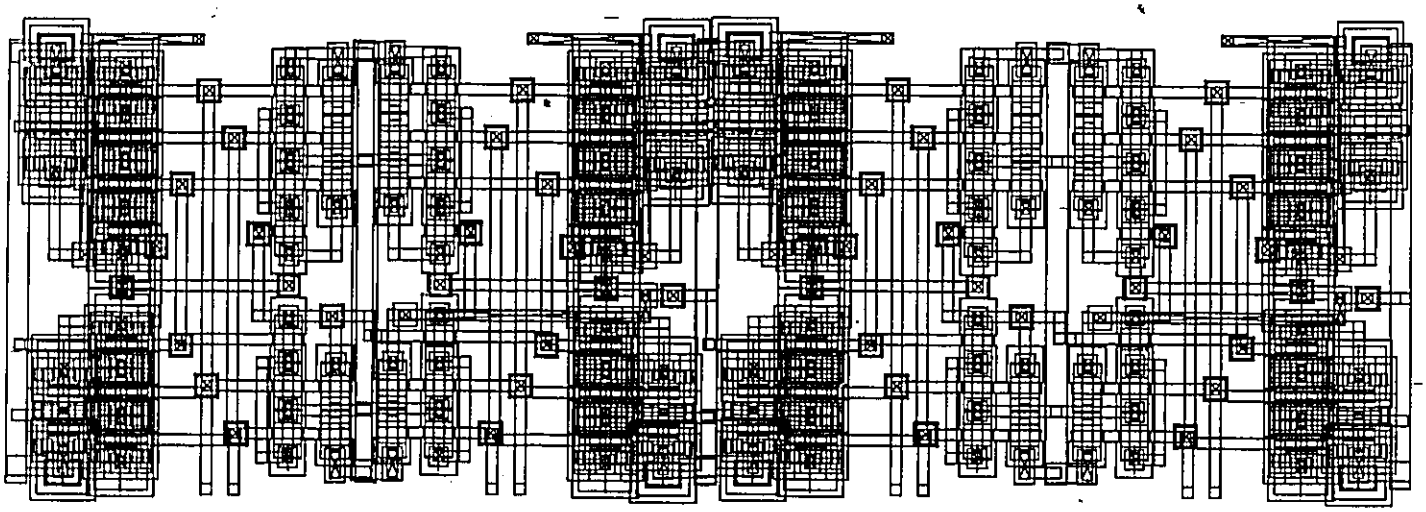


Figure 2.6:Complete mask layout of ripple carry adder

after the complete system layout has been completed. When all the mask modules and I/O pads are placed at their proper locations a final mask layout verification, usually a DRC check, can be performed.

The last design step is to convert the complete mask 's internal representation into a data format that is supported by the silicon foundry. For example, the Canadian Microelectronics Corporation (CMC) supports Caltech Intermediate Format (CIF) as a mask representation. In the workstation, a data file translator program is used to translate between the internal data format file and CIF file. The translator programs called ECIFIN/DCIFIN are explained in more detail in Appendix VII and the source codes are also included. Since CMC has some restrictions on the CIF file, the internal data file has to be carefully converted into a proper CIF file. The translated CIF file can also be read directly and understood by other VLSI tools available in the VLSI Research Laboratory.

To summarize the hierarchical design methodology and the CAD tools available in the design workstation to assist the designer in various design phases, a diagram representing the relationship between the design steps and the VLSI design workstation is shown in Figure 2.7. As illustrated in this diagram, various CAD tools are provided to simplify the design tasks. For instance, a solid line linking the DRC design step and the workstation means that a design-rule checker (DRC) is available to check the geometric for design rule violation. As a whole, all the VLSI CAD tools implemented on the workstation form a fully interactive and functional VLSI design workstation tailored for the full-custom hierarchical design methodology.

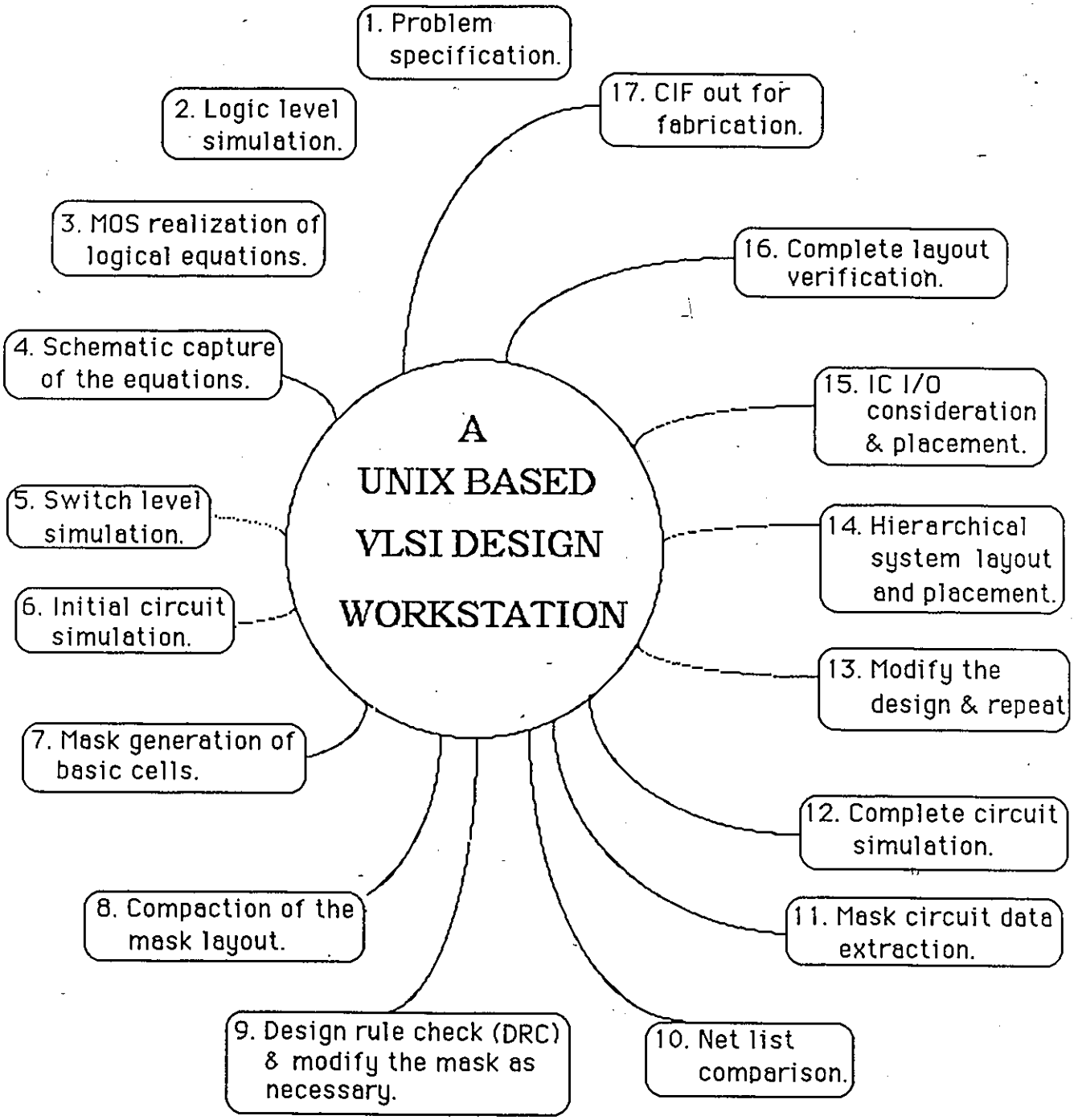


Figure 2.7: Relationship bewteen the Methodology and the Workstation

In order to illustrate the capability of the workstation in more detail two combinational circuit designs have been carried out. The circuit designs include a four-by-four shift register and a transmission gate adder. The mask layout of the shift register was completed in less than two days. Four days were spent for the transmission gate adder. These designs also follow the full-custom hierarchical design method. All circuits have been divided into smaller circuits.

The shift register is based upon a simple latch or D flip-flop circuit. This latch requires two non-overlaing clock signals and their complements. The operation of the latch is relatively simple. Hence, it will not be discussed here but rather interested readers are referred to [WeEs85]. The four-by-four shift register is formed by replicating the latch two-dimensionally. The schematic diagram, simulation results, and mask layouts are shown in Appendix III. The sizes of the D flip-flop and the shift register are 99.6x70.8 micron^2 and 394.8x286.8 micron^2, respectively. A 8-by-8 or 16-by-16 shift register can also be created in this way.

As described previously in this chapter, a 4-bit ripple carry adder was designed. Another adder based on a different approach is given as another design example. This new adder uses the exclusive-or (XOR) gate. The truth table of a XOR gate is tabulated in Table 2.2. The schematic for the XOR circuit is shown in Figure 2.8.

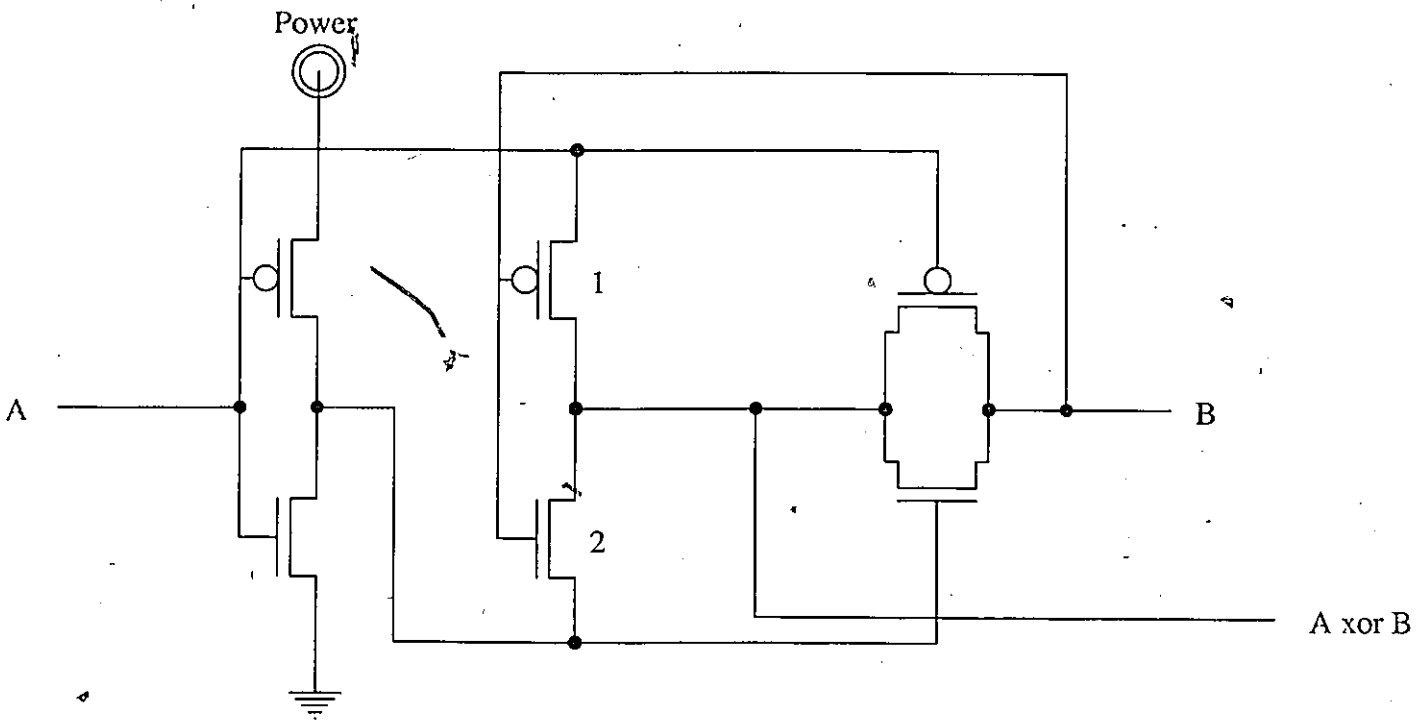


Figure 2.8: Schematic diagram for transmisson gate XOR circuit

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.2: Truth table of an XOR gate

The operation of this circuit is explained as follows:

1 When signal A is logically low, A' is high. The transmission gate is thus closed and transistor pair 1 and 2 is disabled. The output is therefore B.

2 When signal A is logically high, A' is low. Transistor pair 1 and 2 thus act as an inverter and the transmission gate is now open. The output is therefore B'.

With this configuration, a six-transistor XOR gate is formed. By using four transmission gates, four inverters, and two XOR gates, a 1-bit transmission gate adder can be constructed as described in [WeEs85]. The resulting adder has 24 transistors, the same as the ripple carry adder. The new adder offers the advantage of having equal SUM and CARRY delay times. In contrast to the ripple-carry adder where the SUM and CARRY signals are not inverted. The complete schematic, simulation results, and mask layout of a 1-bit transmission gate adder are shown in Appendix III. The simulation results indicate that the performance of the transmission gate adder is better than the ripple carry adder. A 4-bit adder is again created by replicating the 1-bit adder to form a linear array of four. The mask layout of the 4-bit adder is also shown in Appendix III. The size of the 1-bit adder and the 4-bit adder are 159x153.6 micron^2 and 581.85x153.6 micron^2, respectively. This transmission gate adder can also be used in constructing a parallel multiplier.

B. The Role of the Silicon Compiler

As illustrated in the previous section, the mask layout of all the circuit designs is created manually. Although the mask layout editor available in the workstation is easy to use, it still takes several days

or weeks to complete a mask layout. One of the most powerful CAD tools used to produce a desired mask layout representation directly from the behavioral description of a circuit is a silicon compiler. The meaning of the term "silicon compiler" has changed over the years as advances in CAD tools have been made. Initially it implied the concept of specifying some extra switches to a normal programming language compiler so ,for example, it would translate a FORTRAN program into the lowest level mask layout description, rather than into object code. Soon, however, this concept was limited to the use of regular methods for converting logic descriptions into a physical mask layout, such as, those discussed in the last section.

Today's silicon compiler is generally defined as a highly intelligent CAD tool that translates a behavioral description of a circuit design directly into the lowest level description or physical mask layout. Modern silicon compilers still use many of the same steps as do traditional language compilers. Generally, a front-end parser reads the input description and converts it to a structural representation, such as, the IC floor-plan. The back-end then produces the mask layout in two phases. First, a set of pre-defined basic module cells is selected from a cell library. Second, an auto-router is invoked to make the necessary connections between the selected module cells in order to form the desired circuit.

Most contemporary silicon compilers, however, are specialized to produce one type of design. For example, the FIRST silicon compiler, [DeMu84], developed at the University of Ebinburgh, used only bit-serial architectures and was designed for signal processing alications. In highly specialized silicon compilers as in the case of the

one described in the next chapter, the input language can be a very high level one, allowing nonprogrammers to specify chips for their own particular needs. Therefore, a silicon compiler has the potential to replace all the design steps associated with the full-custom MOS realization of logical equations to industrial fabrication specifications.

CHAPTER THREE
A Specialized Silicon Compiler

As was stated in the Introduction, besides creating a fully functional VLSI design workstation, it was desirable to extend the capabilities of the workstation by developing a silicon compiler for a special class of high speed digital signal processing (DSP) applications.

There are several approaches to the design of a VLSI circuit. One may employ a full-custom design effort, such as the one described in the previous chapter, where no existing standard cell libraries are used. In this case the design effort is considerable, but there is an opportunity to create an optimum architecture. Another approach is to simplify a VLSI design by using standard cells and gate-arrays reduce the problem to one of cell selection and interconnection needs. Alternately, a silicon compiler can be used when the designer does not wish to consider the many process dependent steps associated with VLSI design but instead wishes to concentrate on a higher level description of the circuit. As mentioned in Chapter Two, most silicon compilers are designed for those types of applications that use a fixed floor plan and a set of hand-optimized standard cells, such as, adders, memory cells and multipliers. The major reason for developing the Memory Oriented Silicon Compiler (MOSC), described in this chapter, is to provide a tool that allows an IC designer to create RNS oriented realizations of DSP architectures in a rapid manner.

A. General Survey of Silicon Compilers

Since the Signals and Systems Group at the University of Windsor is interesting in digital signal processing applications, a survey of silicon compilers designed for DSP applications was carried out in order to determine state-of-the-art silicon compilation technology and to capture the techniques used in those compilers for constructing the MOSC compiler. Many silicon compilers have been proposed and built during the past decade. A general discussion will highlight three such CAD tools.

1. FIRST: developed at the University of Edinburgh in 1982,
2. LAGER: developed at the University of California in 1986, and
3. BSSC: developed at the GE Corporate R&D Centre in 1987.

These CAD tools are chosen because of their focus on digital signal processing applications as opposed to general circuit design.

1. FIRST

FIRST (Fast Implementation of Real-Time Transforms [DeMR84]) is generally a bit-serial assembly tool [Deny82]. The input to this compiler is a structural description of a circuit, where each bit-serial operator and its interconnections must be specifically given. FIRST users have to take care to specify the exact connections between all basic cells. FIRST also requires users to determine the details of synchronization and timing throughout the complete circuit, and to insert delay cells where necessary in the circuit. This task can be quite tedious and difficult to accomplish, especially when feedback mechanisms are involved.

This compiler first finds the corresponding cells, and then performs the placement of each cell and routes its signals. The floor

plan of circuits generated by this compiler is always fixed, so that the layout scheme may not work efficiently for large circuits, since cells are laid out in two horizontal rows only. The FIRST compiler also provides an interface to a functional simulator that can be used to generate test vectors. The first version of FIRST was built based on 5-micron nMOS technology. As described in [DeMR84], 2.5-micron CMOS technology is currently being investigated for the next version of the FIRST silicon compiler.

2. LAGER

LAGER [RaPB85] was developed at Berkeley and is essentially a datapath compiler with a pre-defined placement of the major components. In other words, circuits generated by LAGER are constructed in a fixed floor plan. The basic components of this datapath compiler are several registers, variable-width shift registers, a variable-width ALU, RAM, ROM and input/output pads. Each circuit generated by LAGER contains the same basic elements, but the specified wordlength will cause the width of the data buses and the number of bit-slices in the ALU and the shift registers to vary.

The input to this compiler is very much like an assembly language program. Data-code stored in the ROM cells to control the datapath is generated from an assembly language-like input file. Therefore, a LAGER user must write out detailed code for each non-primitive operation, and the code must be specific to a given wordlength. The LAGER compiler is particularly not suitable in some applications involving the serial nature of communication between the major components.

3. BSSC

BSSC stands for Bit-Serial Silicon Compiler [YJHN87] and is again based upon bit-serial architecture [Deny82] as the FIRST compiler was. Basically, BSSC is a two-pass compiler. A "C" language type input file is first read into a behavioral-to-structural translator, and then a list of all the necessary cell instances and their interconnections are produced. In other words, a net-list which describes the interconnections of all necessary circuit components is generated. A gate level simulation can also be performed by a low-level simulator. A layout generation system is then invoked to place all the instantianted cells on the chip and route their interconnections.

The structure of all basic cells used in BSSC is well defined. In each cell, every input is 1-bit wide. The power bus and the clock signals are placed along the top of a cell. The ground bus and the complemented clock signals appear along the bottom of the cell. The port locations, for connection purpose, are standardized for all cells. All basic cells for arithmetic operators, relational operators and logical operators are laid out manually based on a 1.25-micron two-level metal CMOS process. One interesting feature of BSSC is that interfaces are provided to various verification tools, such as, a design rule checker and a connectivity checker.

As mentioned above, all of these compilers generate circuits in a fixed floor plan, and use hand-crafted standard cells. The MOSC silicon compiler also follows this trend. However, the MOSC compiler is not a complete circuit mask layout generator but instead it allows

circuit designers to generate basic RNS arithmetic operators in a rapid and efficient manner.

B. Residue Number System and RNS Arithmetic Operations

The MOSC compiler has been developed based upon the concepts of Residue Number System (RNS) [Garn59] and capitalizes on the extensive research conducted by members of the Signals and Systems Group. Before delving into the details of this specialized silicon compiler, let us look at the residue number system. The mathematical advantages and disadvantages of RNS have been known for decades. This mathematically mature topic, however, has not enjoyed the popularity of the weighted magnitude numbering systems, such as, the binary number system. Therefore, a review of RNS will be presented.

Let $(m_1, m_2, \ldots, m_L)$ be a set of relatively prime integers (moduli), ie, no common factor, and let X be an integer in the range of [0,M-1] which is the dynamic range provided by the moduli.

where $$M = \prod_{i=1}^{L} (m_i) \qquad (1)$$

Then by a simple operation, there exists integers k_i, x_i such that

$$X = k_i * m_i + x_i \qquad \text{for } i = 1, 2, \ldots, L \qquad (2)$$

The quantity x_i is called the i^{th} residue of X, and is usually denoted as

$$x_i = X \bmod m_i \qquad (3)$$

For example, we have a number, X = 1462, and a set of moduli (32, 31, 29, 27) is selected. The dynamic range is hence equal to

$$M = 32*31*29*27 = 776736$$

$$M \approx 2^{19.57}$$

It requires almost 20 bits to represent the same number in the binary system. Then,

$$x_1 = 1462 \bmod 32 = 22$$

$$x_2 = 1462 \bmod 31 = 5$$

$$x_3 = 1462 \bmod 29 = 12$$

$$x_4 = 1462 \bmod 27 = 4$$

Hence, $X = (22, 5, 12, 4)$

Obviously X and (M + X) have the same residue representation. Only if X is within the dynamic range [0, M-1], can X then be uniquely determined by the L-tuple $(x_1, x_2, ..., x_L)$. In this case, denoted as

$$X = (x_1, x_2, ..., x_L)$$

Inversion of a residue L-tuple can be performed through the use of the so-called Chinese Remainder Theorem (CRT) [Garn59]. Consider the residue number system with moduli $(m_1, m_2, .., m_L)$ where the corresponding digits are labeled $(x_1, x_2, ..., x_L)$. The following equations define the conversion process.

$$a_1A_1(M/m_1) + a_2A_2(M/m_2) + ...+ a_LA_L(M/m_L) = X \bmod M \quad (4)$$

where

$$A_i(M/m_i) = 1 \bmod m_i \quad (5)$$

For instance, a set of moduli (32, 31, 29, 27) is chosen, and the residue numbers are (22, 5, 12, 4).

Now, $m_1 = 32$, $m_2 = 31$, $m_3 = 29$, $m_4 = 27$

$$M = 32*31*29*27$$

$$M = 776736$$

and $24273\ A_1 = 1 \bmod 32$

$$17\ A_1 = 1 \bmod 32 \Longrightarrow A_1 = 17$$

$25056\ A_2 = 1 \bmod 31$

$8\ A_2 = 1 \bmod 31 \Longrightarrow A_2 = 4$

$26784\ A_3 = 1 \bmod 29$

$17\ A_3 = 1 \bmod 29 \Longrightarrow A_3 = 12$

$28768\ A_4 = 1 \bmod 27$

$13\ A_4 = 1 \bmod 27 \Longrightarrow A_4 = 25$

and $x_1 = 22,\ x_2 = 5,\ x_3 = 12,\ x_4 = 4$

Then,

$24273*17*22 + 25056*4*5 + 26784*12*12 + 28768*25*4 = 16312918$

and $16312918 = X \bmod 776736$

Hence, $X = 1462$

is what was expected.

Let X, Y be within the dynamic range with respect to a set of moduli m_i.

$$X = (x_1, x_2, .., x_L)$$

and

$$Y = (y_1, y_2, .., y_L)$$

If $0 <= XY < M,$

then

$$Z = X \mathbf{o} Y = (z_1, z_2, .., z_L) \qquad (6)$$

where $z_i = (x_i \mathbf{o} y_i) \bmod m_i, \qquad$ for $i = 1, 2, .., L$

and **o** denotes the operation of modular addition, subtraction or multiplication.

Let us now look at examples of RNS addition and multiplication by considering the numbers: (32, 31 29, 27) as a set of moduli.

Suppose $X = 800$ and $Y = 662$

then, for RNS addition,

$$Z = X + Y$$

$$
\begin{array}{rl}
800 = & (\ 0,\ 25,\ 17,\ 17) \\
+\ 662 = & \underline{(22,\ 11,\ 24,\ 14)} \\
1462 & (22,\ 36,\ 41,\ 31) \qquad \text{(a)}
\end{array}
$$

and $\quad Z = (22,\ 5,\ 12,\ 4) \qquad$ (b)

where the result in line (b) comes from the remainder of the result in line (a) with respect to the corresponding moduli.

For RNS multiplication, such as,

$$Z = X * Y$$

$$
\begin{array}{rl}
800 = & (\ 0,\ 25,\ 17,\ 17) \\
*\ 662 = & \underline{(22,\ 11,\ 24,\ 14)} \\
529600 & (\ 0,\ 275,\ 408,\ 238) \\
Z = & (\ 0,\ 27,\ 2,\ 22)
\end{array}
$$

it is clear that the suboperation with each modulus is independent of the other. No carry information needs to be passed between the moduli. The absence of any carry requirements means that the concept of the most and least significant digits is not valid. Thus, parallel architectures can be designed to process all modular partial sums and products concurrently. This parallelism can provide the basis for a speed-up of arithmetic operations, as illustrated in [JeLe77].

RNS arithmetic is exact and therefore free of roundoff error. However, it is this exactness that is often considered as a RNS limitation. The disadvantage of the RNS system is that division, sign detection, and magnitude comparison are inherently difficult operations. Even with these limitations, the usefulness of RNS in digital signal processing applications cannot be denied. For instance, in finite impulse response (FIR) digital filter design, the parallel

realization of RNS addition and multiplication operations results in high speed FIR filters.

As described above, the RNS operations are divided into three major steps: RNS conversion, RNS operations and RNS decoding as illustrated in Figure 3.1.

When interfacing with the binary number system the required RNS conversion can be carried out by a binary-to-residue conversion cell, and the required RNS decoding by a residue-to-binary conversion cell. A number of conversion algorithms [TaRa81] have been developed to perform the conversion process efficiently. Here we shall concentrate on an effective hardware implementation of RNS arithmetic operators. These arithmetic operators may be required in a CRT-based decoder [RaTa86].

C. Memory Oriented Structure for RNS Operations

A common implementation approach used in constructing RNS modular arithmetic operators is based on the table look-up method, [BaJM87], [RaTa86], [BaCo82] and [Tayl82]. High-speed RNS arithmetic operations can be achieved not only because of the parallel nature of RNS, but also since a high-speed residue memory structure is used intensively to store information for the RNS functions. The residue structure is basically a Read-Only-Memory (ROM) cell. A ROM cell can be constructed out of a decoder, a ROM table, and a selector as shown in Figure 3.2. The ROM cell in Figure 3.2 has a n-bit input and a m-bit output.

For example, a residue multiplier for modulo-27 requires two 5-bit inputs (multiplicand and multiplier) and produces a 5-bit output. In this case, n=10 and m=5. The complete multiplication table

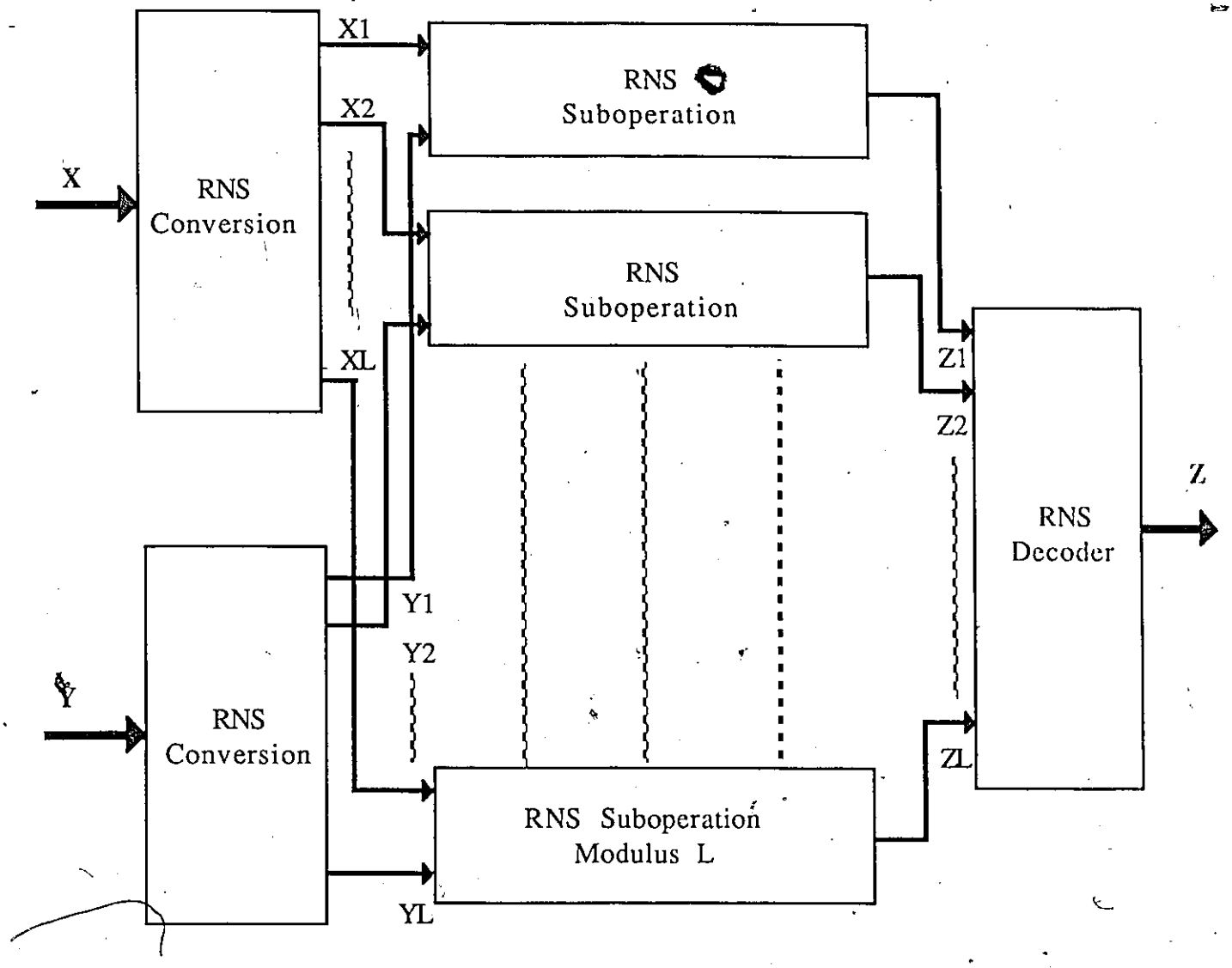


Figure 3.1: Major Components in Closed RNS

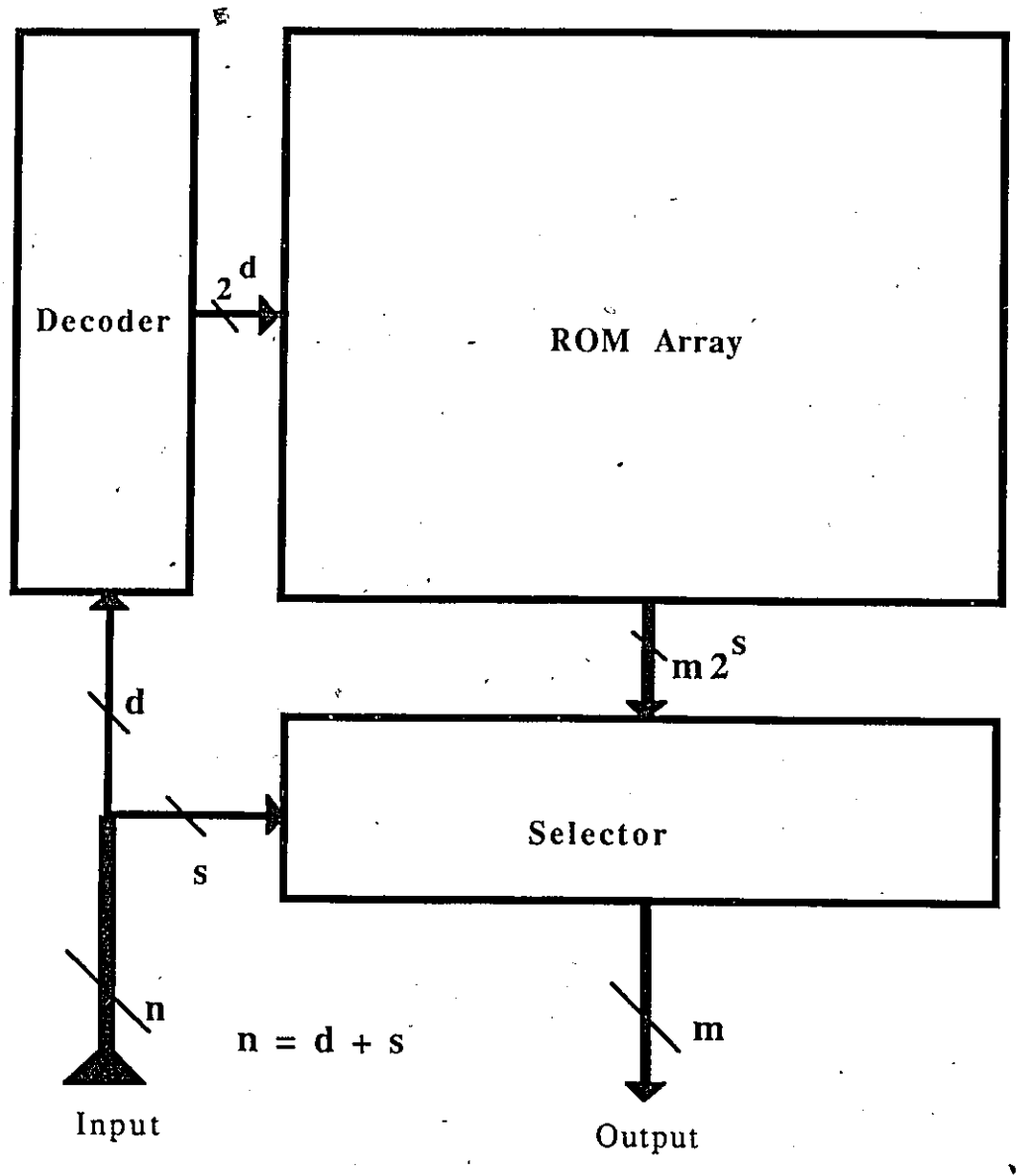


Figure 3.2: A typical ROM structure

can be stored in a single (1024 or 2^{10} X 5)-bit ROM. Since only 729 of the possible 1024 input combinations are valid, the rest of memory cells are not used. On the other hand, the modular multiplier can be implemented in several smaller ROM cells. As described in the previous section, the RNS is a special integer system. Since the multiplication and addition of integers does not require carry information, the operations in each RNS modulus can be performed independently. A curious feature of the RNS is that instead of using a single long wordlength, which provides a large dynamic range, several short wordlength residue structures are parallelled to obtain a sufficient dynamic range while maintaining a fast throughput data flow. Hence, most RNS operations can be formed by dividing the whole operation into a number of suboperations. Each suboperation is realized by cascading a number of the memory oriented cells together to form an array. Each of memory cells has pre-defined memory content. A major advantage in using memory cell arrays to realize the RNS operators is the ease of pipelining for high-speed throughput rate [BaJM87].

A memory oriented architecture which is highly pipelined has been proposed by [TaJM87]. This architecture basically consists of a ROM cell, latches and switching circuitry as shown in Figure 3.3. As illustrated in [TaJM87], this memory oriented structure is appropriated for RNS operations stated in equation (6). As shown in Figure 3.3, if A_i is equal to 1, the output would be equal to the content of a memory cell which is uniquely determined by the input **B**. On the other hand, if A_i is equal to 0, the output is the same as the input **B** because the selection variable A_i forces the input to bypass

the ROM cell. Pipelining is achieved by including a number of latches in the structure and switching the signal lines of input A. Obviously, this structure can only communicate with its closest neighbour. Based on its highly pipelined structure and memory content selection mechanism, this memory oriented structure is chosen as the most fundamental building block used in the MOSC silicon compiler for constructing various RNS operators. If several building blocks are connected, a linear systolic array or ROM array is formed.

D. Hardware Realization of RNS Operators

Once a memory oriented structure has been identified to be the building block for RNS operations, we then wish to examine the structure of modular adder in detail. For example, if we wish to use modulo-m to add two 5-bit numbers, X and Y, we use equation (6).

$$Z = X \mathbf{o} Y$$

$$Z = (X + Y) \bmod m$$

$$Z = (2^4Y_4 + 2^3Y_3 + 2^2Y_2 + 2^1Y_1 + 2^0Y_0 + X) \bmod m \qquad (7)$$

Now equation (7) can be further divided into five simpler steps:

$$sum_0 = (2^0Y_0 + X) \bmod m$$

$$sum_1 = (2^1Y_1 + sum_0) \bmod m$$

$$sum_2 = (2^2Y_2 + sum_1) \bmod m$$

$$sum_3 = (2^3Y_3 + sum_2) \bmod m$$

$$Z = (2^4Y_4 + sum_3) \bmod m$$

The output of each suboperation also has a 5-bit width and is the input of next suboperation, except for the last step. Obviously, five basic cells are required to construct a 5-bit modular adder and they are cascaded to form a linear array, as shown in Figure 3.4. Each cell stores pre-defined memory content.

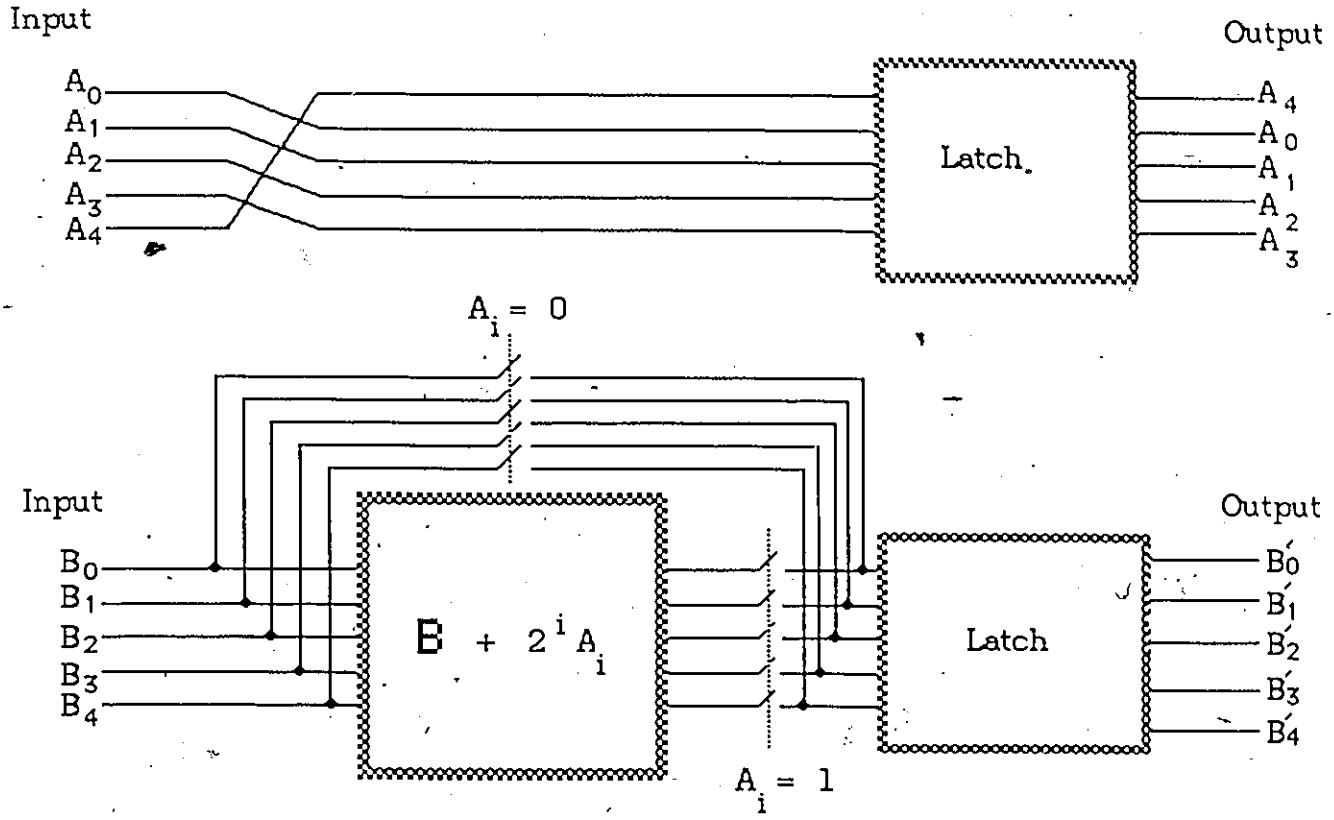
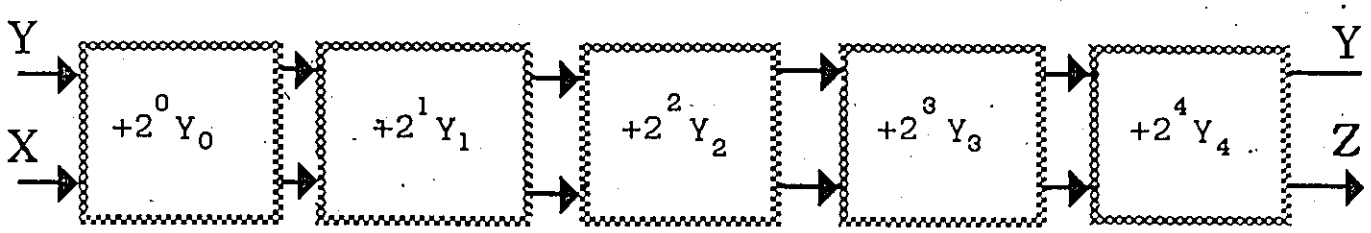


Figure 3.3: Memory oriented structure for RNS operations



$$Z = (X + Y) \bmod m$$

Figure 3.4: Modular 5-bit RNS adder

The memory content of each cell is equal to the input (X, sum_0 to sum_3) plus the appropriate power of two and the input Y acts like a memory content selector. For a modulo-31 adder, the memory content of the five cells is shown in Table 3.1. If we wish to add two numbers: 11 and 25, we have

I/P	*cell0*	*cell1*	*cell2*	*cell3*	*cell4*	*O/P*
	0--> 1	0--> 2	0--> 4	0--> 8	0-->16	
	1--> 2	1--> 3	1--> 5	1--> 9	1-->17	
	2--> 3	2--> 4	2--> 6	2-->10	2-->18	
	3--> 4	3--> 5	3--> 7	3-->11	3-->19	
	4--> 5	4--> 6	4--> 8	4-->12	4-->20	
	5--> 6	5--> 7	5--> 9	5-->13	5-->21	
	6--> 7	6--> 8	6-->10	6-->14	6-->22	
	7--> 8	7--> 9	7-->11	7-->15	7-->23	
	8--> 9	8-->10	8-->12	8-->16	8-->24	
	9-->10	9-->11	9-->13	9-->17	9-->25	
	10-->11	10-->12	10-->14	10-->18	10-->26	
1->	11-->12	11-->13	11-->15	11-->19	11-->27	
	12-->13	12-->14	12-->16	2->12-->20	12-->28	
	13-->14	13-->15	13-->17	13-->21	13-->29	
	14-->15	14-->16	14-->18	14-->22	14-->30	
	15-->16	15-->17	15-->19	15-->23	15--> 0	
	16-->17	16-->18	16-->20	16-->24	16--> 1	
	17-->18	17-->19	17-->21	17-->25	17--> 2	
	18-->19	18-->20	18-->22	18-->26	18--> 3	
	19-->20	19-->21	19-->23	19-->27	19--> 4	
	20-->21	20-->22	20-->24	20-->28	3->20--> 5	>>>>>>5*
	21-->22	21-->23	21-->25	21-->29	21--> 6	
	22-->23	22-->24	22-->26	22-->30	22--> 7	
	23-->24	23-->25	23-->27	23--> 0	23--> 8	
	24-->25	24-->26	24-->28	24--> 1	24--> 9	
	25-->26	25-->27	25-->29	25--> 2	25-->10	
	26-->27	26-->28	26-->30	26--> 3	26-->11	
	27-->28	27-->29	27--> 0	27--> 4	27-->12	
	28-->29	28-->30	28--> 1	28--> 5	28-->13	
	29-->30	29--> 0	29--> 2	29--> 6	29-->14	
	30--> 0	30--> 1	30--> 3	30--> 7	30-->15	
	31--> 1	31--> 2	31--> 4	31--> 8	31-->16	

Table 3.1: Memory content for 5-bit modular addition

$$Z = (X + Y) \bmod m$$
$$= (11 + 25) \bmod 31$$

and the number 25 can be represented in a 5-bit binary form as 11001. Since the number Y = 25 is the memory content selector, *cell0, cell3* and *cell4* are selected. As illustrated in the Table 3.1, in the first step, the number 11 acts as an address to the first memory cell. The output of 12 is then used in the second step as an address to *cell3*. The output is 20. Again, the number 20 is the input to *cell4*. The corresponding output is 5 so that the final result is 5. Now, if we double check the result, we have

$$Z = (11 + 25) \bmod 31$$
$$= (36) \bmod 31$$
$$= 5$$

As was obtained from the five cascaded memory cells.

Once the concept of the modular addition using the five cascaded memory cells is understood, the same memory cells, with different memory content, can be used to construct a modular subtractor. RNS subtraction is accomplished using the additive inverse of the positive residue representation [Garn59]. The additive inverse of a residue number, n, is defined by the following equation,

$$n + n' = 0 \qquad (8)$$

For example, if we wish to subtract 25 from 11, we have

$$Z = (X - Y) \bmod m$$
$$= (11 - 25) \bmod 31$$
$$= (-14) \bmod 31 \qquad \text{(because } (17 + 14) \bmod 31 = 0\text{)}$$
$$= 17 \qquad (9)$$

The memory contents of the five cells used for subtraction are tabulated in Table 3.2. Using the same procedures as in the modular addition, one would get 17 as is indicated in step 3 shown in Table 3.2. The result is also equal to the one in (9).

I/P	*cell0*	*cell1*	*cell2*	*cell3*	*cell4*	*O/P*
	0-->30	0-->29	0-->27	0-->23	0-->15	
	1--> 0	1-->30	1-->28	1-->24	1-->16	
	2--> 1	2--> 0	2-->29	2-->25 3->2-->17	>>>>>17*	
	3--> 2	3--> 1	3-->30	3-->26	3-->18	
	4--> 3	4--> 2	4--> 0	4-->27	4-->19	
	5--> 4	5--> 3	5--> 1	5-->28	5-->20	
	6--> 5	6--> 4	6--> 2	6-->29	6-->21	
	7--> 6	7--> 5	7--> 3	7-->30	7-->22	
	8--> 7	8--> 6	8--> 4	8--> 0	8-->23	
	9--> 8	9--> 7	9--> 5	9--> 1	9-->24	
	10--> 9	10--> 8	10--> 6 2->10--> 2		10-->25	
1->	11-->10	11--> 9	11--> 7	11--> 3	11-->26	
	12-->11	12-->10	12--> 8	12--> 4	12-->27	
	13-->12	13-->11	13--> 9	13--> 5	13-->28	
	14-->13	14-->12	14-->10	14--> 6	14-->29	
	15-->14	15-->13	15-->11	15--> 7	15-->30	
	16-->15	16-->14	16-->12	16--> 8	16--> 0	
	17-->16	17-->15	17-->13	17--> 9	17--> 1	
	18-->17	18-->16	18-->14	18-->10	18--> 2	
	19-->18	19-->17	19-->15	19-->11	19--> 3	
	20-->19	20-->18	20-->16	20-->12	20--> 4	
	21-->20	21-->19	21-->17	21-->13	21--> 5	
	22-->21	22-->20	22-->18	22-->14	22--> 6	
	23-->22	23-->21	23-->19	23-->15	23--> 7	
	24-->23	24-->22	24-->20	24-->16	24--> 8	
	25-->24	25-->23	25-->21	25-->17	25--> 9	
	26-->25	26-->24	26-->22	26-->18	26-->10	
	27-->26	27-->25	27-->23	27-->19	27-->11	
	28-->27	28-->26	28-->24	28-->20	28-->12	
	29-->28	29-->27	29-->25	29-->21	29-->13	
	30-->29	30-->28	30-->26	30-->22	30-->14	
	31-->30	31-->29	31-->27	31-->23	31-->15	

Table 3.2: Memory content for 5-bit modular subtraction

With a pre-defined memory content in the cascaded memory cells, the savings in the number of memory locations is extremely attractive. If a single regular ROM cell is used to store all possible outputs of 5-bit modular addition or subtraction, it would take 2^{10} or 1024 memory locations. However, only 160 memory locations are needed to perform the same operation in this linear memory array.

Based upon the RNS addition and subtraction operations, we can construct a modular multiplier. A modulo-m multiplier can be realized using the Quarter Square Multiplication method [BaJM87a]:

$$Z = (X * Y) \bmod m$$

$$= (((X+Y)^2/4) \bmod m - ((X-Y)^2/4) \bmod m) \bmod m \quad (10)$$

where the quantity $(((*)^2/4) \bmod m)$ is also stored in a lookup table. According to the equation (10), one addition, two subtractions and two quarter square operations are required. A conceptual block diagram of the modular multiplier is shown in Figure 3.5. The memory content, C, of a quarter square operator is determined by the following equation:

$$C = (\lfloor address^2/4 \rfloor) \bmod m \quad (11)$$

The content C for modulo-31 is shown in Table 3.3. For instance, if we wish to use modulo-31 to multiply two residue number; 11 and 25, we have

$$Z = (X * Y) \bmod m$$

$$Z = (11 * 25) \bmod 31$$

$$Z = (275) \bmod 31$$

$$Z = 27$$

Now let us use the quarter square multiplication method.

$$Z = (11 * 25) \bmod 31$$

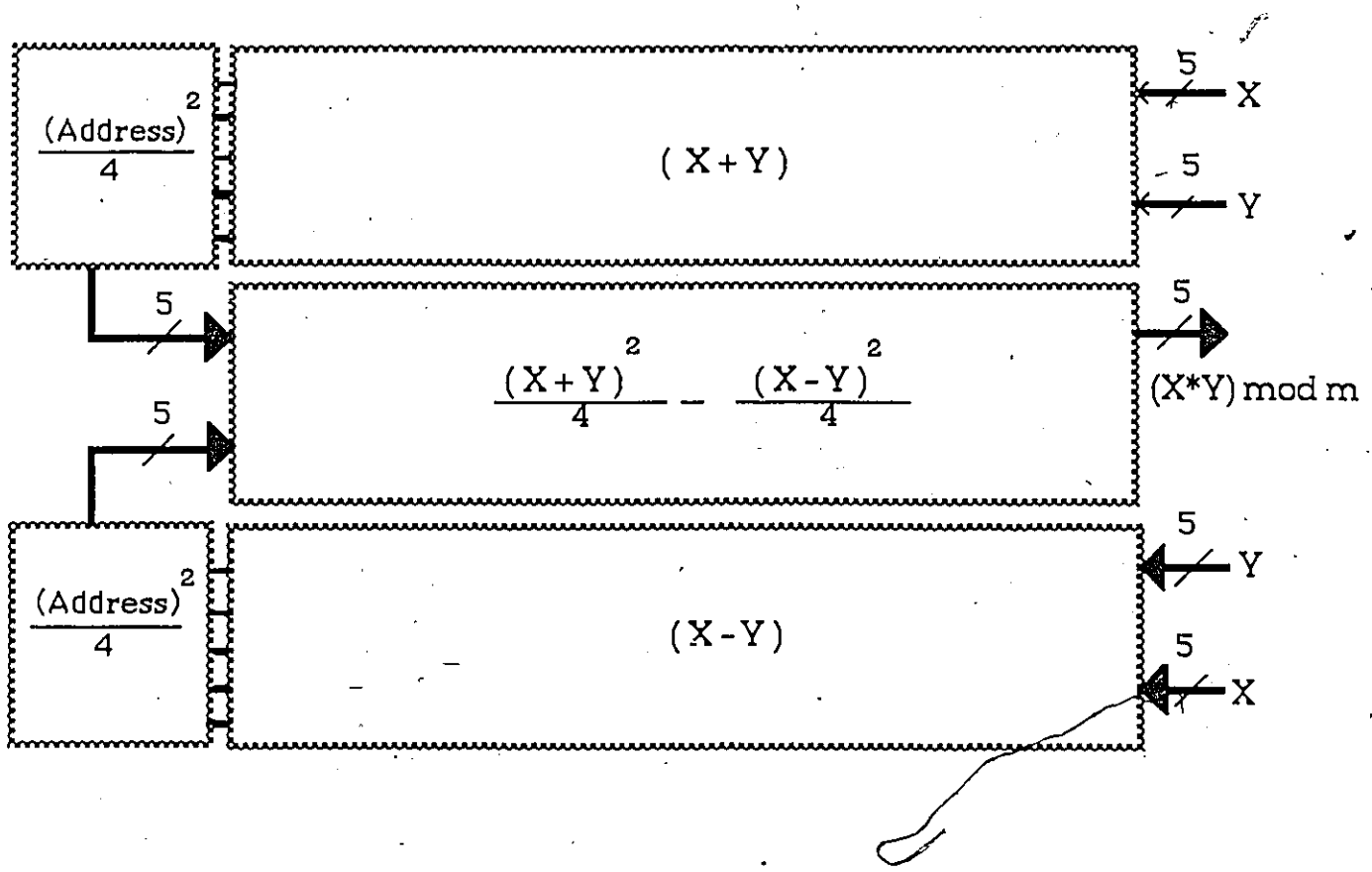


Figure 3.5: Modular multiplier structure using quarter square method

Then, $Z = ((11+25)^2/4 - (11-25)^2/4) \bmod 31$

From the previous addition and subtraction examples, we have

$(11 + 25) \bmod 31 = 5$

and $(11 - 25) \bmod 31 = 17$

0---> 0	11---> 30	22---> 28
1---> 0	12---> 5	23---> 8
2---> 1	13---> 11	24---> 20
3---> 2	14---> 18	25---> 1
4---> 4	15---> 25	26---> 14
5---> 6*	16---> 2	27---> 27
6---> 9	17---> 10*	28---> 10
7--->12	18---> 19	29---> 24
8--->16	19---> 28	30---> 8
9--->20	20---> 7	31---> 23
10--->25	21---> 17	

Table 3.3: Memory content of quarter square operator

The numbers 5 and 17 are now the address to the quarter square operator and the outputs are 6 and 10 respectively, as shown in Table 3.3. Next the numbers 6 and 10 are used as the inputs to the modular subtractor as follows:

$(6 - 10) \bmod 31 = 27$

Using the same procedures as in the previous subtraction example, we would get the result, 27, by using the Table 3.2. Thus the modular multiplier consisting of 17 basic memory cells, as shown in Figure 3.5, has been demonstrated to perform the modular multiplication successfully.

E. Implementation of the MOSC Silicon Compiler

The MOSC (memory oriented silicon compiler) is developed around the memory oriented structure used to construct the RNS adders, RNS subtractors and RNS multipliers. The MOSC silicon

compiler is an application-specific computer-aided design (ASCAD) tool for producing very high speed RNS adders, subtractors, multipliers and constant operators in a rapid manner. The constant operator manipulates a variable by a constant value. Since the MOSC is a high-level ASCAD tool, an IC designer only needs to specify 1) RNS operator type, 2) modulo value, and/or 3) a constant. In order to be useful, the MOSC compiler should reduce costs such as:

1) *Design cost*: The tool should be easy to use, reducing substantially the human time required to specify the design.

2) *Fabrication cost*: The tool should produce a circuit whose function is correct by construction, whose major component count is low and whose components are area-efficient .

3) *Operation cost*: The tool should produce a circuit that is efficient with respect to its power dissipation and operational speed.

For the MOSC compiler the *operation cost* is basically dependent of the complexity of the memory oriented cell. Since the class of circuits the MOSC compiler creates is sufficiently specific and an effective memory oriented structure is employed, these costs can be largely reduced.

The MOSC based design process basically consists of a design specification, a cell library and CIF files as shown in Figure 3.6. The cell library presently contains only two basic cells that were manually created based on the residue architecture, as described in the section C.

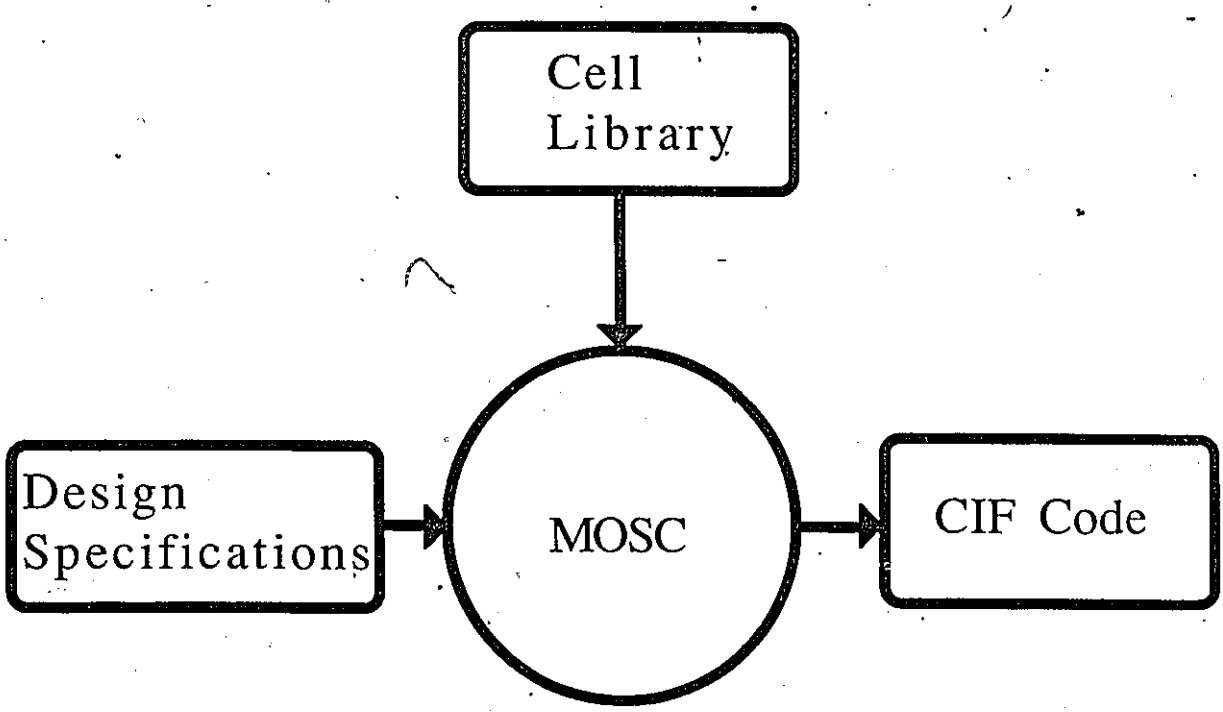


Figure 3.6: MOSC process structure

Let us look at the first cell that was designed by [Bird87]. This memory cell was designed to accommodate a wordlength of five bits so that the maximum modulo-number is $2^5 = 32$. A more detailed structure for the memory cell is shown in Figure 3.7. As illustrated in Figure 3.7, there are 32X5 memory locations for storing the pre-defined memory content. We will not describe the detailed aspects of this memory cell design here but instead refer the reader to [Bird87] which also provides detailed description of circuit performance. The memory locations are configured as a 8X4 matrix in order to obtain a square shape layout. The size of the cell is 421.2 * 377.4 micron^2. The clock period is 24ns and the power consumption from simulation result is 3.2 mWatts at 20 MHz [Bird87] as computed by the SPICE simulation program. The cell was designed in such a way that all

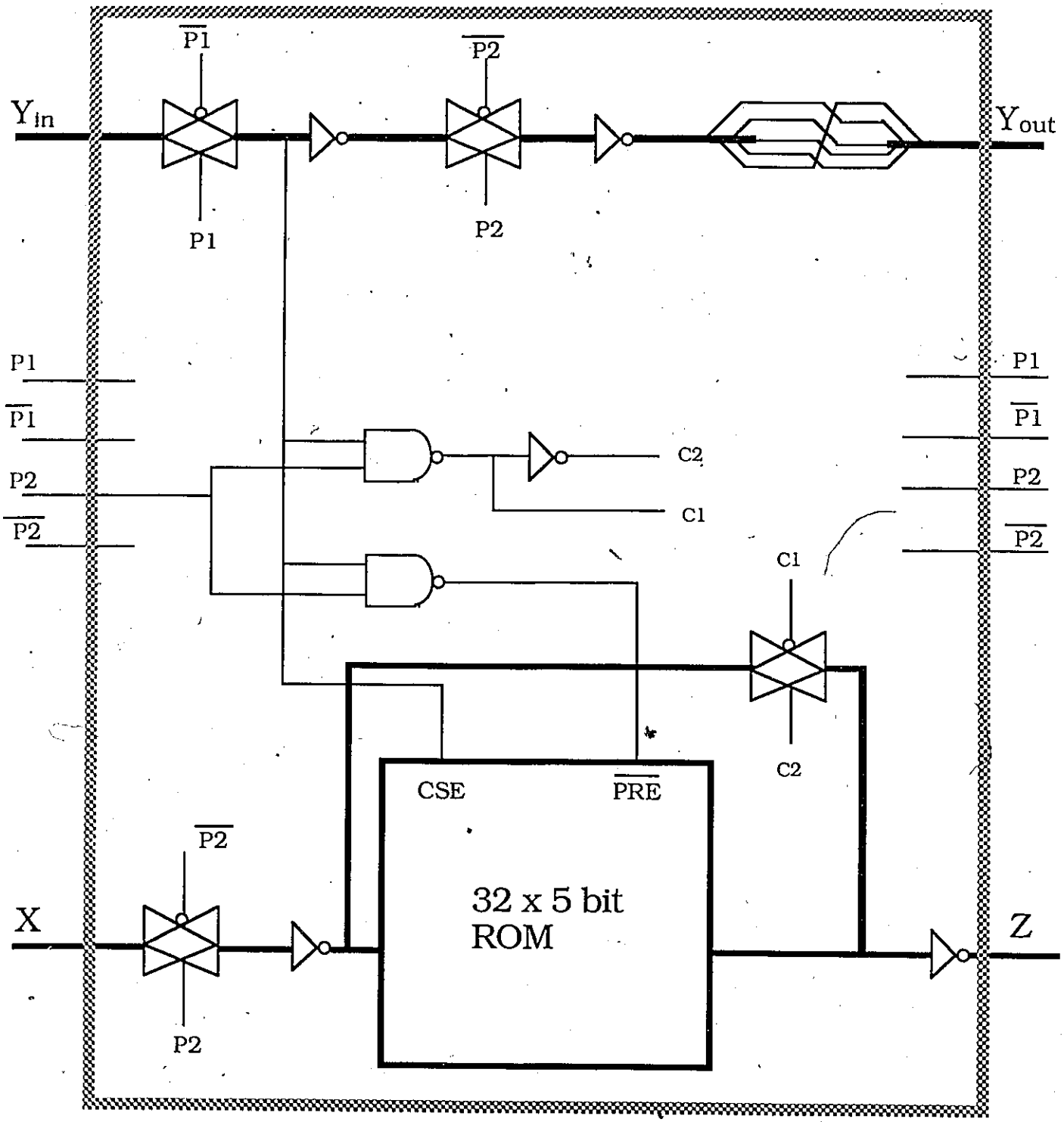


Figure 3.7: Detail structure of the first memory cell

memory locations were left empty. In order to store "1" bit of information in any particular location, a 9X13 diffusion layer (CF) polygon is placed on top of the location as shown in Figure 3.8.

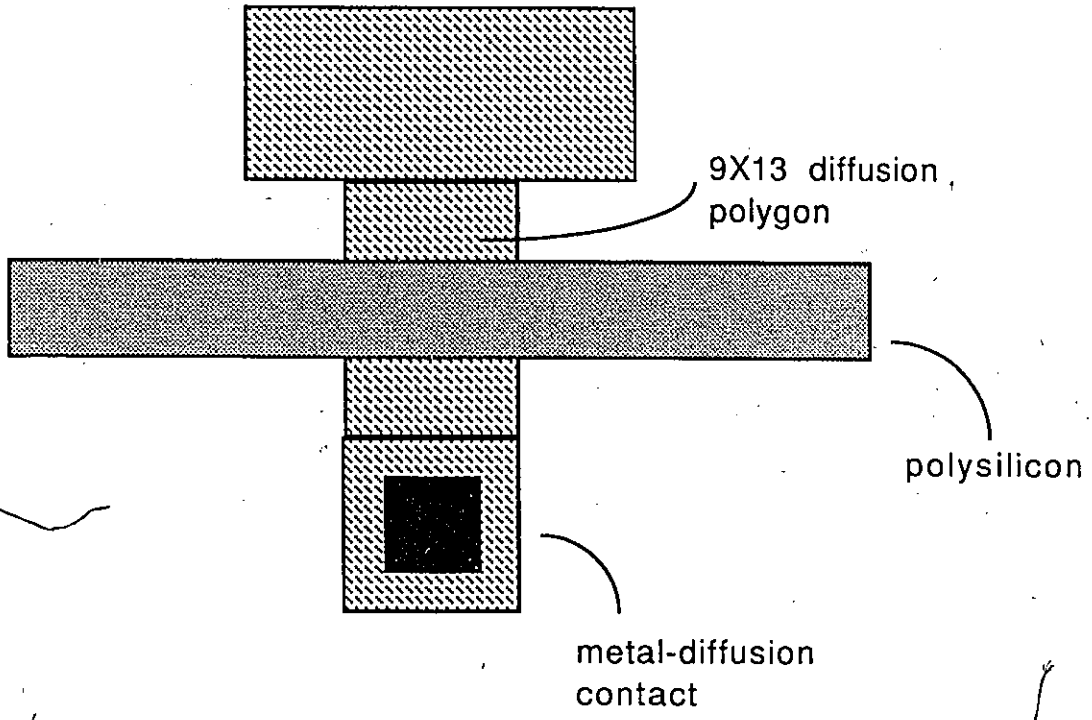


Figure 3.8: "1" bit transistor

The other memory cell was designed by [Raja87]. In addition to the 5-bit wordlength, two extra bits are used to accomplish the fault detection mechanism built into this memory cell. Hence, there are 7 bits of information at each address. The 5^{th} bit represents the parity of the address and the 6^{th} bit represents the parity of the content. An even parity scheme is used in the MOSC compiler. The fault detection mechanism is based on the fact that the content parity in a cell is equal to the address parity in the following cell [TaJM87a]. The size of the memory cell is 842.4X867.6 micron^2. The memory

locations are also configured as a 8X4 matrix. The memory content occupies an area of 216.6X295.2 micron^2. A detailed design description of the memory cell can be found in [Raja87] so it will not be repeated here. In order to store a "0" bit of information in a memory location, four polygons (1 diffusion CF, 1 metal CM and 2 contact cuts CC) are needed. The mask layout of the cell has been improved in such a way that when the cell is replicated to form a linear array, the output signal lines of one cell overlap the input signal lines of the next cell.

Once the actual memory cells are identified, we can consider the implementation of the MOSC compiler, a flow chart of which is shown in Figure 3.9. As illustrated in the flow chart, the MOSC compiler requires a minimum amount of human interaction. Once a user has supplied sufficient information, the MOSC compiler will produce a hierarchical CIF file. The MOSC compiler produces four types of RNS operators: adders, subtractors, multipliers and constant operators. In some cases, a designer may wish to add (or subtract) a constant to (or from) a variable. The MOSC compiler, therefore, includes this capability to allow the user to specify a constant value to be stored in the memory content. In the user environment, a user is only required to supply the following information:

1) Operator type: Adder/Subtractor/Multiplier/Constant operator
2) Number of operators.
3) Modulo number of each operator.
4) Constant value.

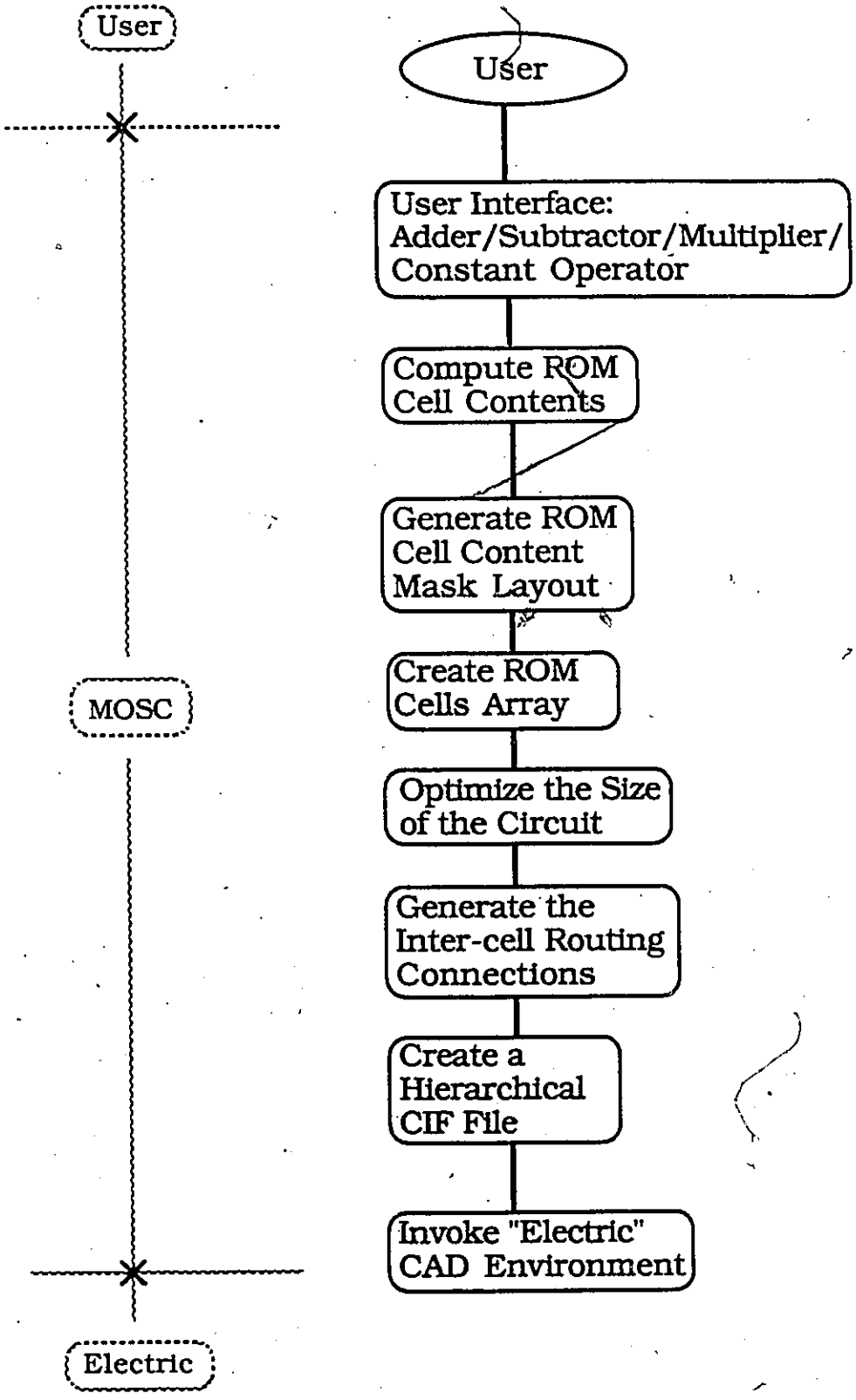


Figure 3.9: Flow chart of the MOSC silicon compiler

The MOSC compiler then computes the memory content for each memory cell. The content is expressed in a binary form. For example, the content 10 at address 5 is expressed as 00 01010. The two bits "00" present the parities of address and content, respectively. Mask layout corresponding to the binary data is then generated. This mask generation is dependent on the basic memory cell selected by the MOSC compiler. In the first memory cell, a subcell is created to represent a 9X13 diffusion layer polygon that is properly centered. The mask generator repeatedly uses the *Call* command, one of the CIF commands, to duplicate the subcell and to place it at proper locations in the memory cell. In the second memory cell, since the mask layout of the memory content is not organized in a regular structure, four polygons must be generated for each memory location. The mask generator can be easily changed to adopt to a new memory cell, and a detailed explanation is included in Appendix VIII. All program listings of the MOSC silicon compiler are also included in this appendix.

After all the mask layouts have been generated, the basic memory cell is pulled out from the cell library to map on top of the mask layout of the memory content, to form a complete RNS memory oriented cell. Then, a linear memory array is created for the desired RNS operator. The MOSC compiler will place all cells as close as possible in order to obtain a minimum chip size. When creating a number of adders or subtractors with different modulo numbers, the operators are properly flipped, and interconnection wires are created to connect the operators together. Finally, hierarchical CIF code is generated and stored in a file specified by the user. For example, the

hierarchy of a modulo-32 adder described in CIF code is shown in Figure 3.10. Five new cells, *DATAC_150*, *DATAC_151*, *DATAC_152*, *DATAC_153* and *DATAC_154*, which represent the memory content, are created and mapped with the basic memory cell, *NEWMOD2*, to form the complete RNS cells: *ROM_FTR1*, *ROM_FTR2*, *ROM_FTR3*, *ROM_FTR4* and *ROM_FTR5*. These five RNS cells are cascaded to form the adder. The top hierarchical level of the desired circuit is always specified by a cell called *CHIP*.

The MOSC compiler then invokes the **Electric** CAD facility to allow the user to view and modify the complete mask layout of the generated circuit. This process is accomplished by creating a *.cadrc* file in the user's login directory. When Electric starts, it reads commands from the *.cadrc* file. Only those commands on lines that begin with the keyword *electric* will be executed. Therefore, if the MOSC compiler places proper commands in the *.cadrc* file and then invokes Electric, the Electric package will turn off the incrememtal DRC and read the specified CIF file into its internal database. The top hierarchical level of the circuit is then displayed on the screen.

A circuit designer, sitting at the terminal, can now overrule the compiler and customize parts of the design. He can also let the compiler generate only the memory content in his custom design. The designer is only required to add I/O pads to complete the design.

F. Software Development Environment

Most of MOSC subprograms were written in the highly-portable "C" programming language. A program, *systolic* which produces mask CIF codes for the second memory structure, was written in the Berkeley PASCAL language. The purpose of writing the program in

Structure of library adder.cif

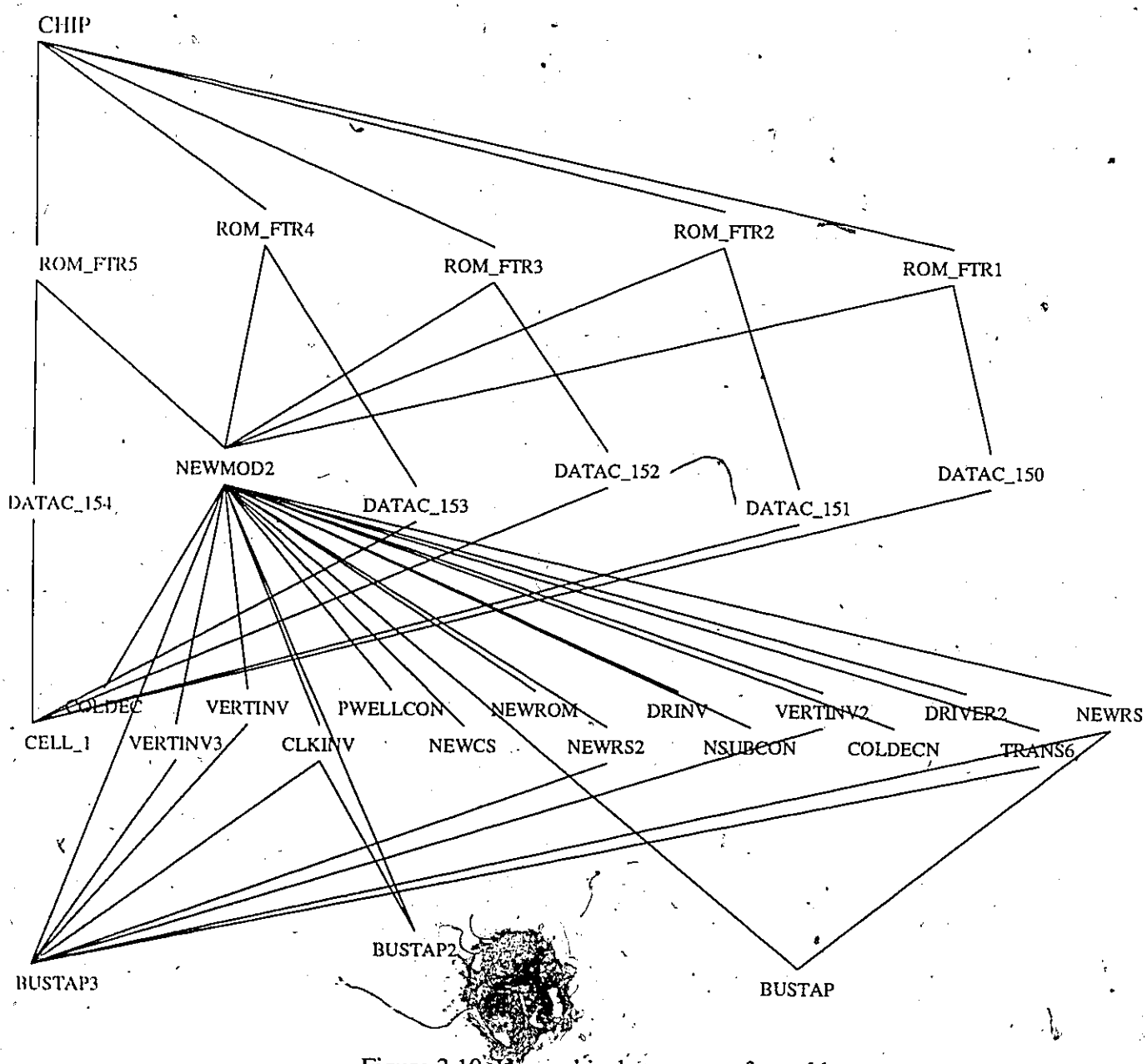


Figure 3.10: Hierarchical structure of an adder

PASCAL was to gain experience with alternative compiler available in the Unix operating system. It proved to run slower than the one written in "C" and it was also more difficult to program.

The MOSC compiler consists of seven main programs and a number of command files which call the main programs in order to perform certain tasks and to construct the desired RNS arithmetic operators. The programs were developed in such a way that a user can run the programs independently and keep the intermediate files so that the user can have maximum control of the data. In addition, command files were written to speed up the design process if the user is interested only in the end-product, CIF file. For example, if we would like to write a command file for building a RNS multiplier, we may produce a file as shown in Figure 3.11. Lines (6-7) in the diagram produce an adder; lines (9-10) produce a subtractor; lines (12-13) produce a quarter square operator; and line (16) finally merges all files together to form a hierarchical CIF file of the multiplier. The rest of the lines are to remove all intermediate files.

If a user would like to update the MOSC compiler they do not have to modify the whole compiler, but only change one or two subprograms. For example, if the fault detection mechanism is incorporated into the first memory structure, only the *gen_mask* program has to be modified in order to take care of the two extra bits. The MOSC compiler can also be expanded easily using the existing programs. Eventually, writing a command file is analogous to writing a structural description of a design using the basic commands, such as, addition or multiplication.

```
# multiplier.com output-file                                    %1
# create those adder.dat, subtractor.dat and modnumber.dat      %2
# for modulo number.                                            %3
modnum                                                          %4
# create adder bit data                                         %5
gen_data bit01 < adder.dat                                      %6
masknew bit01 c1 c2 c3 c4 c5 < cellnum21                        %7
# create substractor data                                       %8
gen_data bit01 < subtractor.dat                                 %9
masknew bit01 cc1 cc2 cc3 cc4 cc5 < cellnum26                   %10
# create addresscontent data                                    %11
multaddss bit01 < modnumber.dat                                 %12
masknew bit01 ccc1 < cellnum31                                  %13
# layout the floor planning for the multiplier                  %14
# create hierarchical CIF code                                  %15
petermult $1 cif.cif c1 c2 c3 c4 c5 cc1 cc2 cc3 cc4 cc5 ccc1    %16
# remove the un-wanted files                                    %17
rm c1 c2 c3 c4 c5 cc1 cc2 cc3 cc4 cc5 ccc1                      %18
rm bit01                                                        %19
```

Figure 3.11: A command file for creating a RNS multiplier

The MOSC silicon compiler is currently running on the VAXstation II/GPX under the Ultrix operating system. The MOSC compiler can be easily transported and installed on the VAX-VMS operating system so that other layout facilities, ie, Applicon and Daisy, can be incorporated with the compiler. Foreign commands in VMS must be created in the user's *.login* file. The foreign commands are used to allow a user to run a program and specify the input/output files on the same VMS command line.

G. Design Examples of MOSC

In this section, we will illustrate the features of the MOSC silicon compiler by using examples. The first chip produced by this compiler is a 4-adder with moduli (32, 31, 29, 27). The input to the compiler is shown in Figure 3.12. As illustrated in the Figure 3.12, a

user invokes the compiler by typing **mosc** followed by an output file name that is used to store the CIF file of the *4-adder* circuit. The user must provide information at each "==>" prompt. The top level of the mask layout for this adder is shown in Figure 3.13. Approximately 1660 mask polygons have been generated by the compiler for this circuit. The size of the chip is 2142 x 1506.6 micron^2. The actual mask layout is shown in Appendix IX.

```
% mosc 4adder.cif
        ********************************
        * MOSC SILICON COMPILER *
        ********************************

        Adder/Multiplier/Constant? (A/M/C)==> a

        ****************************************
        ***    VLSI adders/subtractors   ***
        ****************************************

        Each adder/subtractor consists of 5 modulus.
        Each module has specified ROM content.

        Enter number of adder:==> 4
        Enter the modulo number:==>32
        Adder/subtractor? (A/S)==> a
        Enter the modulo number:==>31
        Adder/subtractor? (A/S)==> a
        Enter the modulo number:==>29
        Adder/subtractor? (A/S)==> a
        Enter the modulo number:==>27
        Adder/subtractor? (A/S)==> a

        *** Silicon compilation finishes
```

Figure 3.12: Input example of a 4-adder to MOSC.

The next example considers a RNS multiplier of modulo-32. The input to the MOSC compiler is shown in Figure 3.14. As shown in the Figure 3.14, the user has to provide only two pieces of data to

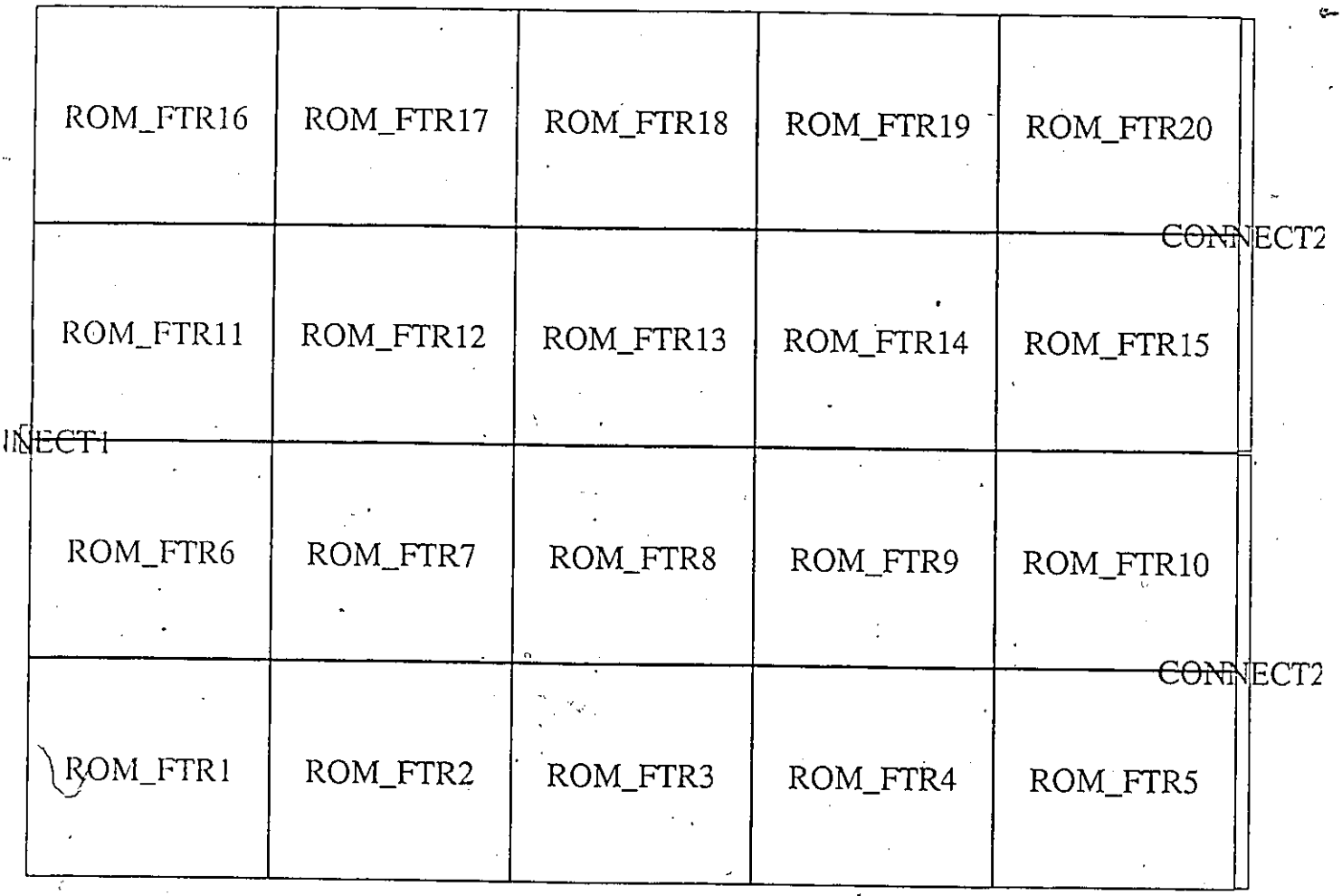


Figure 3.13: Mask layout of the 4-adder.

produce a modular multiplier. The mask layout of the multiplier is shown in Figure 3.15. The MOSC compiler creates approximately 920 mask polygons for this chip. The size of the chip is 2518.2 x 1132.2 micron^2.

```
% mosc multiplier32.cif

**********************************
* MOSC SILICON COMPILER *
**********************************

Adder/Multiplier/Constant? (A/M/C)==> m
Enter the modulo number:==> 32
Enter the modulo number:
Adder/Subtractor?(A/S)
Enter the starting cell number DS#:
Enter the starting cellname number CEL_#:
Enter the modulo number:
Adder/Subtractor?(A/S)
Enter the starting cell number DS#:
Enter the starting cellname number CEL_#:
Enter the modulo number:
Adder/Subtractor?(A/S)
Enter the starting cell number DS#:
Enter the starting cellname number CEL_#:

*** Silicon compilation finishes.
```

Figure 3.14: Input example to MOSC for modular multiplier.

The final example presented here is also a modular multiplier, but it multiplies a 5-bit input value, X, by a constant C (C=12 in this example). The input to the MOSC compiler is shown in Figure 3.16. The user must supply modulo-value and a constant. As illustrated in Figure 3.16, the user can also add/subtract an input value to/from a constant by typing *add* or *sub* in the third prompt. The size of the circuit is 421.2*377.4 micron^2. In this example, the compiler

ROM_FTR11	ROM_FTR5	ROM_FTR4	ROM_FTR3	ROM_FTR2	ROM_FTR1
	ROM_FTR6	ROM_FTR7	ROM_FTR8	ROM_FTR9	ROM_FTR10
ROM_FTR11	ROM_FTR10	ROM_FTR9	ROM_FTR8	ROM_FTR7	ROM_FTR6

Figure 3.15: Mask layout of a modulo-32 multiplier.

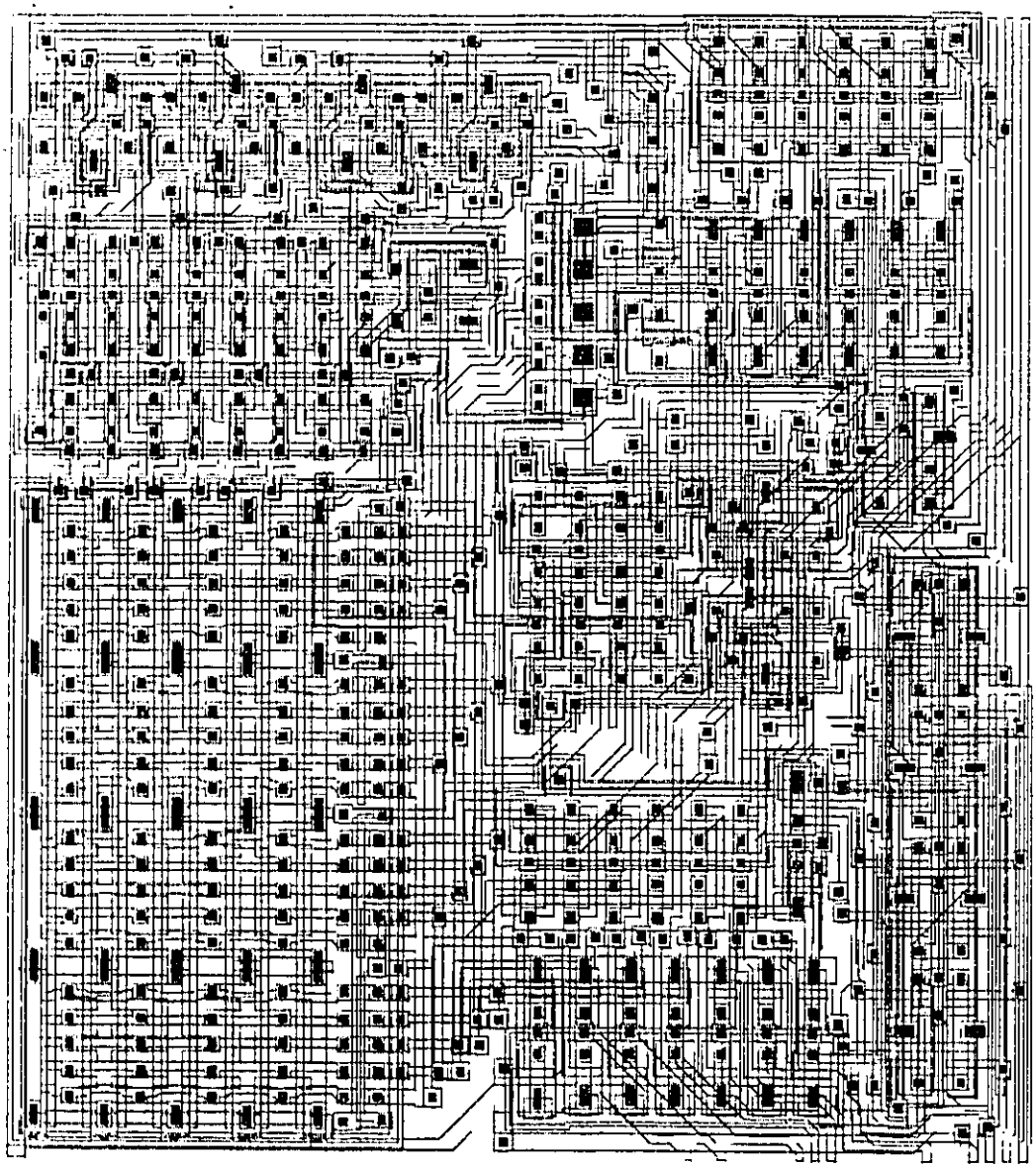
produces total of 53 CIF lines to describe the appropriate memory content. The mask layout of the circuit is shown in Figure 3.17

```
% mosc const_mult_12.cif
**********************************
* MOSC SILICON COMPILER *
**********************************

Adder/Multiplier/Constant? (A/M/C)==> c

**********************************
***     +/-/* Constant     ***
**********************************

Enter the modulo number:==> 32
Select Adder/Subtractor/Multiplier (add/sub/mul)==> mul
Enter the constant value:==> 12
Enter the starting cell number DS#:
Enter the starting cellname number CEL_#:

*** Silicon compilation finishes
```

Figure 3.16: Input example to MOSC for constant multiplier

More examples can be found in Appendix X that show various modular operators with the second memory structure. If a designer just wishes to build a modular operator, they only have to add several I/O pads around the desired circuit. On the other hand, if a finite impulse response (FIR) digital filter is designed, only adders and multipliers are needed in accordance with FIR equation (12). The filter coefficients (H) are stored in the memory cells. The filter is described in the terms of the summation-convolution.

$$y(n) = \sum_{k=0}^{N-1} H(k)*u(n-k) \qquad (12)$$



where u(n) and y(n) are the filter input and output, respectively, the H(k)'s are the filter coefficients, and N is the order of the filter. All necessary adders and multipliers can be produced effectively by the MOSC compiler. The designer is only required to connect them to form a FIR filter. The RNS converter and the RNS decoder mentioned in Section B are also needed to perform the Binary/RNS number conversion.

CHAPTER FOUR

Results and Discussion

As was stated in the Introduction, the objectives of this thesis were to create a fully functional Unix based VLSI design workstation for full-custom hierarchical VLSI design, and to develop a specialized silicon compiler to synthesis memory oriented cells for constructing RNS arithmetic operators in a rapid and efficient manner. The full-custom hierarchical design methodology has been identified and illustrated with the design example of the 4-bit ripple carry adder in Chapter Two. The VLSI design workstation is primarily based on the **Electric** design system, the MOSC silicon compiler, and a number of utility programs. If one were to examine the usefulness of the VLSI design workstation, the criteria would be stated as follows:

1) *Time*: time taken to complete a design.

2) *Ease of operation*: easy to operate the CAD tools.

and

3) *Flexibility*: system can be expanded and modified for various design environments.

Since the workstation can be used in the full-custom hierarchical VLSI design, and can also offer the capability of silicon compilation, we would look at the three criteria for the workstation used in the full-custom design and for the MOSC silicon compiler separately.

For the full-custom design, circuits in the workstation are represented as networks that contain nodes and connecting arcs. The

nodes are electrical components such as transistors, logic gates, and electrical contacts. The arcs are simply wires that connect the nodes to create the electrical connectivity. In addition, each node has a set of ports which are the sites of arc connection. The geometrical information of a circuit mask layout is stored in the actual coordinates of nodes and arcs. Each node and arc contain a set of attributes, such as, mask layers and layer dimensions. Hence, the electrical and geometrical properties are obtained once the circuit mask layout has been completed. Therefore, no time is spent on a mask network extraction process.

Collections of nodes and arcs can also be aggregated into cells which can be used higher in the hierarchy to act as nodes. These user-defined nodes have ports that come from internal nodes whose ports are exported. With the identified VLSI design methodology, the top-down design method can be achieved. The capability of specifying layout in a top-down fashion encourages modularity in circuit design. In addition, any changes in any cells can be made without the need for redesign of the whole layout. Therefore, this top-down design method and bottom-up hierarchical update capability largely reduce the time taken in the mask layout process.

The design rule checker (DRC) works incrementally along with the layout editor system to provide immediate feedback to the designer's modifications. Hence, the designer can make immediate correction to the layout if an error is reported. In addition, Electric currently has seven simulator interfaces that generate netlists from the electrical connectivity of a mask layout or a circuit schematic. As mentioned in Chapter Two, the most interesting simulation

environment in the workstation is the SPICE simulation package. A complete SPICE deck input file can be automatically generated. In contrast to the Phoenix Data System package, the SPICE input file can be created in a single step. The SPICE output can also be displayed graphically on the workstation by invoking the PLOT program. The PLOT program allows users to select the window size and an unlimited number of curves to be displayed on the workstation. The Relax2 circuit simulator has also been modified to run successfully on the workstation. Since the DRC and various simulation interface tools are integrated with the circuit layout system, the design time can further be reduced.

The workstation provides a powerful and flexible user interface. Commands can be issued as single keystroke, single click on the mouse, menu selections, or full commands typed in with their parameters. The first three forms are shorthand for the last form, and dynamic binding can attach any full command to a single key or mouse button. Therefore, the whole keyboard and mouse can be reprogrammed for various commands in different applications. Powerful macro facility is also provided for faster design operations. A user can create his own parameterized macro command for his custom design operation , and access the database object variables to obtain information such as a node's attributes. These variables can be manipulated both arithmetically and conditionally to speed up the mask layout process. Multiple windows, a powerful "undo" capability and useful help facility further simplify the design task. Combined with all user interface facilities, the user interface can be tailored to

resemble any system that is familiar to or comfortable for the user. Hence, the workstation is very easy to operate.

All Electric's program files have been organized in such a way that it can be modified and upgraded effectively by making use of a Unix command, **make** For instance, if an I/O interface subprogram has been modified, the **make** command will compile the modified file only and reflect the changes to the rest of the design system. Hence, there is no need to compile other subprograms. In addition, the workstation contains a number of different design environments which are specified by different technology files. Currently it supports nMOS design, CMOS design, bipolar design, schematic capture, PCB design and general-purpose artwork environments. Users can change the design environment by issuing a single command. Since the workstation is capable of using different technologies in a single design, flexible layout schemes become possible. For example, a CMOS circuit can be treated as a component in a printed circuit board (PCB) design, enabling proper planning of chip environment details such as pin requirements.

Users can also create new design environments. Adding environments simply requires that the necessary tables and routines describing the new environment be coded into a "C" program module, and a technology table in the main program must be updated. The routines are for initialization, special control, node description, arc description and port description. Tabular information includes design rules, simulation characteristics such as the CMC's SPICE data, default size of various components, connectivity information, and layer colors. Much of the information handled by the routines is also

coded tabularly. Therefore, new technology file is relatively easy to create so that users are not restricted to a limited number of design environments.

One of the flexible features of the design system is X-Window's networking capability, that provides a multiple-terminal working environment such that a user is able to have maximum layout area on the screen. It was achieved by specifying a desired graphical output terminal in the graphic interface module. If a user invokes the design system in a DEC VT241 terminal, he can use the VT241 terminal for textual I/O information, and use the entire graphical monitor for the graphical editing purpose. Therefore, users are not limited to one working terminal only.

The most flexible aspect of the design system is the use of graphical constraints as a programming language. The attributes of an arc provide a limited form of control over mask layout. For example, many designers need arcs at 45 degrees angles. Rather than provide 45 degree angles as a possible orthogonal orientation, the constraint could be changed to one of angle rigidity in which an arc is forced to remain at its current orientation. Another useful constraint would provide a flexibility limit that allows stretching within a specified range. Beyond that range, the arc becomes rigid. Once the arc connecting a number of nodes has become rigid, the user can treat the connected nodes as a single rigid component.

Another feature of the programming language is the conditional expression. Flexibility-limited wires provide one aspect of this feature. Attachment of conditional expressions to components and their characteristics could also be useful. Looping control would

be helpful in the graphical layout system. This could be done by extending the use of "array" in the mask layout process. It currently supports indexed arrays of components. If a desired component is selected, the user can create one dimensional or two dimensional linear arrays of the component as illustrated in design examples in Chapter Two. Combined with a variable assignment feature, these arrays could be viewed as a form of loop construct. For example, the width of a wire could be used as a loop parameter to determine the number of contacts to use in connecting the wire.

In addition to all useful features of the workstation, a 4-by-4 shift register, a 4-bit transmission gate adder and a 4-bit ripple carry adder have been designed and simulated successfully in less than 10 days by an un-experienced mask layout designer.

Based on the selected criteria and the circuit design examples, the VLSI design workstation has proved to be a functional, easy-to-operate, and flexible system for full-custom hierarchical VLSI design. However, it is not perfect. Few limitations have been experienced. The most serious limitation is relatively inaccurate SPICE deck extraction which generates correct netlist information but incorrect parasitic capacitance or resistance values. For 3-micron CMOS design, the 0.6 factor is not incorporated in the mask extraction. In addition, the "C" version of SPICE 3A6 is currently not available. Hence, the SPICE simulation must be performed by using the GSPICE program in the VAX-11/785 or 750 computers.

Another limitation is the mask layout I/O format. The Caltech Intermediate Format (CIF) is the only I/O format supported by the system. Since the CIF code does not provide any information of

connectivity, all electrical properties are lost when the internal representation of mask layout is converted into CIF code. If a user would like to read a CIF file into the internal database of the system, he must turn off the DRC analyzer first and use the "universal wire" to connect the appropriate polygons to build the connectivity. In most cases, it is impossible to obtain the correct connectivity information from a CIF file. Fortunately, most of these limitations can be removed by using the Phoenix Data System package available in the VLSI Research Laboratory.

The MOSC silicon compiler was primarily developed for producing RNS arithmetic operators. The main goal of using a silicon compiler is to obtain mask level description of a desired circuit in a rapid manner. The design time and ease-of-use are hence the appropriate criteria in judging the usefulness of the MOSC silicon compiler. As illustrated in Chapter Three, the MOSC compiler based on the fully pipelined memory oriented cells, and the quarter square multiplication methd. This compiler can produce 5-bit modular RNS adders, subtractors, multipliers and constant operators in a simple and efficient manner. It also incorporates the two extra bits in the memory cell for fault detection purpose. MOSC's users do not need to create a *program-like* input file in order to obtain various modulo-number arithmetic operators since the user-interface in the MOSC is *question-driven*. All memory structure cells are placed as close as permissible by the design rules in order to achieve the minimum size. In addition, the number of memory structure cell is kept as low as possible. For example, it employs the quarter square multiplication method to construct modular multiplier. Therefore, it

can create the mask level representation of a 5-bit modular RNS multiplier in seventeen memory cells and in less than one minute, and it requires only two user-input information such as operator's type and modulo-number. It was hence proved to be able to reduce the design time dramatically and to be easy to use.

Since the MOSC is a highly application specific CAD tool, it is not flexible in producing general circuits for various applications. However, since the MOSC silicon compiler was developed in a modular fashion as mentioned in the section of software development environment in Chapter Three, expansion can easily be done to incorporate FIR filter design capability and to create RNS-to-Binary decoders based on the CRT method.

To summarize the capabilities of the VLSI design workstation, a diagram presenting various CAD tools and the workstation's feature is shown in Figure 4.1. The workstation represents a dynamic VLSI design facility which can continually grow to simplify the increasing complexity of the VLSI circuit design.

With the Unix based VLSI design workstation, the VLSI design facilities in the laboratory has been expanded as shown in Figure 4.2. The VLSI design workstation can be a standalone system, and can also be integrated with the Phoenix Data System package and the Daisy workstation. The SPICE deck input file generated by the workstation can be read directly by the GSPICE simulation package.

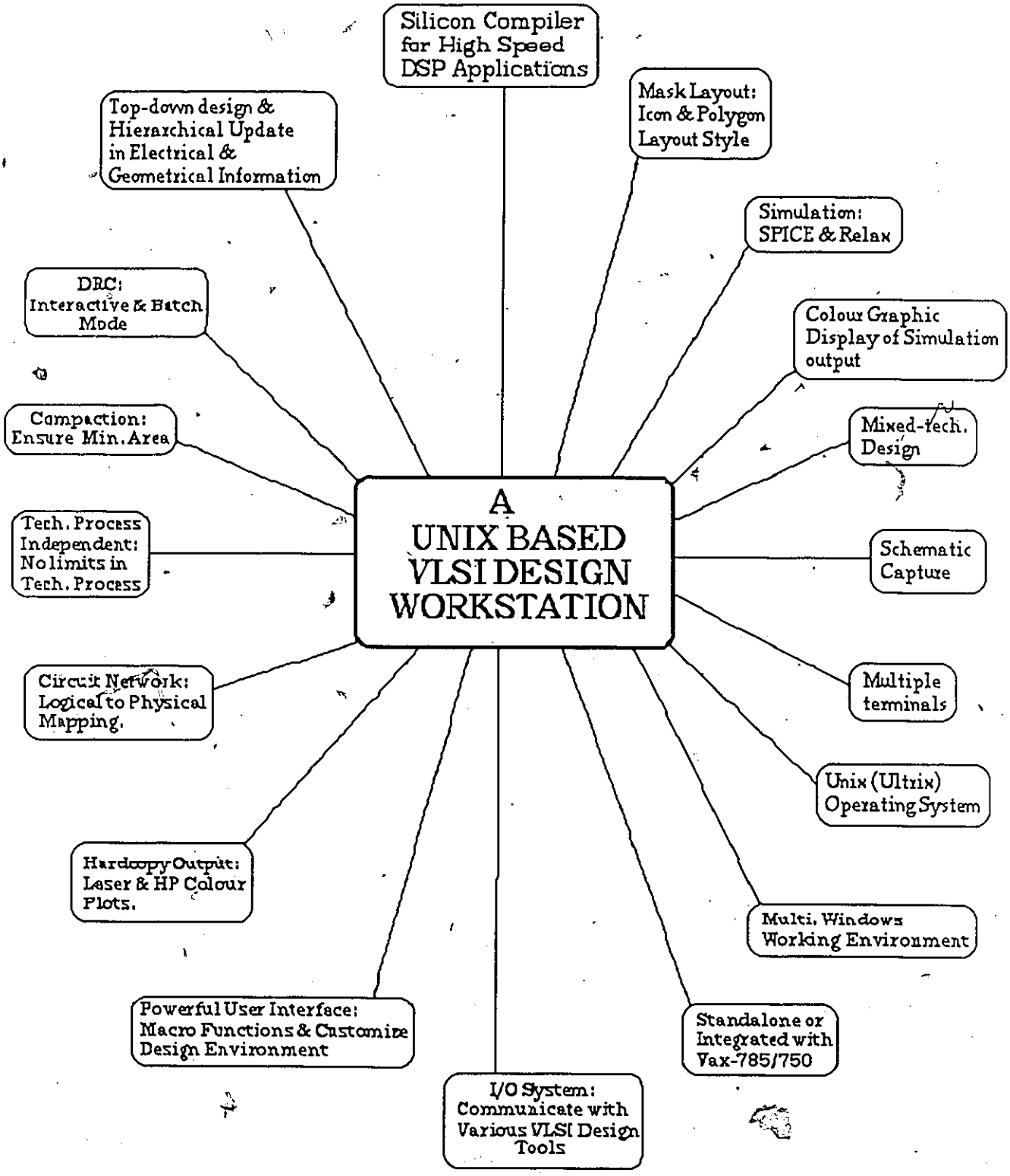


Figure 4.1: Capabilities of the Unix based VLSI design workstation

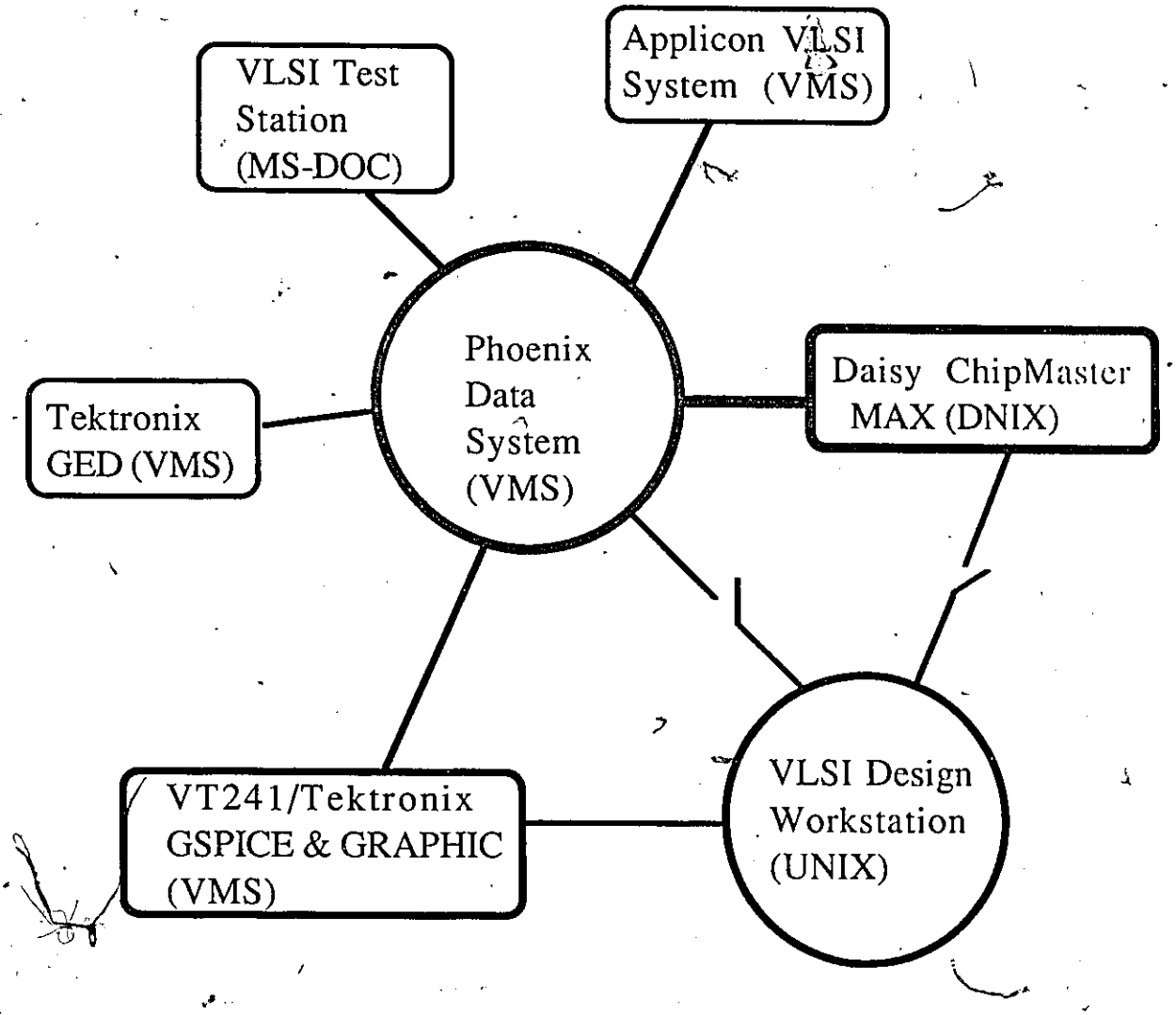


Figure 4.2: VLSI design facilities with the Unix based VLSI design workstation

CHAPTER FIVE

Conclusion

A Unix based VLSI design workstation has been successfully developed on a DEC VAXstation II/GPX minicomputer. The VLSI design workstation is developed around the Ultrix (Unix) operating system, the X-Window facility, the MOSC silicon compiler, a modified Electric design system, and a number of utility programs. The capabilities of the workstation are best summarized and illustrated in Figure 4.1 on page 78, which also shows what CAD tools are currently available on the workstation.

The most powerful capability of the VLSI design workstation is provided by the development of the Memory Oriented Silicon Compiler (MOSC). The MOSC compiler utilizes the concepts of the Residue Number System to create very high-speed digital circuit components. The MOSC compiler employs two fully pipelined memory structure cells, the quarter square multiplication method, and a fault detection mechanism. Based on these techniques the MOSC compiler has successfully synthesized 5-bit RNS adders, subtractors, multipliers and constant operators. The existence of these RNS arithmetic operators allows one to rapidly design RNS architectures for digital signal processing applications, such as, FIR filters. One of the significant advantages of the MOSC compiler is that it allows inexperienced designers to create RNS based VLSI circuits in a short time period. The nature of the mask layouts generated by the

MOSC compiler compares favourably with those created manually by an experienced designer. In addition, unlike the other silicon compilers mentioned in Chapter Three, the user-interface developed for the MOSC compiler is *question-driven* so that users do not have to create a *program-like* input file. Hence, the MOSC compiler is a most user-friendly CAD tool.

Once the MOSC compiler has acquired sufficient circuit information from the user, it will create the appropriate memory content and then map the content onto the basic memory oriented cell to form a complete RNS based cell. A number of these RNS cells with different memory contents are placed together to form a linear array realization of the desired RNS arithmetic operator. A hierarchical CIF file representing the desired circuit is then generated. The MOSC compiler can also display the generated mask layout on the screen by controlling the Electric design system. A user can also instruct the MOSC compiler to generate the mask representation of the memory content only. With the ECIFIN/DCIFIN translators, the generated CIF file can also be used by other VLSI design tools available in the VLSI Research Laboratory.

The MOSC compiler was developed in such a way that it allows for the incorporation of more capabilities. For example, if an auto-router is developed and integrated with the MOSC compiler a wider range of applications, such as, FIR filter design could be handled directly.

Unlike FIRST, LAGER and BSSC which are complete IC silicon compilers, the MOSC compiler generates only major RNS arithmetic components. Therefore, a user must manually create the mask layout

of the connections between the major components and the I/O pads in order to complete the design.

The Electric design system has been modified and upgraded so that it can run in a X-Window environment. A number of special utilities including the plotting programs for SPICE and Relax2 simulation outputs, drivers for hardcopy devices, the ECIFIN/DCIFIN translator, and macro functions have also been developed. A VLSI design methodology supported by the workstation has been formulated to simplify the complexity of the VLSI circuit design process. All the VLSI CAD tools implemented on the workstation have been described along with their use in the design methodology. A number of circuit designs, including a four-by-four shift register and two 4-bit combinational adders, have been executed on the workstation to verify its capabilities.

The workstation supports a top-down design methodology and a hierarchical bottom-up update capability. It is possible to develop mask layouts by interacting at the transistor level as opposed to the basic mask level. Both geometrical and electrical information associated with the mask layout are available once the mask layout has been completed. In addition, all simulator interfaces and design-rule checks are integrated with the layout editor system. Hence, VLSI circuit design time can be significantly reduced.

Since the user interface designed for the workstation is very flexible, it can be re-defined for different design applications. A powerful and parameterized macro facility is also provided to allow a user to customize his or her working environment. The combination of multiple windows, a multiple terminal display capability and an

informative *help* facility provides for a workstation that is highly user-friendly.

The workstation is quite flexible in that users can easily add a new design environment by creating a technology file in the "C" programming language. In other words, the design system implemented on the workstation is technology independent. The workstation also supports the use of different technologies in a single design. The constraint system in the layout editor further enhances flexibility by restricting wire properties so that a user can manipulate a number of nodes and wires as a single object.

Future development to make the workstation more powerful might include the acquistion of more simulation tools, such as, RSIM, MARS and SPICE3. A versatile data translator might also be developed to convert the design information from Electric's database to the GDS II format or the EDIF format in order to occupy less memory space and maintain some useful information, such as, the electrical connectivity.

Finally, a last suggestion for further work is to develop a graphic interface driver for the modified Electric design system to support the Tektronix graphic terminals available in the VLSI Research Laboratary, since these terminals offer higher resolution and local memory for faster graphic operations, the performance of the VLSI design system driving a Tektronix terminal will definitely increase.

References

[BaCo82] F. Barsi, A. Cola, "A VLSI Binary Multiplier using Residue Number System," IEEE ICCC82, New York, pp.583-589, 1982.

[BaJM87] M. Bayoumi, G. Jullien, W. Miller, "A Look-Up Table VLSI Design Methodology for RNS Structures Used in DSP Applications," IEEE Trans. on Circuits and Systems, vol.CAS-34, no.3, pp.604-615, June 1987.

[BaJM87a] M. Bayoumi, G. Jullien, W. Miller, "A VLSI Implementation of Residue Adders," IEEE Trans. on Circuits and Systems, vol.CAS-34, no.3, pp.284-288, March 1987.

[Bird87] P.D. Bird, "The Application of Multi-Valued Logic to the Implementation of Residue Number System Hardware," Master Degree Thesis, Dept. of Electrical Engineering, University of Windsor, 1987.

[Brya81] R. Bryant, "MOSSIM: A Switch-level Simulator for MOS LSI," Proceedings of the 18th Design Automation Conference, July 1871, pp.786-790.

[DeMR84] P. Denyer, A. Murray, D. Renshaw, "FIRST-Prospect and Retrospect," VLSI Signal Processing, IEEE Press, pp.252-263, 1984.

[Deny82] P. Denyer, "An Introduction to bit-serial architectures or signal processing," in VLSI Architecture, Prentice-Hall, 1982.

[Garn59] H. Garner,"The Residue Number System", IRE Transactions, June 1959, pp.140-147.

[JeLe77] W Jenkins, B. Leon, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters," IEEE Trans. on Circuits and Systems, vol. CAS-24, no.4, pp191-201, April 1977.

[LeRu82] E. Lelarasmee, A. Ruehli, "The Waveform Relaxation Method for Time-domain Analysis of large Scale Integrated Circuits,", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. CAD-1, No.3, July 1982, pp.131-145.

[Nage75] L. Nagel,"SPICE2: a Computer Program to Simulate Semiconductor Circuits," Memo ERL-M520, University of California, Berkeley, 1975.

[Raja87] P.V. Raja, Master Degree Thesis to be published, Dept. of Electrical Engineering, University of Windsor, 1987.

[RaPB85] J.Rabaey, S.Pope, R.Brodersen, "An Integrated automated layout generation system for DSP circuits", IEEE Trans. Computer-Aided Design, vol.CAD-4,no.3, pp.285-296, July 1985.

[RaTa86] A. Rammnarayanan, F. Taylor, "RNS Cellular Arrays," IEEE Trans. on Circuits and Systems, vol.CAS-3, no. 5, pp.526-532, May 1986.

[Tayl82] F.J. Taylor, "A VLSI Residue Arithmetic Multiplier," IEEE Trans. on Computers, vol.C-31, no.6, pp.540-546, June 1982.

[TaRa81] F. Taylor, A. Rammnarayanan, "An Efficient Residue-to-Decimal Converter," IEEE Trans. on Circuits and Systems, vol.CAS-28, no. 12, pp.1164-1169, Dec. 1981.

[TaJM87] M.Taheri, G.A.Jullien, W.C.Miller, "Systolic ROM Arrays for Implementing RNS FIR Filters", Proceedings International Conference on Acoutics, Speech and Signal Processing", 1987, pp.771-774.

[TaJM87a] M.Taheri, G.A.Jullien, W.C.Miller, "Fault detection in RNS Systolic Arrays", Electronic Letters, Vol.23, no.4, pp.165-166. Feb. 1987.

[WeEs85] N.Weste, K.Eshraghian, "Principles of CMOS VLSI Design", Addison Wesley, Don Mills, 1985.

[YJHN87] F. Yassa, J.Jasica, R.Hartley, S.Noujaim, "A Silicon Compiler for Digital Signal Processing: Methodology, Implementation, and Applications", Proceedings of the IEEE, vol. 75, No.9, pp.1272-1281, Sept. 1987.

Bibliography

N.Weste, K.Eshraghian, "Principles of CMOS VLSI Design," Addison Wesley, Don Mills, 1985.

L.A.Glasser, D.W.Dobberpuhl, "Design and Analysis of VLSI Circuits," Addison Wesley, Don Mills, 1985P. Cappello, "VLSI Signal Processing," IEEE Press, New York, 1984.

D.A. Hodges, "Analysis and Design of Digital Integrated Circuits," McGraw-Hill, 1983.

S.M. Rubin, "Computer Aids for VLSI Design," Addison-Wesley, 1987.

P. Cappello, C. Wu, "Computer-Aided Design of VLSI FIR Filters," Proceedings of the IEEE, vol.75, no.9, pp.1260-1271, Sept 1987.

D. Gajski, "The Structure of a Silicon Compiler," IEEE ICCC82, New York, pp.272-276, 1982.[HSCW82]

T. Hedges, K. Slater, G. Clow, T. Whitney, "The Siclops Silicon compiler," IEEE ICCC82, New York, pp.277-280, 1982.

Appendix I

Ultrix, Windowing, and Communication Facilities

UNIX Tutorial

Unix is a general-purpose, multi-user, and interactive operating system for most popular machines such as DEC machines, SUN workstations, Apollo machines and etc. **Ultrix**, Digital's Unix operating system, provides compatibility with AT&T's System V while maintaining all Berkeley 4.2 BSD commands, system calls, library functions. The Ultrix operating system is currently installed on the VAXstation II/GPX workstation in the CAD/CAM Centre.

This tutorial is written primarily for Unix (Ultrix) beginners. This tutorial will describe the most handy commands, standard I/O redirection and pipeline process. A Mail facility will also be described in detail in the Mail tutorial.

Before we learn anything about the Unix operating system, we should realize that Unix distinguishes between *upper-* and *lower-*case characters; so 'A' and 'a' are not the same.

First of all, let us login the system. You must have a valid login name. When you get a *login:* message, type your login name in lower case, follow it by a RETURN.

login: username<RETURN>

If a password is required, you will be asked for it, and printing will be turned off while you type it. Don't forget the RETURN.

When you see a prompt character **%**, you are in shell command mode. Before we do anything, I would like to show you how to logoff the system. To end a working session, type

logout or **ctrl-d**

I. Basic Commands

When you are in the shell command mode that means you have the % prompt, you can issue Unix commands. Try typing

% date

you will get something like:

MON SEP 7 14:17:10 EST 1987

Another command you might type is

% who

joe sam alger root

which will tell you everyone who is currently logged in. However, if you type

% whom

you will be told

whom: Command not found

If you know you have make a small mistake, use <DELETE> key to correct it.

One of the most useful commands is **ls** (list) which will tell you what file(s) you have in your current directory.

% ls

a.out demo demo.c

If you type something like

% ls -l

you will get something like

-rwxr----- 1 joe 20 Jul 22 2:56 a.out

-rwxr----- 1 joe 120 Jul 20 3:46 demo
-rw-r----- 1 joe 201 Jul 18 8:40 demo.c

You will be told more about your files in the directory. The date and time are of the change to the file. The 30, 120 and 201 are the number of characters in each file. And, joe is the owner of the file.

Another useful command is **cat**. The **cat** displays the content of a specified file(s) on the screen.

% **cat** *junk temp*

displays two files, *junk* and *temp*, on the screen.

Another similar command is **more**.

% **more** *junk*

The **more** command will display each full-page of the file *junk* and stop until you hit SPACE BAR or RETURN. This stop-and-go process continues until the whole file is completely displayed.

While you are in the middle of a process and you want to stop the process, You can always send an interrupt signal by typing **ctrl-c** on the keyboard. In most cases, it will work and return you with the % prompt.

From now on I would like to show you more Ultrix commands in a simple and straight forward manner.

*Rename a file. Type

% **mv** old-filename new-filename

*Copy a file. Type

% **cp** old-filename new-filename

*Remove or delete a file. Type

% **mv** filename

*Make or create a new directory. Type

% **mkdir** directory-name

*Change your current directory. Type

% **cd** new-directory-name

*Go back to previous directory. Type

% **cd ..**

*Go to your login directory. Type

% **cd**

*Remove or delete a directory. Type

% **rm** directory-name/*
% **rmdir** directory-name

*Report current directory. Type

% **pwd**

*Show file type. Type

% **file** filename

There are many more useful commands, To describe all of them is out of the scope of this tutorial. However, with the commands mentioned above you should be able to get around in the Ultrix operating system.

II. Standard I/O Redirection

Most of the commands we have seen so far produce output on the terminal screen. However, you can replace the terminal screen by a file for either input or output, or both.

For example,

% ls > *filelist*

A list of your files will be placed in the file called *filelist*.

Also,

% **cat** *f1 f2 f3* > *temp*

The files *f1*, *f2* and *f3* are copied into a file called *temp* with the above command line.

In addition,

% **cat** *f4 f5 f6* >> *temp*

The symbol >> operates very much like > does, except that it means " add to the end of ." so that files *f4*, *f5*, *f6* are added to the end of the file *temp*.

In a similar way, the symbol < means to take the input for a program from the following file, instead of from the terminal. The form is

% command < data

For example,

% **vi** *file* < *script*

You can also redirect both input and output in one command line. For example,

% **vi** *file* < *script* > *letter*

III. Pipes

One of the advantages of using the Unix operating system is the idea of **pipe**. A pipe is simply a way to connect the output of one program to the input of another program, so the two run as a sequence of processes - *a pipeline.*

For example,

% **pr** *f1 f2 f3*

will print the files *f1 f2* and *f3*, beginning each on a new page. Suppose you want them run together instead. You could say

```
% cat f1 f2 f3 > temp
% pr < temp
% rm temp
```

but it takes a lot of work to finish a single task. However, we can use pipe structure. Type

```
% cat f1 f2 f3 | pr
```

The vertical bar | means to take the output from **cat**, which would normally have gone to the terminal, and put it into **pr** to be neatly formatted. For instance, if you want to count how many people are logged on, you can use

```
% who | wc -l
```

If you want to count how many files you have in your current directory, you can type

```
% ls | wc -w
```

From the power of the pipeline, you can build your own power tools from the simpler commands.

IV. The Shell

So far, we have seen the capabilities of <, > and |. These commands or symbols are actually a subset of commands inside the shell (**Sh**). The shell is an intelligent program that interprets what you type as command and arguments, It also looks after translating * and ? into lists of filename. The shell has other capabilities too, you can run two programs with one command line by separating the commands with a semicolon (;). Thus,

% **date; who**

does both commands before returning with a prompt character. In addition, you can also have more than one program running simultaneously if you wish. For example, if you are doing something time-consuming, and you don't want to wait around for the results before starting something else, you can say

% command < input-file > output-file &

The ampersand at the end of the command line say " start this command running, then take further commands from the terminal immediately." For instance, if you want to run SPICE porgram as a background or batch job, you can type

% **spice** < data.in > data.out &

The last command structure I would like to show you is *command file*. If you want to execute a command file, you can type

% **sh** < *command-file*

This says to run the shell with the file *command-file* as input. The effect is as if you have typed the commands on the terminal.

The Ultrix operating system also provides many software tools for document preparation such as **vi**, **ex** and **nroff**, and for programming. C, Fortran-77 and Pascal compilers are also available. For detail information of Ultrix commands, Ultrix user's manual should be referenced.

Mail and DECnet-Ultrix Utilities

Mail facility provides a simple and user-friendly environment for sending and receiving mail messages and files. It divides incoming mails into its constituent messages and allows the user to deal with them in any order. In addition, it provides a set of ed-like commands for manipulating messages and sending mail. Mail offers the user simple editing capabilities to ease the composition of outgoing messages, as well as providing the abilities to define and send to names which address groups of users.

I. Common Usage on Mail Messages

The **mail** command has two distinct usages: *send* and *receive* mail. Sending mail is simple: to send a message to a user whose login name is, say *joe*, use the shell command: **mail**

```
% mail joe
Subject: mail title
   Your message is typed here.
.....
.....
EOT
```

When you reach the end of the message, type **ctrl-d** (end of file or end of transmission) at the beginning of a line, which will cause mail to echo *EOT* and return you to the shell.

If you want to send the same message to several other people, you can list their login names on the command line. Thus

```
% mail sam joe susan
Subject: mail title
```

Your message goes here.

.....

.....

EOT

will send the message to *sam, joe* and *susan.*

Receiving a mail is also simple: just type

% **mail**

The **mail** will respond by typing its version number and date, and then listing the message you have waiting. Each message has an corresponding message number which will allow a user to handle his mail efficiently. To see the message, type

t *message-number*

Then the message will be displayed on the screen. If you want to see next message, type

n

After you have read the message and you may want to delete it, you should type

d *message-number*

Finally, you want to exit the mail program, type

q

or type

x

then you will be returned to the shell command.

II. Common Usage on Mail File

Sending a file to someone is easier. If you want to send *joe* a file called *workfile*, you should enter

% **mail** *joe* < *workfile*

Similarly, if you want to send a file to someone at different machine such as **cadcam**, you say

% **mail** *cadcam::joe < workfile*

Moreover, if you want to send a file to a group of users, type

% **mail** *joe sam susan < workfile*

In addition, you can also send both mail message and file to someone with the mail program. you say

% **mail**
To: username
Subject: title
type your message here
.....
.....
~**r** *filename*
EOT

then a file called *filename* with your message will be sent to a user called *username.*

It would be very useful if we can save the message into a permanent file for future reference. The mail program also offers this capability. Inside the mail program you can type

s *message-number filename*

Again you can save several messages into a file by typing

s *message-number1 messages-number2 filename*

Now you have seen the basic capabilities of the mail program which enables you to communicate with other users in the DECnet network. There are many other useful commands in mail program. All the commands mentioned above will help you to manipulate the mail messages and mail files efficiently.

There are *five* more **DECnet-ULTRIX** utilities that allow you to work with files across *heterogeneous* DECnet operating systems. Basically, The utilities allow you to:

1. *View remote DECnet directories*
2. *Display remote DECnet files*
3. *Transfer DECnet files*
4. *Delete remote Decnet files*
5. *Remote login*

When working with files across heterogeneous DECnet operating systems, you should enclose in single quotes (' ') any information following the node name in a non-Ultrix file specification. This prevents the local shell from reading and interpreting the non-Ultrix file specifications. Examples are used to illustrate the use of each utility.

Viewing a Remote Directory

The **dls** utility allows you to display the contents of a remote DECnet directory. The following command lists all of the files located in directory *[users]* on disk *sys$user:* on the *cadcam* node.

% **dls** *cadcam::'sys$user:[users]'*

dls displays the output on your terminal screen by default. You can direct the output to a file by using I/O redirection method.

Displaying a Remote File

The **dcat** utility displays a remote DECnet file. This command can also concatenate the contents of more than one remote DECnet file and direct the output to your terminal screen by default. You can direct the output to a separate file if you want to copy the file.

% **dcat** *cadcam::'sys$user:[users]demo.dat'*

This command displays the contents of file *demo.dat* located in directory *[users]* on disk *sys$user:* on the remote node *cadcam.*

Copying a File

The **dcp** utility allows you to copy files to and from remote DECnet nodes.

The following command transfers the *demo.dat* from a Ultrix operating system to the file *vms.dat* on the remote DECnet-VAX node *cadcam.*

% **dcp** *demo.dat cadcam/users/password::'dua1:[users]vms.dat'*

This command transfers a file *vms.dat* from VMS operation system to a file on a remote DECnet-Ultrix node.

% **dcp** *cadcam/users/password::'dua1:[users]vms.dat'* *

Deleting a Remote File

The **drm** utility removes a file from a remote DECnet node. For example, the following command removes the file *paints.3* from remote node *cadcam.*

% **drm** *cadcam/username/password::'sys$user:[users]paints.3'*

Logging on to a Remote DECnet Node

The **dlogin** utility allows you to log on to a remote DECnet node and to communicate with all programs running under that system. Once connected to a host, you must specify the commands for that system. To login on a remote system called *cadcam*, type

```
% dlogin cadcam
Username: users
Password:
.....
```

To log out of a remote session, enter the **logout** command for the remote operating system.

The **dlogin** utility also allows you to execute commands on your local Ultrix operation system while you are engaged in a remote session session. To temporarily return control to your local Ultrix node, press

% ~ local command>

Once you receive the *local command>* prompt, you can execute commands on your local node. When you wish to resume your **dlogin** session, press **RETURN** at the *local command>* prompt, and your remote session prompt will return.

The DECnet-Ultrix utility commands as described above are sufficient for most users to manipulate remote files across heterogeneous operating systems. For further information about the commands, refer to DECnet-Ultrix User's and Programming's Guide.

ULTRIX WINDOW MANAGER

A Simple Tutorial

Ultrix Window Manager (**uwm**) is a specialized application that provides a flexible interface to the multitasking capabilities of the Ultrix operating system. Standard capabilities include resizing, restacking, moving, and iconifying windows and pop-up menus for creating new windows and changing colour characteristics. Users have complete control over the windowing environment. A startup file can be edited easily to change such characteristics as background and window colours. Command functions can be assigned directly to mouse buttons and custom menus can also be created to invoke shell programs.

This tutorial will show you the basic capabilities of the window manager, and how to activate and use the window manager effectively.

I. Running the Window Manager

To start the window manager, enter the **uwm** command in the primary login window, as follows:

uwm &

Always start the **uwm** command as a background process (shown by the ampersand). When the window manager has been successfully initialized, the keyboard beeps once. You can invoke the window manager automatically every time you log in by adding the **uwm** command and its options to your *.login* file. If you do not add this

uwm command to your Ultrix .login file, you must enter the command again every time you log in.

II. Logging Out

When you log out through the primary window, you terminate your connection with the window server, and any windows you created during your session instantly disappear.

To log out of windows individually, go into each window and enter **ctrl-d** on the beginning of a line. If a window is in an icon state, remap it a window, place the cursor inside the window, and then log out using **ctrl-d**. After logging out of individual windows, log out of the primary window to disconnect from the window server.

III. Using the Window Manager

The **uwm** command activates the default window manager. This default manager provides an immediate interface to the window server by invoking a subset of predefined window manager functions. An example of the default window manager is stored in the file, */usr/lib/uwm/default.uwmrc*. Once you become familiar with the default functions, you can customize it for your own unique need.

A. Creating Windows

The Window server lets you simultaneously interact with multiple processes by allowing you to create multiple display windows on the screen.

To generate a window, either select Create Window from the Extended Window Operations Menu which will be described later in section B or enter the **xterm** command in the primary login window:

xterm &

When you enter the **xterm** command, a small window appears at the mouse cursor position. The upper left corner of the screen displays the window size in columns by rows. To move the mouse cursor to a screen location where you want to begin sizing the window. Use the mouse to size the window, as follows:

* Press the left mouse button to create an 80-column by 24-row window.
* Press the right mouse button to create an 80-column by 65-row window.
* Press the centre mouse button to create a window that is the height and width you choose. Move the mouse horizontally and vertically until the size indicator displays the desired height and width in columns and rows. Release the mouse button and the window appears on the screen.

If the new window covers the primary window, use the Window Operation Menu defined later to resize the window, convert it to an icon, or move it. To display multiple applications, just create more windows. Once the window is created, a virtual terminal line is allocated and anything that would normally display on the screen displays in the window.

B. Selecting Menu Functions

The default environment provides two menus to allow you to use the window manager functions. These menus are:

* **WINDOW OPS (Window Operations)**
* **EXTENDED WINDOW OPS (Extended Window Operations)**

After the window manager is running in background as mentioned above, you can display these menus with the CTRL key and mouse buttons. Use the CTRL key and right mouse button to display the Window Operations Menu; use the CTRL key and right mouse button to display the Extended Window Operation Menu.

The Window Operation Menu, shown in Figure 1, lets you manipulate the windows that you generate on the display screen. Using this menu, you can convert windows to icons or icons to windows, and you can move, resize, lower, and raise windows.

WINDOW OPS
(De)Iconify
Move
Resize
Lower
Raise

Figure 1: Window Operation Menu

**** (De)Iconify**

This selection either converts a window to an icon or converts an icon to a window. To convert a window to an icon as follows:

1. Move the mouse so the menu cursor points to (De)Iconify.
2. While holding down the CTRL key, release the mouse button.
3. Move the mouse cursor into the window.
4. Press the right mouse button.

When you release the CTRL key and the right button, the window disappears and its corresponding icon appears in a default location. To convert an icon to its corresponding window is the same except picking the icon instead of picking the window.

**** Move**

This option lets you move an existing window or icon to other screen location. To move a window or icon as follows:

1. Move the mouse so the cursor points to Move.
2. While holding down the CTRL key, release the button.
3. Move the cursor into the window or icon you want to move and press the right button again.
4. Move the cursor to a new location and release the CTRL key and right button.

** **Resize**

This option lets you resize an existing window. When resizing, the location of the mouse cursor within the window determines its direction of expansion. to resize a window as follows:

1. Move the cursor to Resize.
2. While holding down the CTRL key, release the button.
3. Move the cursor into the window you want to resize and press the right button again.
4. While holding down both key and button, move the cursor. When the window has the desired size, release the CTRL key and right button.

** **Lower**

If an upper window obstructs the window below it, you can move the selected window to the bottom with the Lower menu selection. To lower a window as follows:

1. Move the mouse so the cursor points to Lower.
2. While holding down the CTRL key, release the button.
3. Move the cursor into the window to be lowered and press the right button again.
4. Release the CTRL key and right button, and the window drops

to the bottom.

** **Raise**

This selection raises a selected window that is obstructed by a window above it. It is the converse of the lower function. To raise a window, move the mouse so the menu cursor points to raise, and then use the same procedure described for Lower.

The Extended Window Operations Menu, shown in Figure 2, lets you generate terminal emulator (**xterm**) windows and select other window manager function.

EXTENDED WINDOW OPS
Create Window
Iconify at New Position
Focus Keyboard on Window
Freeze All Windows
Unfreeze All Windows
Circulate Windows Up
Circulate Windows Down

Figure 2; Extended Window Operation Menu

** **Create Window**

This option allows you to create a window by invoking the xterm command. Hence, the procedure is the same as described in section A.

** **Iconify at New Position**

This option changes a window into an icon and sets a new default location for the icon. To select this option as follows:

1. Move the mouse so the cursor points to Iconify at New Location.
2. While holding down the CTRL key, release the center button.
3 Move the cursor into the window and press the center button.

4. Move the cursor to a new location and release the CTRL key and center button.

**** Focus Keyboard on Window**

This selection lets you direct all keyboard input to a single, selected window. To change the focus as follows:

1. Move the mouse so the cursor points to Focus Keyboard on Window.
2. While holding down the CTRL key, release the center button.
3. Movr the cursor into the window into which you want all keyboard input to be directed, or into the root window to convert focus back to all window, and press the center button.
4. Release the CTRL key and the center button.

**** Freeze all Windows**

This menu selection halts all window input and output. To freeze all windows as follows:

1. Move the mouse so the cursor points to Freeze All Windows.
2. Release the button, move the cursor out of the menu, and press the center button again.
3. Release the CTRL key and the center button and all window action stops.

**** Unfreeze All Windows**

1. Move the mouse so the cursor points to Unfreeze All Windows.
2. Release the button, move the cursor out of the menu, and press the center button again.
3. Release the CTRL key and the center button to update all windows.

**** Circulate Windows Up**

This option causes obstructed windows to rotate from back to front. To circulate windows as follows:

1. Move the mouse so the cursor points to Circulate Windows Up.
2. While holding down the CTRL key, release the center button.
3. Move the cursor into the window and press the center button.
4. Release the CTRL key and the center button and the windows circulate from the bottom the top.

**** Circulate Windows Down**

This selection causes windows to rotate from front to back. This function is the converse of Circulate Windows Up. To select this function, use the same procedure as described for Circulated Windows Up.

After you have been familiar with the default window environment, you can modify the default window manager file, */usr/lib/uwm/default.uwmrc*, to customize your needs A detail description of customizing the window manager can be found in Ultrix-32w manual.

Appendix II

Electric VLSI Design Environment

Appendix II
Electric VLSI Design Environment

This appendix describes how to modify and compile the Electric software package for running on X-Window environment. A general description on Electric will also be given. For implementing the Electric design system in the DEC VAXstation II/GPX workstation the biggest problem is to fully understand the complete Electric system, its design philosophy and software development method, and the hardware and software features of the GPX workstation must also be determined.

In the GPX workstation, three graphic drivers are available in the Ultrix environment, such as, QIL, X-Window and GKS. The QIL, QDSS Interface Library, has been designed specifically for DEC QDSS video system so that programs based on this driver would have higher graphic performance. However, it is hardware dependent, and it is very difficult and inflexible to program. Most importantly, the QIL does not handle system inputs as not monitoring the mouse and the keyboard. The X-Window, developed by the MIT, is a multiple windows and networking graphic system. Programs based on the X-Window are highly transportable in Unix workstation community. Most major graphic routines are supported, and it is relatively easy to program. The GKS, Graphic Kernel System, is an industrial standard for general graphic applications. Since it is hardware independent, the best graphic performance for a particular machine would not be

obtained. Therefore, the X-Window is selected as having higher potential to configure the Electric to operate in the GPX workstation.

The programming structure of Electric must be understood before we can modify the Electric's source code. Files in Electric are divided into a number of category, namely main core programs, database management programs, graphic-related programs, I/O programs, analysis aid programs, simulation interface programs, technology programs and user interface programs. Each category has a central program. For instance, there are twenty files in technology category. *tectable.c* is the corresponding central program. The rest are the various technology files such as PCB technology file, CMOS-3D technology file and etc. Therefore, adding new technology is simple: copy one of the technology description files, update the new technology, update the table in *tectable.c*, and compile and link the Electric package.

Electric design system consists of approximate 344 files which must be properly stored in a file structure for effective management. It was also found that a number of files must be stored in different directories. Hence, all "C" programs have to be scanned to determine what *include* files are needed, and where the files are supposed to be. Besides the *include* files, other special files such as *help* and *macro* files must also be placed at */users/local/electric/help* and */users/local/electric/lib* directories. Once the file structure has been created, the *config.h* file must be modified to reflect the file structure, and the main program will use the information stored in the *config.h* to find the required files.

Another problem associated with the implementation is that the "C" compiler in Ultrix environment is not 100% compatible with the one in Berkeley BSD4.2. Hence, the Electric's source code must be modified to correct the subtle differences. A noticeable compilation error is related to the REGISTER variable declaration. After a long and tedious process of correction all files have been compiled successfully.

Although no error occurs in the compilation, the Electric still cannot run on the GPX workstation. As the Electric was developed primarily for the Sun workstation and the AED frame-buffer terminal, an X-Window graphic interface driver must be created for the GPX workstation. A major problem found in this creation process is the severely limited documentation on the X-Window programming, and it may be due to the fact that X-Window system is still under development. The properties of most graphic routines supported by X-Window are learned experimentally. The current version 10.0 of X-Window system has severely limited capability in manipulating text. However, a GPX graphic interface program for Electric developed by TUNS was given with the aid of the CMC. Hence, all required files are ready for compilation. All files are compiled and linked successfully using a Unix command **make**. A *makefile* required by **make** must be created first to indicate the GPX graphic terminal as the default graphical output terminal. Finally, an operational Electric file is generated, and Electric runs successfully on the GPX workstation. Still a lot of modification must be done to fine-tune the Electric system, especially the CMOS technology file and the graphic interface program.

A number of areas have been modified in the graphic interface driver in order to obtain better graphic performance. For instance, the pattern of grid and the shape of the mouse cursor have been changed so that they can be easily recognized. Since the grid must be drawn and erased in an extremely fast speed without destroying other graphic objects on the screen, a fast grid drawing routine must be developed. For providing a maximum screen working area a multiple-terminal environment must be investigated and developed. Since X-Window system also supports networking applications, an attempt is made to have textual information displayed on a DEC VT241 terminal, and to have graphical information displayed on the GPX graphical monitor. This attempt is accomplished using **XOpenDisplay**(server) routine and specifying that the server is equal to GPX server: **unix:0**. The routine takes the name of the server, and opens a connection to the server for display hardware. Once the connection has been established an **XCreateWindow** routine is used to create various size of graphic displaying window.

Another limitation associated with several X-Window routines is unreliable operation in the GPX workstation. For instance, a patterned line routine may crash the server connection. Hence, all routines in the interface driver are developed without using those unreliable X-Window commands.

Since we are interested in CMOS design, the CMOS-3D technology file, *tecncmos3.c*, must be updated for the proper design. Each technology file describes both electrical and geometrical properties, and defines properties of layers, arcs, nodes, variables, and routines which store the information stated in arcs, nodes and

variables in the database. Nodes and arcs are the basic components in Electric system, and the nodes represent transistors and contacts, and arcs act like electrical wires.

Layers are composed of the real mask layers and pseudo layers. Layers are used to build nodes and arcs. The pseudo layers are used by Electric for special purposes such as pin nodes and transistor area. Only five layers may be transparent and the rest must be opaque. The five layers correspond the metal, polysilicon, diffusion, p+ and well layers. Each layer has a unique colour and they can combine to form up to 32 different colours. For instance, a transistor is built by combining these five layers, and a unique colour pattern is formed to represent this transistor. For the 8-plane GPX workstation five planes are used for the five layers, one for grid, one for cursor and one for background. Two tables describing the design rules must be created for the DRC purpose. One table specifies the distance between electrically connected layers and other one states the distance between un-connected layers. These tables are upper-diagonal arrays that have one row and column for every layer so that specfying distance between various layers is easy.

The arcs are based on the layers. The difference between these two is that an arc can be composed of many layers. For example, the CMOS diffusion-well arc consists of a layer of diffusion and a layer of well. Nodes include pins, transistors, contacts and pure mask layers. Each node has one or more ports which are used for connection purpose. For arc and node we can specify the geometrical configuration using the their ***style*** properties. For example, we can select **filled box, closed box** or **crossed box** for each layer. Their

colours are determined by the layer colours. After the layers, arcs, and nodes have been defined, all of this information must be tied to the database using the routines which are standard and can be copied from an existing technology file. A last section in the technology file is the variable which is primarily used for simulation purpose. For example, an updated SPICE simulation data has been stored in this section for generating proper SPICE input file. After all files have been modified, an operational Electric program is generated.

Now let us look at the general features and operations of the Electric design system. Electric is a fully interactive electrical design aid that features multiple design technologies, multiple analysis aids, a powerful user-interface, and top-down design capability. The technologies include MOS design, bipolar design, schematic, printed circuit board (PCB) design, and general artwork environments. The analysis aids handle simulation interfaces, incremental design-rule checking, textual I/O data conversion, and a hierarchical constraint system. Most importantly, the system implements a flexible model of circuit representation that allows hierarchical top-down design, This is done by propagating the constraints in a bottom-up fashion so that the entire circuit is always properly connected.

Circuits are represented as networks that contain nodes and connecting arcs. The nodes are electrical components such as transistors, logic gates, and electrical contacts. The arcs are simply wires that connect to nodes. In addition, each node has a set ports which are sites of arc connection. Collections of nodes and arcs can also be aggregated into cells which can be used higher in the

hierarchy to act as nodes. These user-defined nodes have ports that come from internal nodes whose ports are exported. Cells are collected in libraries which contain a hierarchically consistent design. Arcs have properties that help constrain the design. For example, an arc can be orthogonal to the axes (manhattan) or rubber-band flexible. Arcs can also be stretchable or rigid under modification of their connecting nodes. The constraints propagate hierarchically from the bottom-up.

The incremental design-rule checker is normally on and watches all changes made to the circuit. It does not correct but indicates the error in textual window and in graphic window when design rules are violated. Hierarchy is not handled, so the contents of sub-cells are not checked. Also, non-manhattan geometry is not handled. There is an option in the design rule checker to find short circuits on a global basis. This does not check all design rules: merely the shorting together of layout on different electrical nets.

A 2-D compactor attempts to reduce the size of a cell by removing unnecessary space between elements. It can be invoked with **tellaid**. It does not do hierarchical compaction, does not guarantee optimal compaction, nor can it handle non-manhattan geometry properly. The compactor will also spread out the cell to guarantee no design-rule violations, if the **spread** option is set. This option can also be over-ruled.

There are seven simulation interfaces: ESIM, RSIM, RNL, MOSSIM, MARS, CADAT, and SPICE. The first six simulators are for switch-level simulation. In preparation for most simulators, it is necessary to export those ports that we wish to manipulate or

examine. Power and ground ports must be exported. Similarly, clock signals must be also exported. The most interesting simulation interface is SPICE. In preparation for SPICE simulation, it is best to invoke the command file *spice.mac* which provides all necessary commands for SPICE input specification. Suppose we have a complete mask layout and we wish to create a SPICE deck. We must indicate power source, input signal, circuit nodes which will be examined, and type of circuit analysis. For example, to make a 5-volt supply, use:

-setsource v "DC 5"

and if an input signal source is to produce values—that are 5-volts from 0 to 5NS and then 0-volt, use:

-setsource v "PULSE(0 5 0NS 0NS 0NS 5NS 20NS)

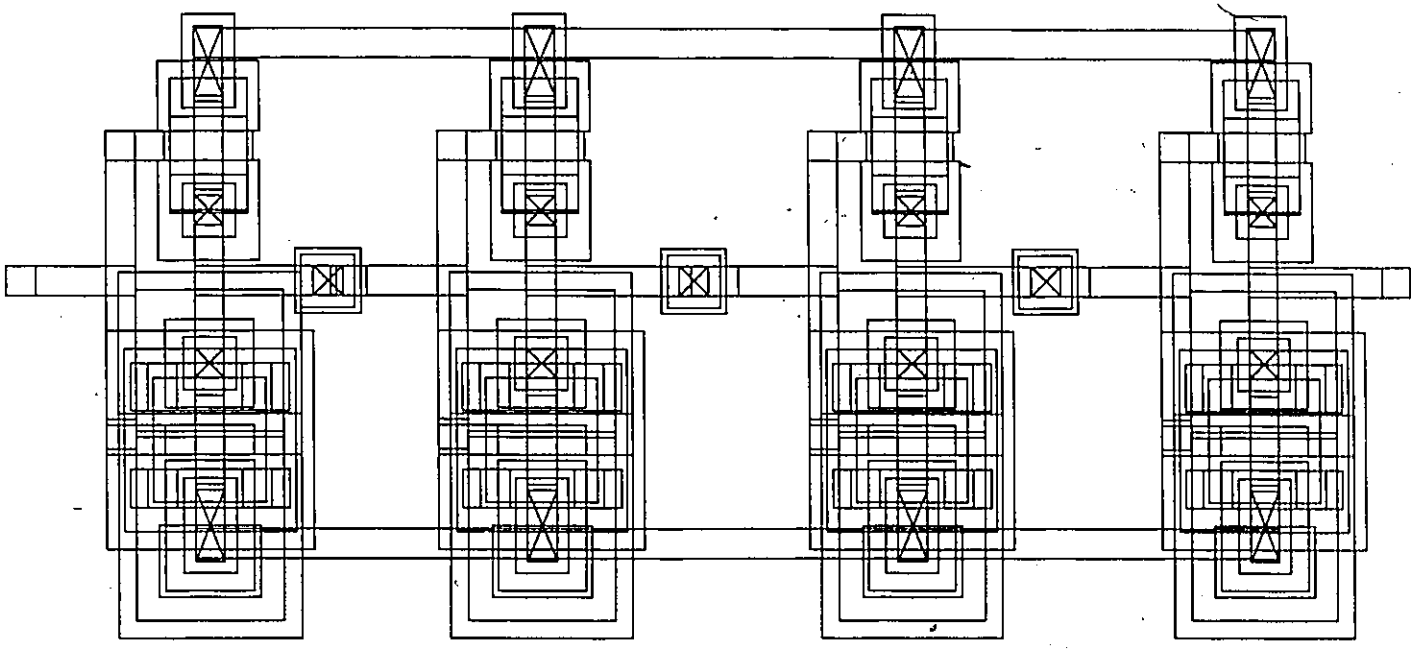
Next, all values that are being examined and plotted must be exported and have meter nodes placed on them. If a meter is used to watch voltage from 0 to 5 volts, use:

-setmeter "(0,5)"

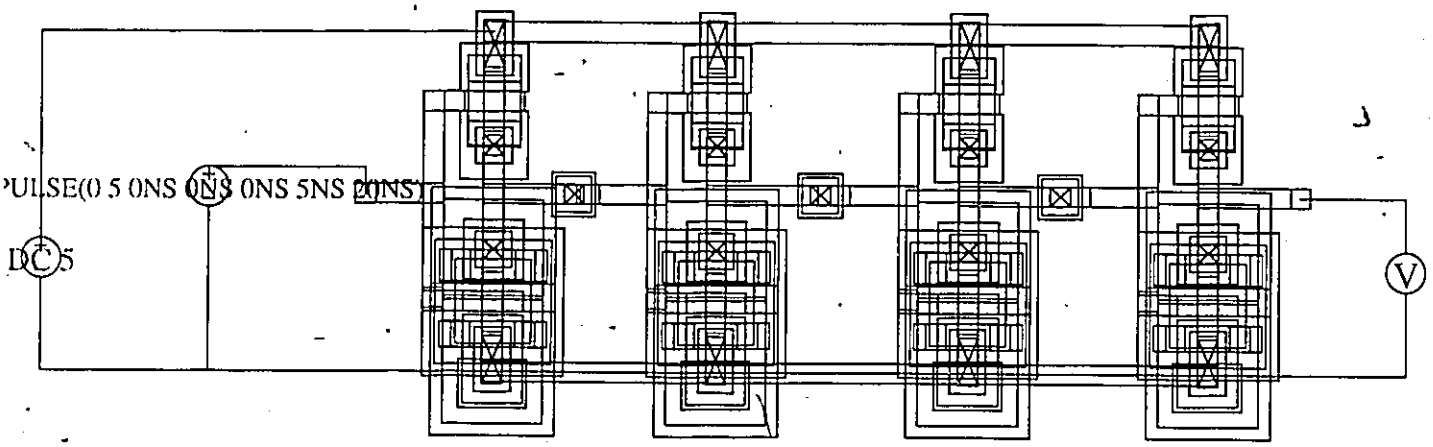
Finally the type of circuit analysis to the SPICE system must be specified. For transient analysis an unconnected source node must be given. Use:

-setsource t "0.2NS 20NS"

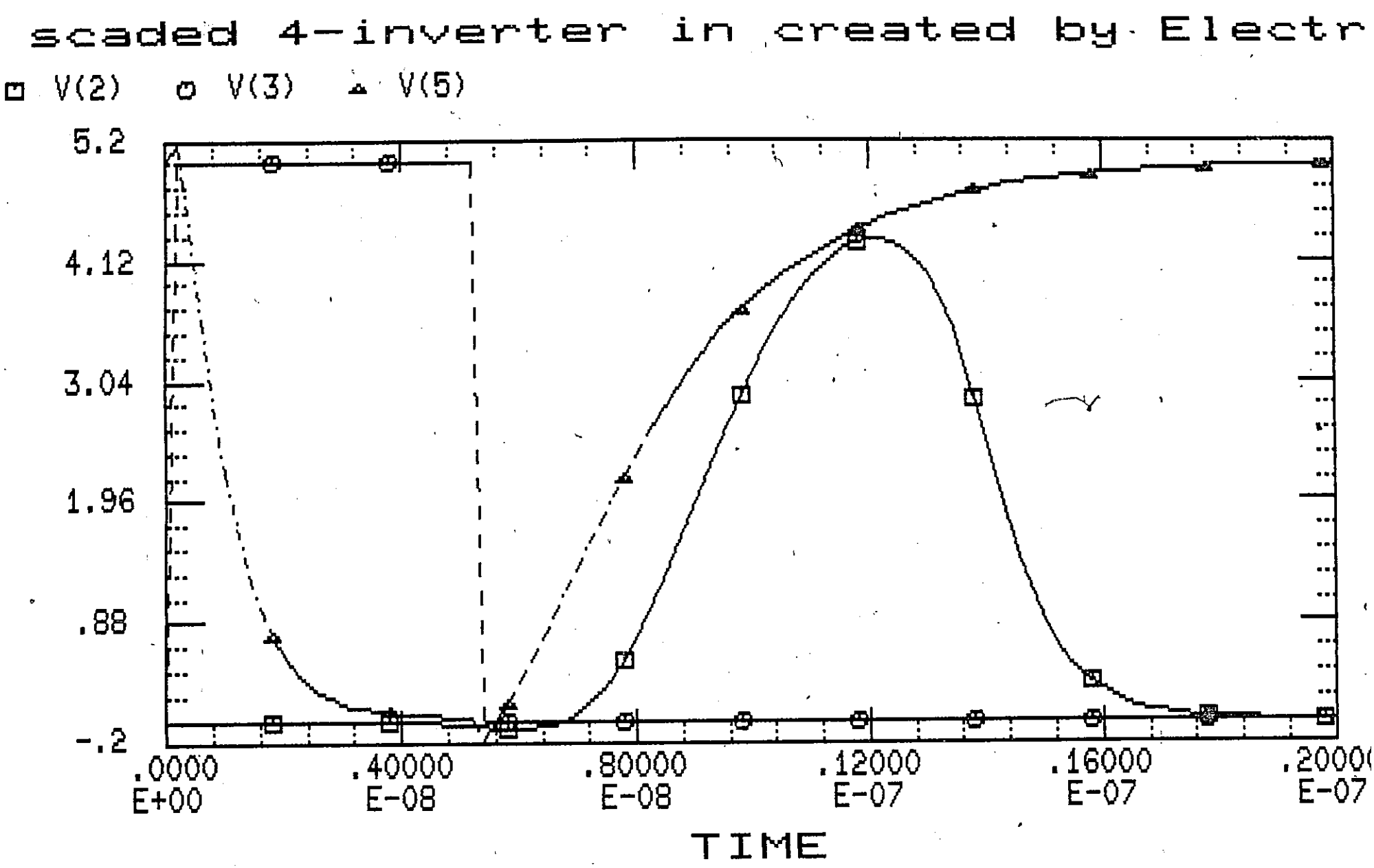
Now, to create a SPICE deck, the **deck** command is issued to create a *sim.spi* file which is ready to be read by SPICE simulation program. For example, four cascaded inverters design with the graphical SPICE input is shown in the following diagram. The SPICE input file of the cascaded inverters and the simulation results are also in the following diagrams.



TRAN 0.2NS 20NS



```
Cascaded 4-inverter in  Series
*** UC SPICE *** , MIN_RESIST 50.000000, MIN_CAPAC 0.040000PF
.OPTIONS NOMOD NOPAGE
* Northern Telecom 3 Micron CMOS PROCESS
* Models Taken From CMC Document IC87-1

* Modified by Alger Yeung on 9/4/87

.OPTIONS DEFL=3UM DEFW=3UM DEFAS=60PM^2 DEFAD=60PM^2
+        LIMPTS=20000 ITL3=10 ITL4=30 ITL5=40000
+        LVLTIM=2  ITL6=30 METHOD=TRAP GMIN=1.E-10
+        DEFL=3U DEFW=3U ABSTOL=10PA VNTOL=10UV

.MODEL N NMOS ( LEVEL=1 VTO=0.7 KP=4.0E-5 GAMMA=1.1 PHI=0.6
+        LAMBDA=1.0E-2 PB=0.7 CGSO=3.0E-10 CGDO=3.0E-10 CGBO=5.0E-10
+        RSH=25 CJ=4.4E-4 MJ=0.5 CJSW=4.0E-10 MJSW=0.3 JS=1.0E-5
+        TOX=5.0E-8 NSUB=1.7E+16 TPG=1 XJ=6.E-7 LD=3.5E-7 UO=775 )

.MODEL P PMOS ( LEVEL=1 VTO=-0.8 KP=12E-6 GAMMA=0.6 PHI=0.6
+        LAMBDA=3.0E-2 PB=0.6 CGSO=2.5E-10 CGDO=2.5E-10 CGBO=5.0E-10
+        RSH=80 CJ=1.5E-4 MJ=0.6 CJSW=4.0E-10 MJSW=0.6 JS=1.0E-5
+        TOX=5.0E-8 NSUB=5.0E+15 TPG=1 XJ=5.E-7 LD=2.5E-7 UO=250 )

.SUBCKT HIGHER 4 2 1
** PIN 4: POWER
** PIN 2: OUT1
** PIN 1: INP1
*** COMPONENT N-Transistor:
M1 2 1 0 0 N L=5.00U W=5.00U AS=222.00P AD=273.50P
*** COMPONENT P-Transistor:
M2 2 1 4 4 P L=5.00U W=5.00U AS=222.00P AD=273.50P
.ENDS HIGHER
* POWER=4
* OUTPUT=2
* INPUT=3
* INTERMEDATE=5
X1 4 7 5 HIGHER
X2 4 2 6 HIGHER
X3 4 6 7 HIGHER
X4 4 5 3 HIGHER
V2 4 0 DC 5
V1 3 0 PULSE(0 5 0NS 0NS 0NS 5NS 20NS)
.TRAN .2NS 20NS
.WIDTH OUT=124
.PRINT TRAN V(2) V(3) V(5)
.PRINT TRAN V(2) V(7)
.END
```


scaded 4-inverter in created by Electr
V(2)
V(3)
V(5)
5.2
4.12
3.04
1.96
.88
-.2
.0000 E+00
.40000 E-08
.80000 E-08
.12000 E-07
.16000 E-07
.2000(E-07
TIME

A last section about the Electric is the network comparison. A network maintainer is able to compare the networks in the cells being displayed on the screen. Use **window split** to edit two cells at once, then use **tellaid network compare** to compare them. Once compared, nodes in one cell can be equated with nodes in the other simply by selecting one and using **tellaid network compare highlight-other**. For example, if a schematic and the corresponding mask layout are compared, it would be very easily to check whether the network of the mask layout is identical to the one in the schematic.

For more information about Electric, use the *help* facility on the Electric design system or refer to Electric reference manual and Electric tutorial. Both can be obtained from the GPX workstation by typing:

```
tbl doc/electric.1 | troff -man
troff -ms doc/electutor.nr
```

More information regarding the creation of technology file, database variables and routines can also be found in the workstation.

Appendix III

Circuit Designs and Simulations

Appendix III

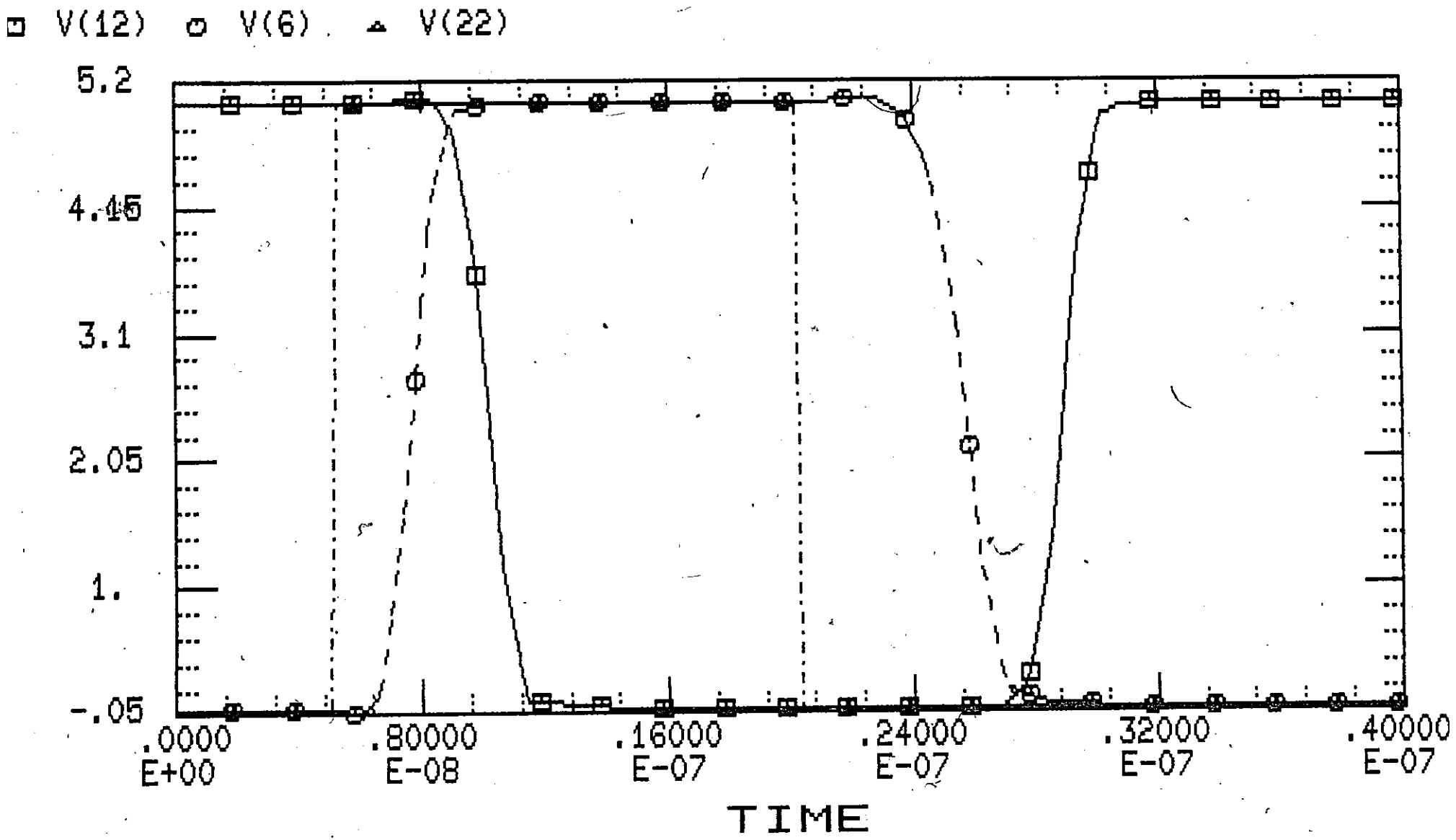
Circuit designs and simulations

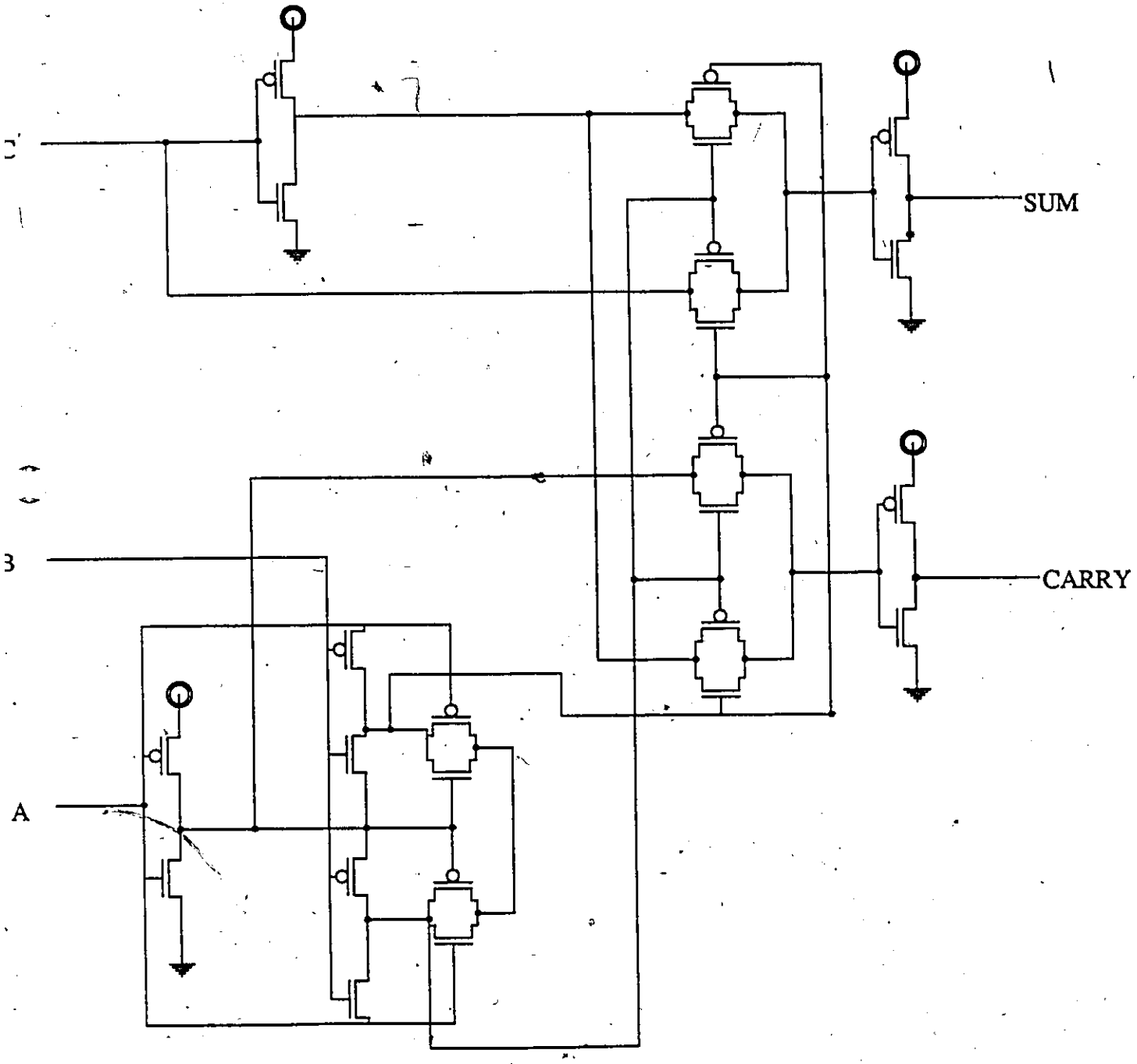
This appendix shows a number of circuit designs and the circuit simulation results. The designs include a 4-bit ripple carry adder, a 4-bit transmission gate adder, and a four-by-four shift register. The schematic diagrams, SPICE deck input files, mask layouts are shown in the following.

```
1-BIT RIPPLE CARRY ADDER
*** UC SPICE *** , MIN_RESIST 0.000000, MIN_CAPAC 0.000000PF
.OPTIONS NOMOD NOPAGE

* Northern Telecom 3 Micron CMOS PROCESS
* Models Taken From CMC Document IC87-1

* Modified by Alger Yeung on 9/4/87

.OPTIONS DEFL=3UM DEFW=3UM DEFAS=60PM^2 DEFAD=60PM^2
+        LIMPTS=20000 ITL3=10 ITL4=30 ITL5=40000
+        LVLTIM=2  ITL6=30 METHOD=TRAP GMIN=1.E-10
+        DEFL=3U DEFW=3U ABSTOL=10PA VNTOL=10UV

.MODEL N NMOS ( LEVEL=1 VTO=0.7 KP=4.0E-5 GAMMA=1.1 PHI=0.6
+        LAMBDA=1.0E-2 PB=0.7 CGSO=3.0E-10 CGDO=3.0E-10 CGBO=5.0E-10
+        RSH=25 CJ=4.4E-4 MJ=0.5 CJSW=4.0E-10 MJSW=0.3 JS=1.0E-5
+        TOX=5.0E-8 NSUB=1.7E+16 TPG=1 XJ=6.E-7 LD=3.5E-7 UO=775 )

.MODEL P PMOS ( LEVEL=1 VTO=-0.8 KP=12E-6 GAMMA=0.6 PHI=0.6
+        LAMBDA=3.0E-2 PB=0.6 CGSO=2.5E-10 CGDO=2.5E-10 CGBO=5.0E-10
+        RSH=80 CJ=1.5E-4 MJ=0.6 CJSW=4.0E-10 MJSW=0.6 JS=1.0E-5
+        TOX=5.0E-8 NSUB=5.0E+15 TPG=1 XJ=5.E-7 LD=2.5E-7 UO=250 )


.SUBCKT SCHEM 21 22 23 6 12 20 3 8
** PIN 6: CARRY
** PIN 12: SUM
** PIN 23: C
** PIN 22: B
** PIN 21: A
** PIN 20: POWER
*** COMPONENT Transistor:
M1 1 21 20 20 P
*** COMPONENT Transistor:
M2 1 22 20 20 P
*** COMPONENT Transistor:
M3 2 22 1 20 P
*** COMPONENT Transistor:
M4 3 23 1 20 P
*** COMPONENT Transistor:
M5 3 21 2 20 P
*** COMPONENT Transistor:
M6 3 23 4 0 N
*** COMPONENT Transistor:
M7 3 21 5 0 N
*** COMPONENT Transistor:
M8 4 21 0 0 N
*** COMPONENT Transistor:
M9 4 22 0 0 N
*** COMPONENT Transistor:
M10 5 22 0 0 N
*** COMPONENT Transistor:
M11 9 23 20 20 P
```

```
*** COMPONENT Transistor:
M12 9 21 20 20 P
*** COMPONENT Transistor:
M13 9 22 20 20 P
*** COMPONENT Transistor:
M14 10 21 9 20 P
*** COMPONENT Transistor:
M15 11 22 10 20 P
*** COMPONENT Transistor:
M16 8 3 9 20 P
*** COMPONENT Transistor:
M17 8 23 11 20 P
*** COMPONENT Transistor:
M18 8 3 7 0 N
*** COMPONENT Transistor:
M19 7 21 0 0 N
*** COMPONENT Transistor:
M20 7 22 0 0 N
*** COMPONENT Transistor:
M21 7 23 0 0 N
*** COMPONENT Transistor:
M22 8 23 13 0 N
*** COMPONENT Transistor:
M23 13 21 14 0 N
*** COMPONENT Transistor:
M24 14 22 0 0 N
*** COMPONENT Transistor:
M25 6 3 20 20 P
*** COMPONENT Transistor:
M26 6 3 0 0 N
*** COMPONENT Transistor:
M27 12 8 20 20 P
*** COMPONENT Transistor:
M28 12 8 0 0 N
.ENDS SCHEM
X1 21 22 23 6 12 20 3 8 SCHEM

V2 20 0 DC 5
V3 23 0 DC 0
V4 21 0 DC 5
V1 22 0 PULSE(0 5 5NS 0NS 0NS 15NS 40NS)

.TRAN 0.2NS 40NS
.PRINT TRAN V(12) V(6) V(22)
.WIDTH OUT=125
.END
```

1-BIT RIPPLE CARRY ADDER

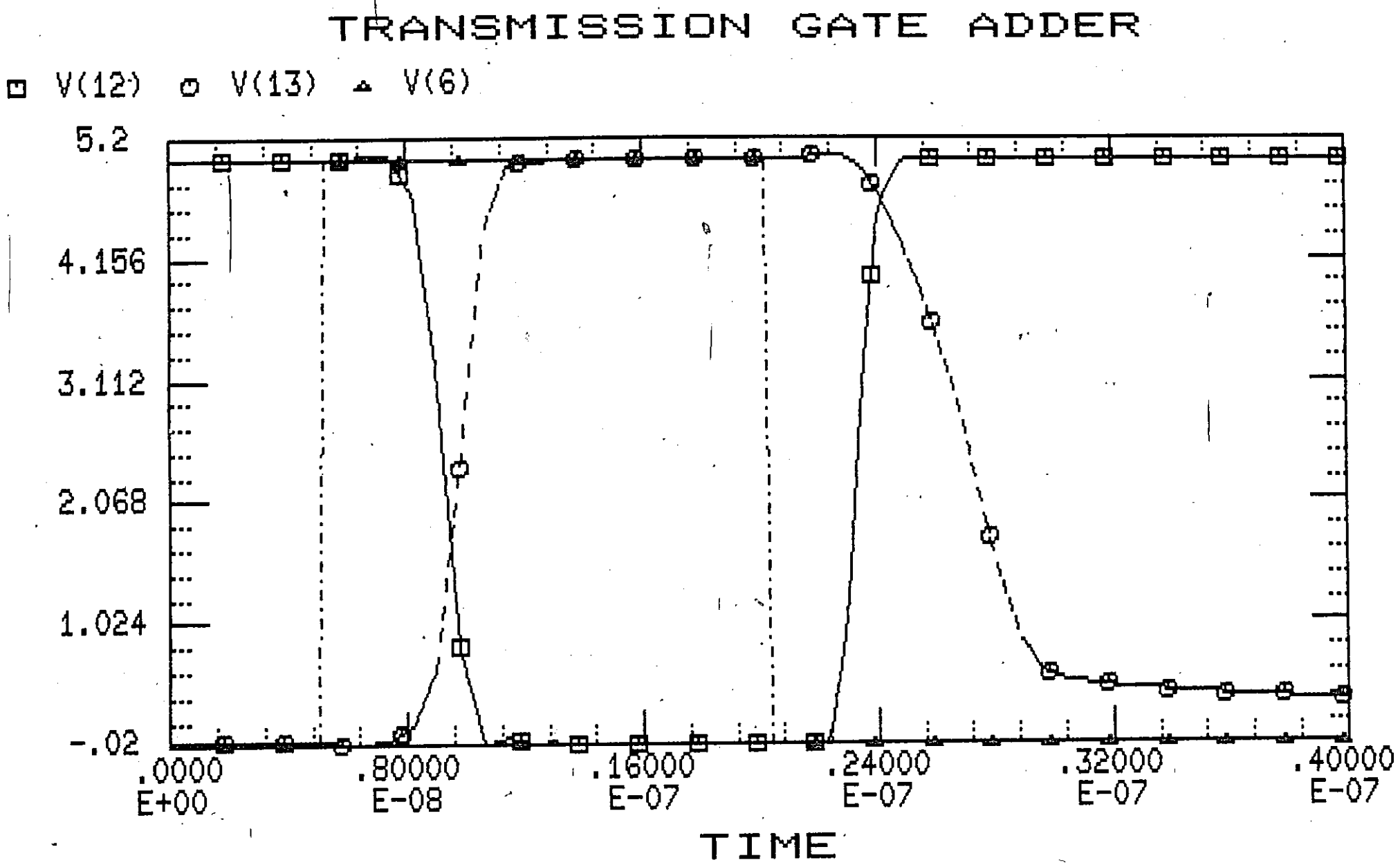


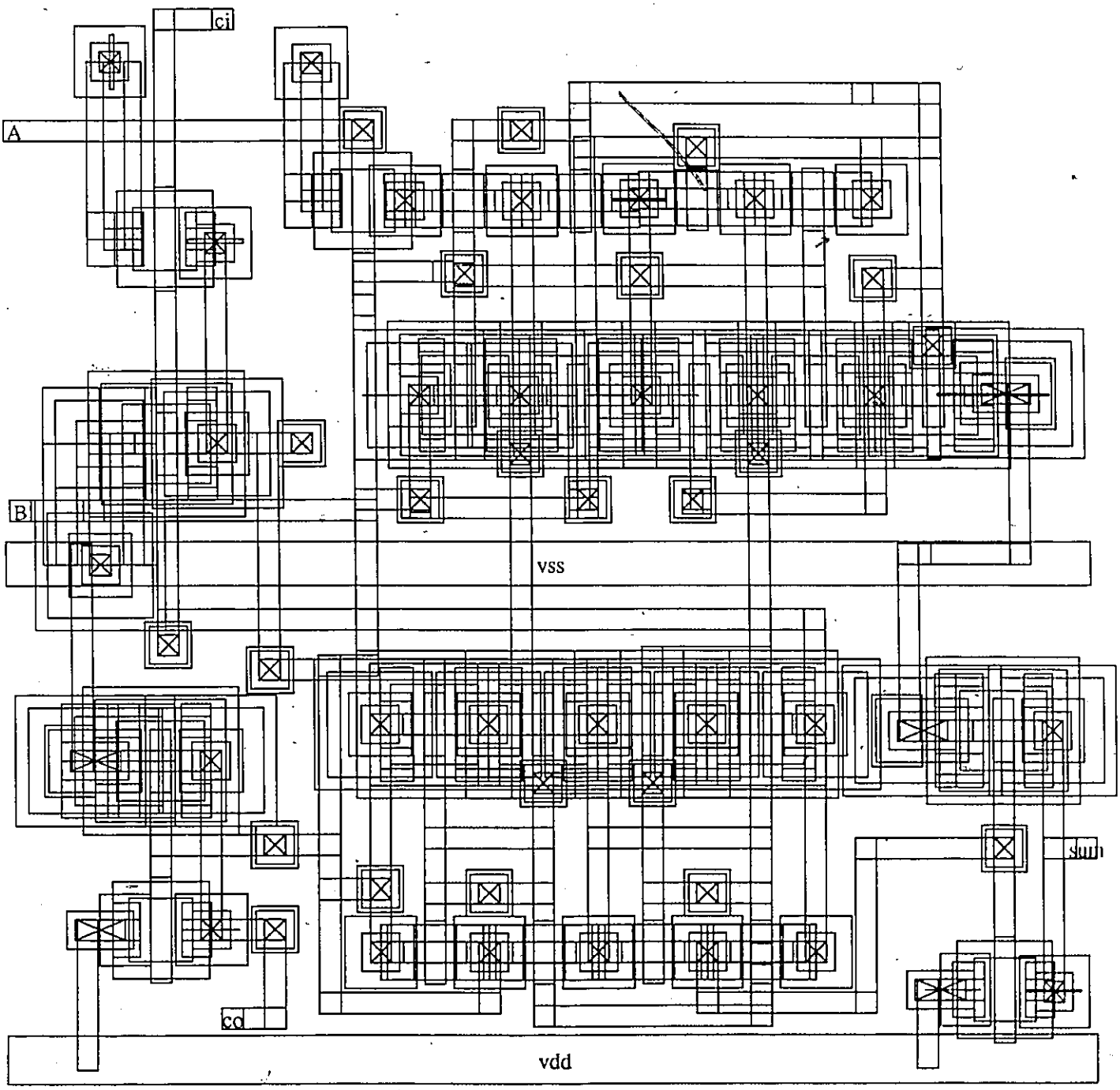


Schematic of 1-bit transmission gate adder

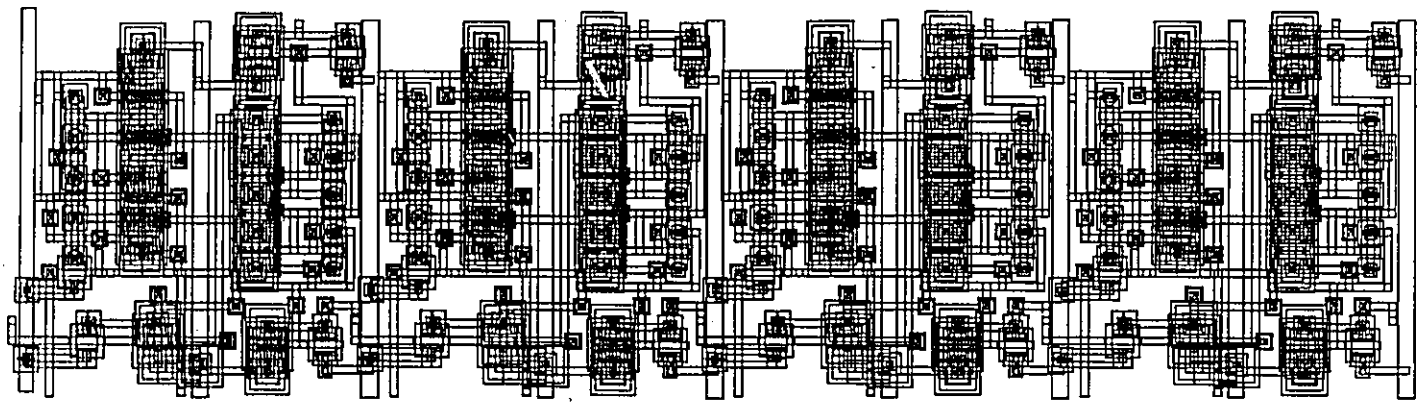
```
TRANSMISSION GATE ADDER
*** UC SPICE *** , MIN_RESIST 0.000000, MIN_CAPAC 0.000000PF
.OPTIONS NOMOD NOPAGE

* Northern Telecom 3 Micron CMOS PROCESS
* Models Taken From CMC Document IC87-1

* Modified by Alger Yeung on 9/4/87

.OPTIONS DEFL=3UM DEFW=3UM DEFAS=60PM^2 DEFAD=60PM^2
+        LIMPTS=20000 ITL3=10 ITL4=30 ITL5=40000
+        LVLTIM=2  ITL6=30 METHOD=TRAP GMIN=1.E-10
+        DEFL=3U DEFW=3U ABSTOL=10PA VNTOL=10UV

.MODEL N NMOS ( LEVEL=1 VTO=0.7 KP=4.0E-5 GAMMA=1.1 PHI=0.6
+        LAMBDA=1.0E-2 PB=0.7 CGSO=3.0E-10 CGDO=3.0E-10 CGBO=5.0E-10
+        RSH=25 CJ=4.4E-4 MJ=0.5 CJSW=4.0E-10 MJSW=0.3 JS=1.0E-5
+        TOX=5.0E-8 NSUB=1.7E+16 TPG=1 XJ=6.E-7 LD=3.5E-7 UO=775 )

.MODEL P PMOS ( LEVEL=1 VTO=-0.8 KP=12E-6 GAMMA=0.6 PHI=0.6
+        LAMBDA=3.0E-2 PB=0.6 CGSO=2.5E-10 CGDO=2.5E-10 CGBO=5.0E-10
+        RSH=80 CJ=1.5E-4 MJ=0.6 CJSW=4.0E-10 MJSW=0.6 JS=1.0E-5
+        TOX=5.0E-8 NSUB=5.0E+15 TPG=1 XJ=5.E-7 LD=2.5E-7 UO=250 )


.SUBCKT SCHEMATIC 13 12 8 6 4 2
** PIN 13: CARRY
** PIN 12: SUM
** PIN 8: C
** PIN 6: B
** PIN 4: A
** PIN 2: POWER
*** COMPONENT Transistor:
M1 0 10 12 0 N
*** COMPONENT Transistor:
M2 0 11 13 0 N
*** COMPONENT Transistor:
M3 4 6 5 0 N
*** COMPONENT Transistor:
M4 14 6 3 0 N
*** COMPONENT Transistor:
M5 0 4 14 0 N
*** COMPONENT Transistor:
M6 0 8 7 0 N
*** COMPONENT Transistor:
M7 5 4 9 0 N
*** COMPONENT Transistor:
M8 3 14 9 0 N
*** COMPONENT Transistor:
M9 7 3 11 0 N
*** COMPONENT Transistor:
M10 14 5 11 0 N
*** COMPONENT Transistor:
M11 8 3 10 0 N
```

```
*** COMPONENT Transistor:
M12 7 5 10 0 N
*** COMPONENT Transistor:
M13 14 4 2 2 P
*** COMPONENT Transistor:
M14 13 11 2 2 P
*** COMPONENT Transistor:
M15 12 10 2 2 P
*** COMPONENT Transistor:
M16 5 6 14 2 P
*** COMPONENT Transistor:
M17 3 6 4 2 P
*** COMPONENT Transistor:
M18 7 8 2 2 P
*** COMPONENT Transistor:
M19 5 14 9 2 P
*** COMPONENT Transistor:
M20 3 4 9 2 P
*** COMPONENT Transistor:
M21 7 3 10 2 P
*** COMPONENT Transistor:
M22 7 5 11 2 P
*** COMPONENT Transistor:
M23 14 3 11 2 P
*** COMPONENT Transistor:
M24 8 5 10 2 P
.ENDS SCHEMATIC
** PIN 13: CARRY
** PIN 12: SUM
** PIN 8: C
** PIN 6: B
** PIN 4: A
** PIN 2: POWER
X1 13 12 8 6 4 2 SCHEMATIC

V2 2 0 DC 5
V3 8 0 DC 0
V4 4 0 DC 5
V1 6 0 PULSE(0 5 5NS 0NS 0NS 15NS 40NS)

.TRAN 0.2NS 40NS
.PRINT TRAN V(12) V(13) V(6)
.WIDTH OUT=125
.END
```

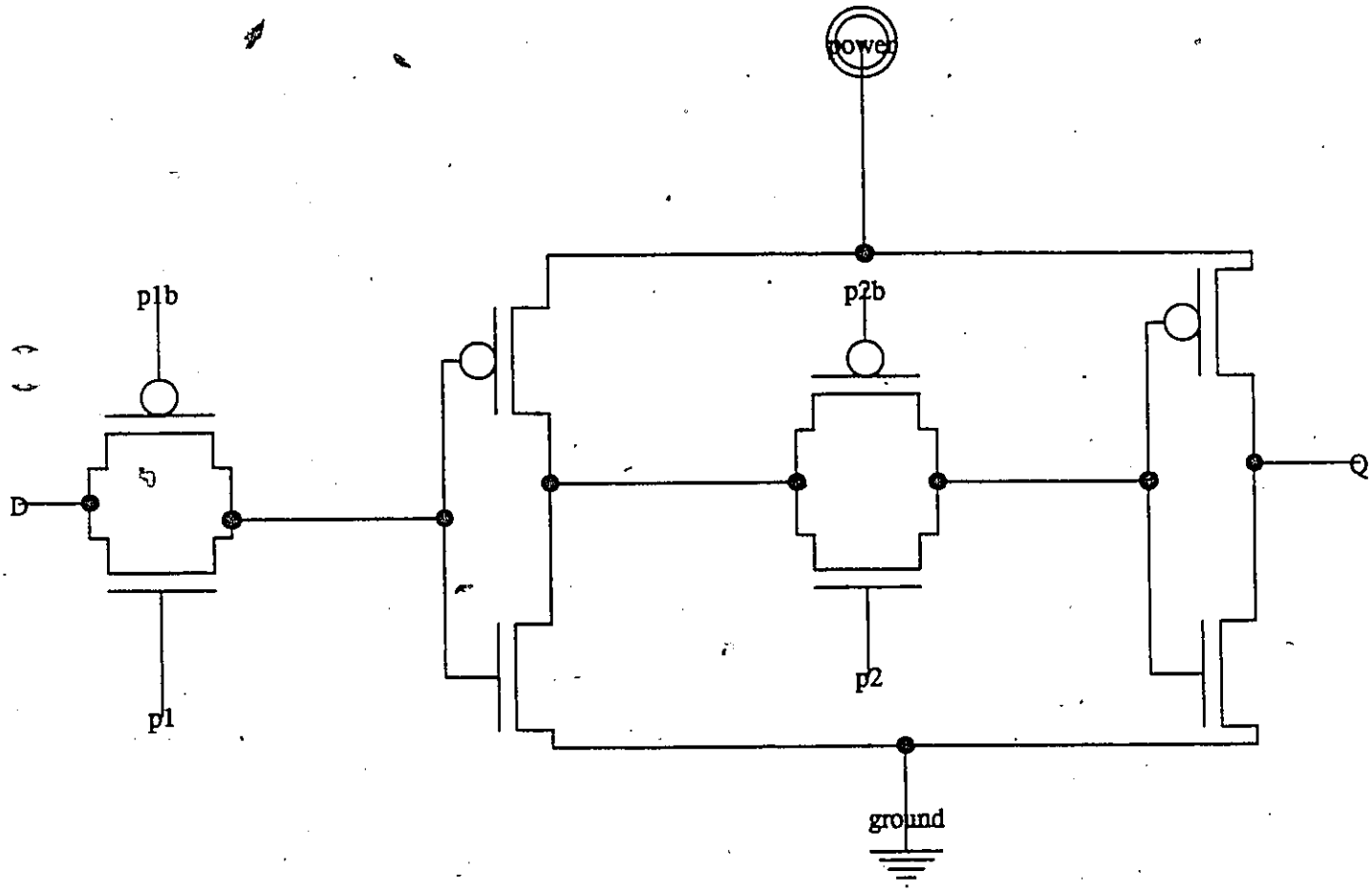

TRANSMISSION GATE ADDER
V(12)
V(13)
V(6)
5.2
4.156
3.112
2.068
1.024
-.02
.0000
E+00
.80000
E-08
.16000
E-07
.24000
E-07
.32000
E-07
.40000
E-07
TIME



Mask layout of 1-bit transmission gate adder



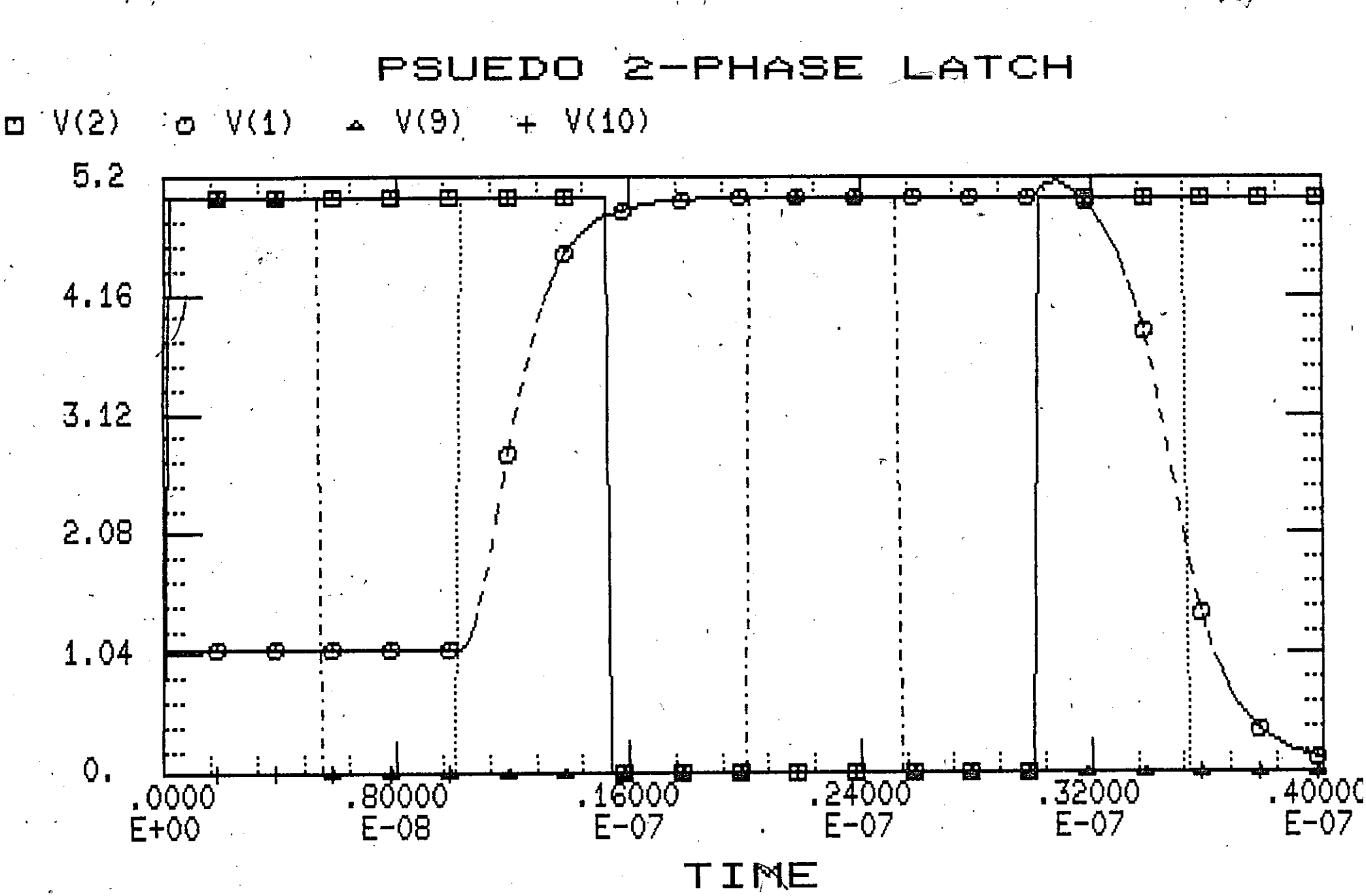
Mask layout of a 4-bit transmission gate adder

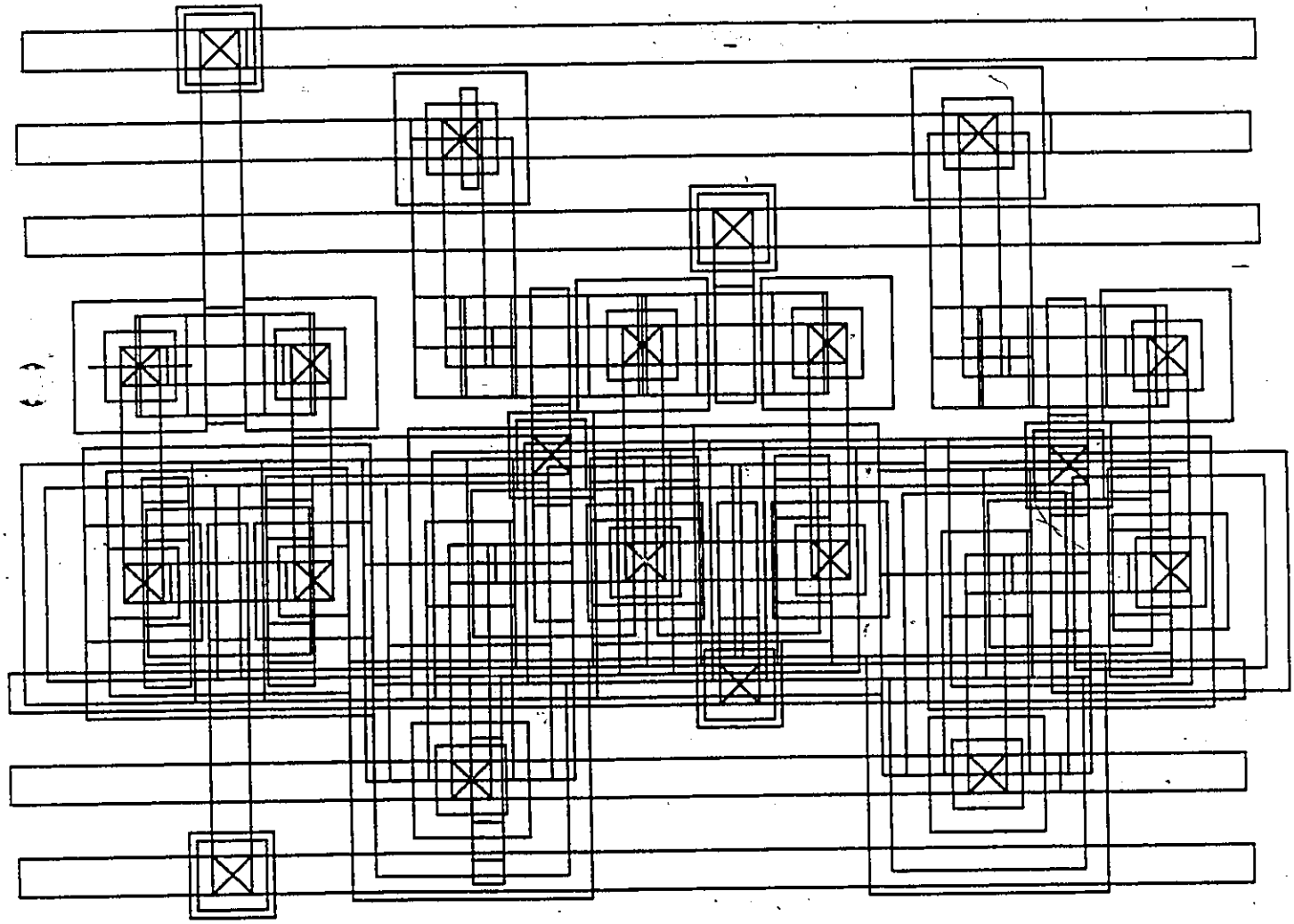


Schematic of pseudo 2-phase latch

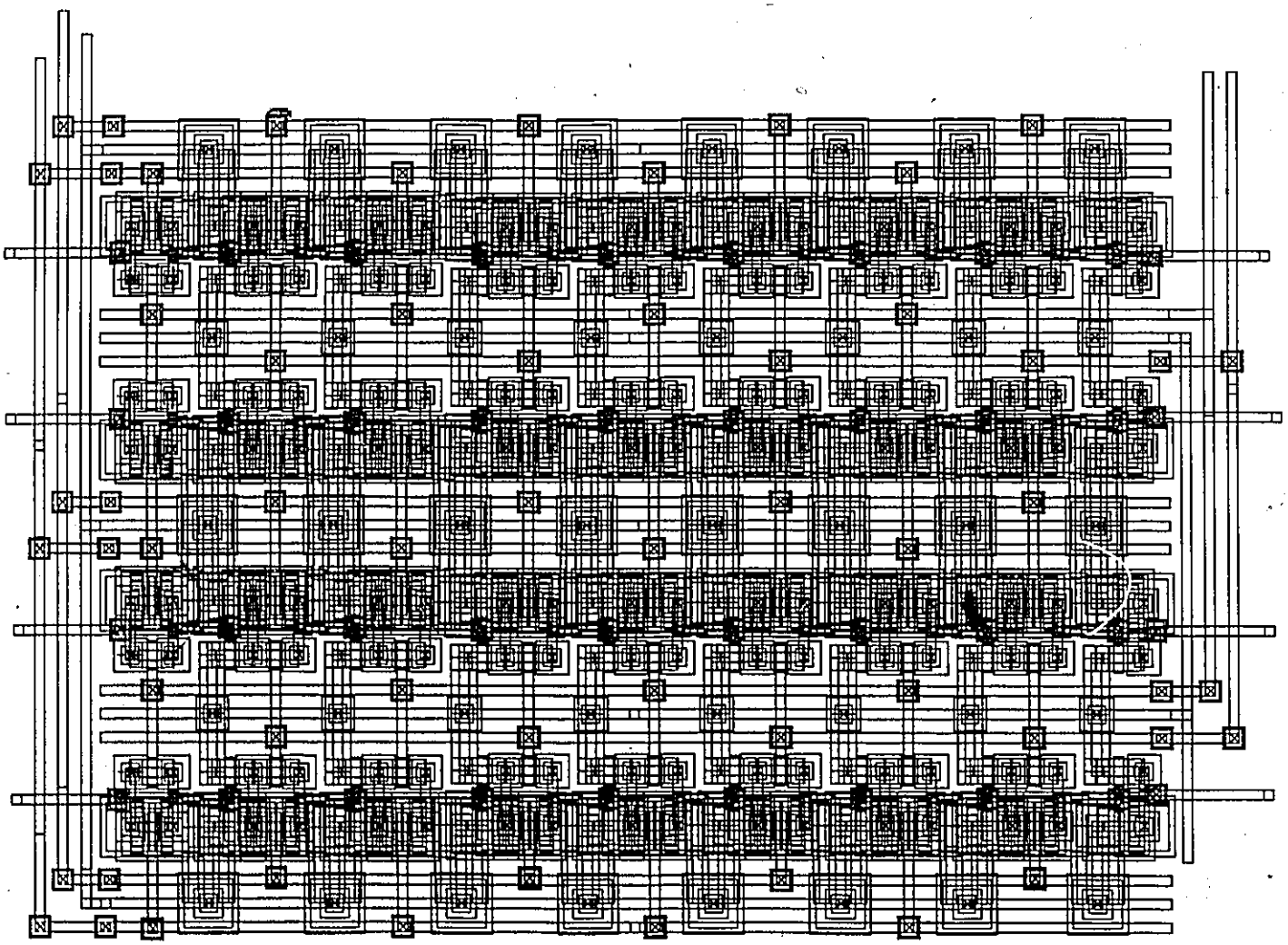
```
PSUEDO 2-PHASE LATCH
*** UC SPICE *** , MIN_RESIST 50.000000, MIN_CAPAC 0.040000PF
.OPTIONS NOMOD NOPAGE
* Northern Telecom 3 Micron CMOS PROCESS
* Models Taken From CMC Document IC87-1

* Modified by Alger Yeung on 9/4/87

.OPTIONS DEFL=3UM DEFW=3UM DEFAS=60PM^2 DEFAD=60PM^2
+        LIMPTS=20000 ITL3=10 ITL4=30 ITL5=40000
+        LVLTIM=2  ITL6=30 METHOD=TRAP GMIN=1.E-10
+        DEFL=3U DEFW=3U ABSTOL=10PA VNTOL=10UV

.MODEL N NMOS ( LEVEL=1 VTO=0.7 KP=4.0E-5 GAMMA=1.1 PHI=0.6
+        LAMBDA=1.0E-2 PB=0.7 CGSO=3.0E-10 CGDO=3.0E-10 CGBO=5.0E-10
+        RSH=25 CJ=4.4E-4 MJ=0.5 CJSW=4.0E-10 MJSW=0.3 JS=1.0E-5
+        TOX=5.0E-8 NSUB=1.7E+16 TPG=1 XJ=6.E-7 LD=3.5E-7 UO=775 )

.MODEL P PMOS ( LEVEL=1 VTO=-0.8 KP=12E-6 GAMMA=0.6 PHI=0.6
+        LAMBDA=3.0E-2 PB=0.6 CGSO=2.5E-10 CGDO=2.5E-10 CGBO=5.0E-10
+        RSH=80 CJ=1.5E-4 MJ=0.6 CJSW=4.0E-10 MJSW=0.6 JS=1.0E-5
+        TOX=5.0E-8 NSUB=5.0E+15 TPG=1 XJ=5.E-7 LD=2.5E-7 UO=250 )

.SUBCKT LATCH 3 9 7 10 8 2 1
** PIN 3: P1BAR
** PIN 9: P1
** PIN 7: P2BAR
** PIN 10: P2
** PIN 8: POWER
** PIN 2: D
** PIN 1: Q
*** COMPONENT P-Transistor:
M1 2 3 4 8 P L=5.00U W=5.00U AS=217.00P AD=108.50P
*** COMPONENT N-Transistor:
M2 2 9 4 0 N L=5.00U W=5.00U AS=217.00P AD=108.50P
*** COMPONENT P-Transistor:
M3 5 7 6 8 P L=5.00U W=5.00U AS=181.50P AD=111.00P
*** COMPONENT N-Transistor:
M4 5 10 6 0 N L=5.00U W=5.00U AS=181.50P AD=111.00P
*** COMPONENT P-Transistor:
M5 8 4 5 8 P L=5.00U W=15.00U AS=502.50P AD=181.50P
*** COMPONENT N-Transistor:
M6 0 4 5 0 N L=5.00U W=5.00U AS=477.50P AD=181.50P
*** COMPONENT P-Transistor:
M7 8 6 1 8 P L=5.00U W=15.00U AS=502.50P AD=227.00P
*** COMPONENT N-Transistor:
M8 0 6 1 0 N L=5.00U W=5.00U AS=477.50P AD=227.00P
.ENDS LATCH
X1 3 9 7 10 8 2 1 LATCH
.WIDTH OUT=125
** PIN 3: P1BAR
** PIN 9: P1
** PIN 7: P2BAR
** PIN 10: P2
```

```
** PIN 8: POWER
** PIN 2: D
** PIN 1: Q
VPOWER 8 0 DC 5
VP1 9 0 PULSE(0 5 0NS 0NS 0NS 5NS 20NS)
VPBAR1 3 0 PULSE(0 5 5NS 0NS 0NS 15NS 20NS)
VP2 10 0 PULSE(0 5 10NS 0NS 0NS 5NS 20NS)
VPBAR2 7 0 PULSE(5 0 10NS 0NS 0NS 05NS 20NS)
*VD 2 0 PULSE(0 5 0NS 0NS 0NS 5NS 20NS)
VD 2 0 PULSE(0 5 0NS 0NS 0NS 15NS 30NS)

*##############################
* Start of Output Generation
*##############################
.TRAN 0.2NS 40NS
.PRINT TRAN V(2) V(1) V(9) V(10)
.END
```


PSUEDO 2-PHASE LATCH
V(2)
V(1)
V(9)
V(10)
5.2
4.16
3.12
2.08
1.04
0.
.0000
E+00
.80000
E-08
.16000
E-07
.24000
E-07
.32000
E-07
.40000
E-07
TIME



Mask layout of pseudo 2-phase latch



Appendix IV

Graphic SPICE Manual

Table of Contents

Graphic Spice Simulation Tutorial for the DEC VT241 Colour Terminal

Alger W. Yeung
CAD/CAM Centre
University of Windsor
Windsor, Ontario

GRAPHIC SPICE SIMULATION TUTORIAL

SPICE is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analyses. SPICE is capable of handling circuit with resistors, capacitors, inductors, mutual inductors, voltage and current sources, and semiconductor devices, namely diode, BJT, JFET AND MOSFET. SPICE has been widely used in VLSI circuit simulation because of it's built-in models for the semiconductor devices. In general, circuit simulation is a 3-stage process:

1. Prepare circuit description in SPICE form.
2. Invoke GSPICE simulation program.
3. Interpret the simulation result.

In the first stage, we must prepare a circuit description file which shows the connections between different electrical components in a netlist form and indicates the desired circuit analysis to be performed by GSPICE. In the next stage, we invoke the SPICE simulation program, and specify the circuit description file. The result of the simulation is stored in specified files. In the last stage, the simulation result is displayed in either tabular or graph form and judged whether, or not, it is satisfactory. If not, the whole process must be performed again with a modified circuit description file until the simulation result is satisfactory.

I. Preparation of Circuit Description

This tutorial is not intend to show you how to prepare the circuit description file in SPICE form. This tutorial, however, will concentrate on the second and last stages. We will show you how to run the GSPICE program in both interactive and batch mode. The result will be displayed graphically by using the GRAPHIC program which allows the user to see one, two or all curves on a single graph. A zoom function is also incorporated into the display program so that the user can examine the result in more detail. The creation of circuit description file is completely explained in the SPICE manuals available from the Department of Electrical Engineering.

A simple circuit description file is prepared to analyze the performance of a CMOS inverter. The file, shown below, defines a subcircuit of an inverter gate. Power and ground are specified and the type of analysis is also indicated. All the nodes of the circuit are numbered. A .PRINT command is used to indicate that the result of the analysis at the specified nodes will be stored in a tabular form. The .PRINT command serves two purposes:

1. Store the data in a tabular form.

2. Prepare a data file which is used to obtain colour plots in a VT240 terminal.

Since the .PRINT command will allow the plotting of graphs through the program GRAPHIC, the .PLOT command is not necessary in order to obtain graphical output.

```
Test of Inverter Cct.
*_____
* MOS TRANSISTOR MODELS
*_____
* DRAIN GATE SOURCE SUBSTRATE MOD1 L=3U W=3U
* nmos enhancement
.OPTIONS DEFW=3U DEFL=3U DEFAD=81P DEFAS=81P NOMOD LIMPTS=2000
.TEMP 27
.MODEL NMOSE NMOS LEVEL=3 KP=50.0E-6 VTO=0.7 TOX=5E-8 GAMMA=1.1
```

```
+PHI=0.6 LAMBDA=0.01 RD=40 RS=40 PB=0.7 CGSO=3E-10 CGBO=5E-10
+CGDO=3E-10 RSH=25 CJ=44E-5 MJ=0.5 CJSW=4E-10 MJSW=0.3 JS=1E-5
+NSUB=1.7E16 XJ=6E-7 LD=3.5E-7 UO=775 VMAX=1E5 THETA=0.11
+ETA=0.05 KAPPA=1
* pmos enhancement
.MODEL PMOSE PMOS LEVEL=3 KP=16.0E-6 VTO=-0.8 TOX=5E-8 GAMMA=0.6
+PHI=0.6 LAMBDA=0.03 RD=100 RS=100 PB=0.6 CGSO=2.5E-10 CGBO=5E-10
+CGDO=2.5E-10 RSH=80 CJ=15E-5 MJ=0.6 CJSW=4E-10 MJSW=0.6 JS=1E-5
+NSUB=5E15 XJ=5E-7 LD=2.5E-7 UO=250 VMAX=0.7E5 THETA=0.13
+ETA=0.3 KAPPA=1
*_____
* Start of Subcircuits
*_____
*_____
* Simple Inverter
*_____
* 1=VCC 2=VSS 3=INPUT 4=OUTPUT
.SUBCKT INVERT 1 2 3 4
M1 4 3 1 1 PMOSE W=9.3U L=3U PD=42U PS=30U AD=71P AS=49P NRD=2.1 NRS=1.5
M2 4 3 2 2 NMOSE W=3U L=3U PD=24U PS=38U AD=37P AS=72P NRD=2.6 NRS=4.8
.ENDS INVERT
*_____
* Start of Main Circuits
*_____
VCC 1 0 DC 5
X1 1 0 2 3 INVERT
X2 1 0 3 4 INVERT
X3 1 0 4 5 INVERT
V1 2 0 PULSE(5 0 10NS 0 0 10NS 20NS)
*_____
* Start of Output Generation
*_____
.TRAN 0.1NS 50NS
.PRINT TRAN V(3) V(4) V(5)
.END
```

II. Invoke GSPICE simulation

We can run GSPICE simulation in either interactive mode or batch mode. If your circuit has more than ten transistors, it is recommend that you run SPICE as a batch job, because of the time required for simulation.

To run SPICE in an interactive mode, type

RUN GSPICE

The GSPICE program will prompt you for the SPICE input filename and two output filenames. Type the filenames as follows:

INPUT FILE: inverter.in
OUTPUT FILE: inverter.out
GRAPHIC OUTPUT FILE: inverter.dat

The Italic font indicates the computer prompt. The simulation of this simple circuit will take less than 1 minute to complete. After the complete of the simulation, two files, inverter.out and inverter.dat, are created in your current directory.

To run GSPICE as a batch job, we adopt the filename convention which assumes that the GSPICE input file has a file type of ".in", the output file to have a file type of ".out", and graphic output file of ".dat". Once this convention is established, running SPICE as a batch job is very easy; just type

BSPICE

The program will ask you to enter the input file name as follows:

Enter the input name: inverter

A batch job is then created and executed. After the batch job has finished, the program will beep once to indicate the completion of the simulation. As with the interactive program, two files, inverter.out and inverter.dat, are created in your current directory.

Alternatively, you can create the batch job and specify the input filename all on one command line as follows:

BSPICE inverter

III. Plot the Simulation Result

Since the ".dat" is created, we are ready to plot the simulation result by using a program called GRAPHIC. To run the GRAPHIC program, type

RUN GRAPHIC

The program will prompt you for input file name as follows:

Enter the input file name: inverter

The program will ask you to select a graphic setting or to take the default setting. We will take the default setting for the time being. We will show you how to change the graph setting later to customize your graphic output. The program will tell you how many different output plots there are in the input file and will ask you which two curves you want to plot. If you would like to see only one curve on the graph paper, you can select the same curve twice. The program offers you three choices; you can see one, two or all curves on the screen.

In this example of the inverter circuit, we select V(3) and V(4) to be plotted, as shown below. The screen will be cleared, and then the following dialog occurs.

Take the default graphic setting? (Y/N) Y

There are 3 curves in the input data.

1. *V(3)*
2. *V(4)*
3. *V(5)*
4. *All curves*

Which two or all curves you want? (1-4)

1st curve: 1

2nd curve: 2

A graph is plotted as illustrated in Fig 1.

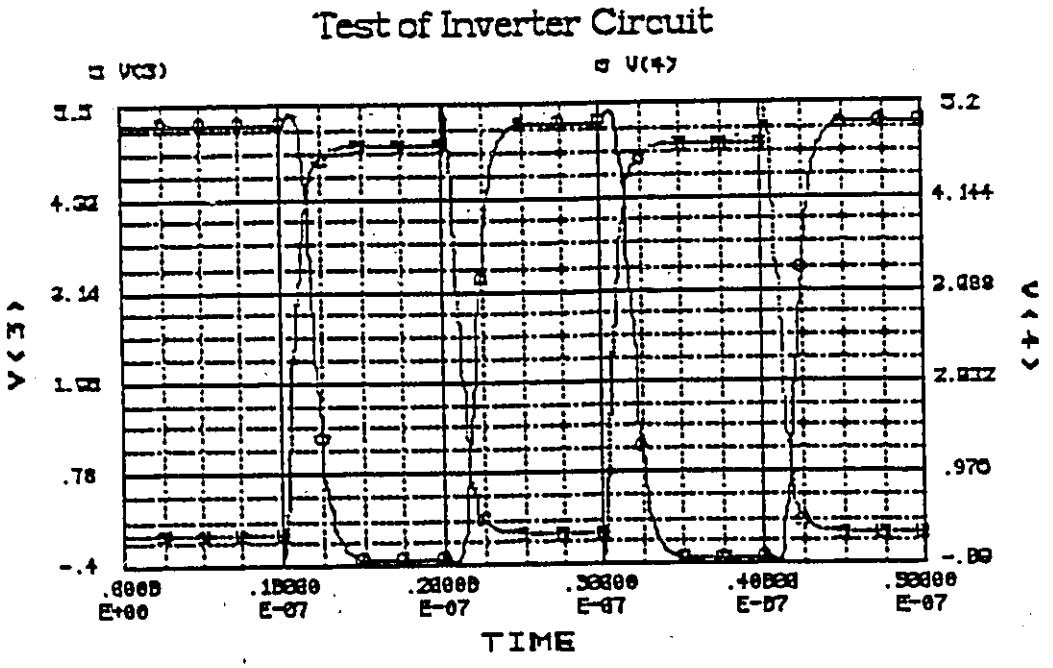


Fig 1 Graph of V(3) and V(4)

The graph consists of title, legend, title axes, markers and two curves which are drawn in two different colours, namely red and green on the colour monitor. The program also offers grid and shade which are set in the graphic output setting. The GRAPHIC program also incorporates a zoom allowing a more detail plot of the graph to be obtained.

Let us look at the zoom feature. After the graph has been completely drawn, hit the Z key to bring up the zoom function. A cross-hair cursor will be displayed at the lower left-hand corner of the graph paper. You can move and position the cursor by pressing the four ARROW keys, as shown in Fig 2.

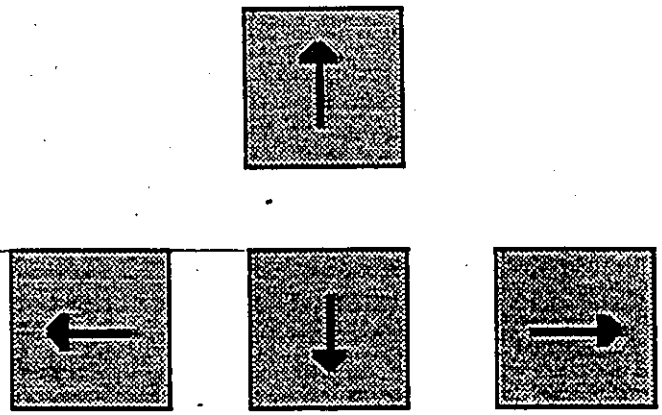


Fig 2 Arrow keys for controlling the cursor

The speed of moving the cross-hair cursor can be changed by pressing the PF3 or PF4 keys. Hitting the PF4 key causes the cursor to move faster whereas hitting the PF3 key makes the cursor slower. If you would like to see a portion of the graph in a full-screen scale, you position the cursor at a corner of that portion of the graph and hit any key except the Z key, C key or DELETE key. The cursor then disappears and a + symbol is drawn. Now you can define the zoom window by pressing the four ARROW keys again. A 'rubber-banding' rectangle is displayed as shown in Fig 3.

The increment value for the changing the size of the zooming window can also be adjusted by pressing the PF1, PF2, PF3 or PF4 keys. The increment value of PF4 is the greatest and the value of PF1 is the smallest one. After the zoom window has been

defined, you hit the Z key to zoom in. However, if you want to change the zoom window or start at a new location, hit the C key and then you see the cross-hair cursor again. You always use the cross-hair cursor to define the first corner of the window. In some cases, if you decide not to zoom in any part of the graph, hit the DELETE key to delete the zooming function.

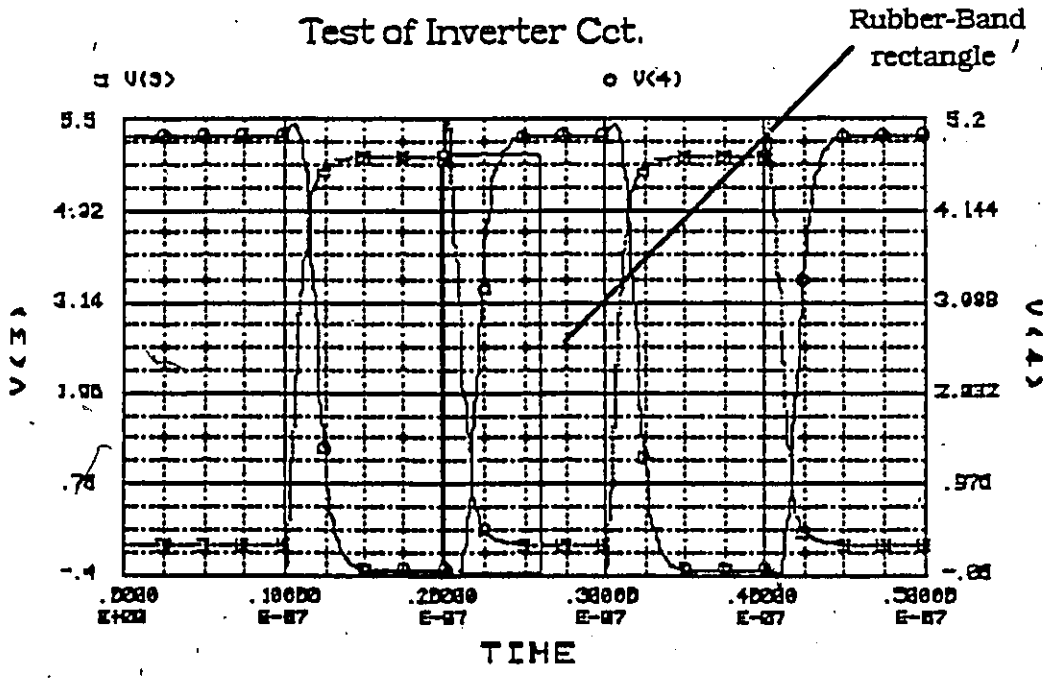


Fig 3 'Rubber-Banding' rectangle

When you have defined the zoom window, you hit the Z key. The content of the window is plotted in full, as shown Fig 4.

When you decide to leave the graphic window, hit the RETURN key once. The program will ask you whether you want to zoom out by selecting Y/N. If Y is entered, the graph of the Fig 1 is drawn again. If the N key is pressed, the program will ask you whether you want you want to stay with the same set of data by choosing Y/N. Entering Y will bring you back to Fig 1. If you hit the N key instead, you exit the GRAPHIC program. Since you have seen what happens when you select two different plots, we would like to show you now what will be drawn if you select to draw all plots on one graph. Using the example of the inverter circuit, we select 4 (All curves); you will see all three plots, as shown in Fig 5.

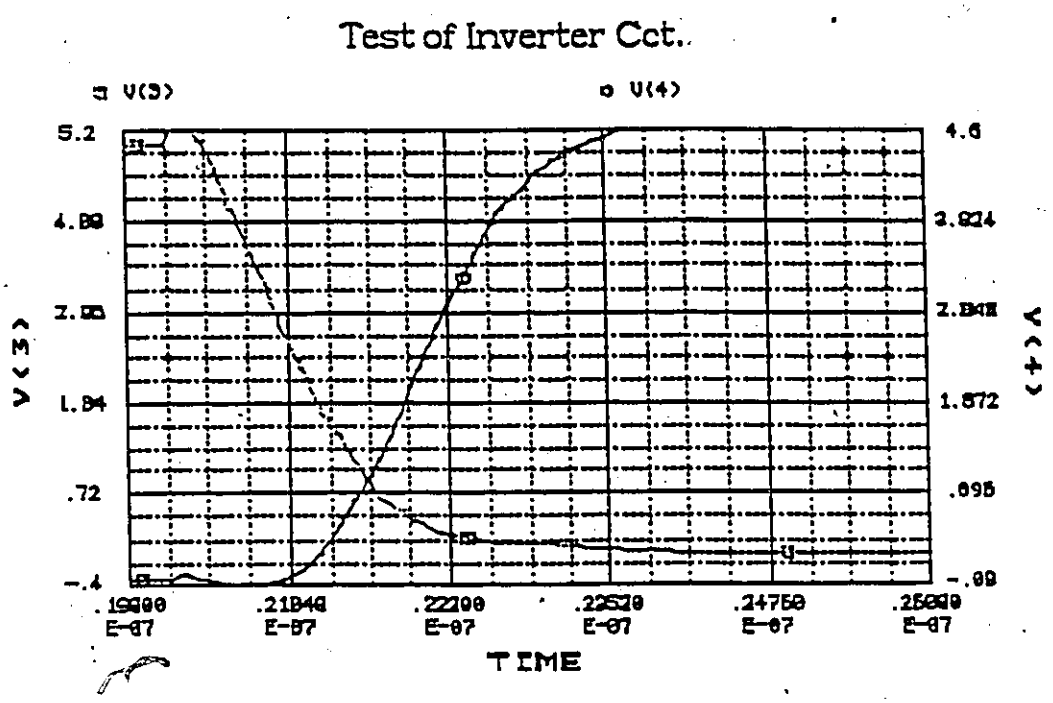


Fig 4 Zoomed plot of window

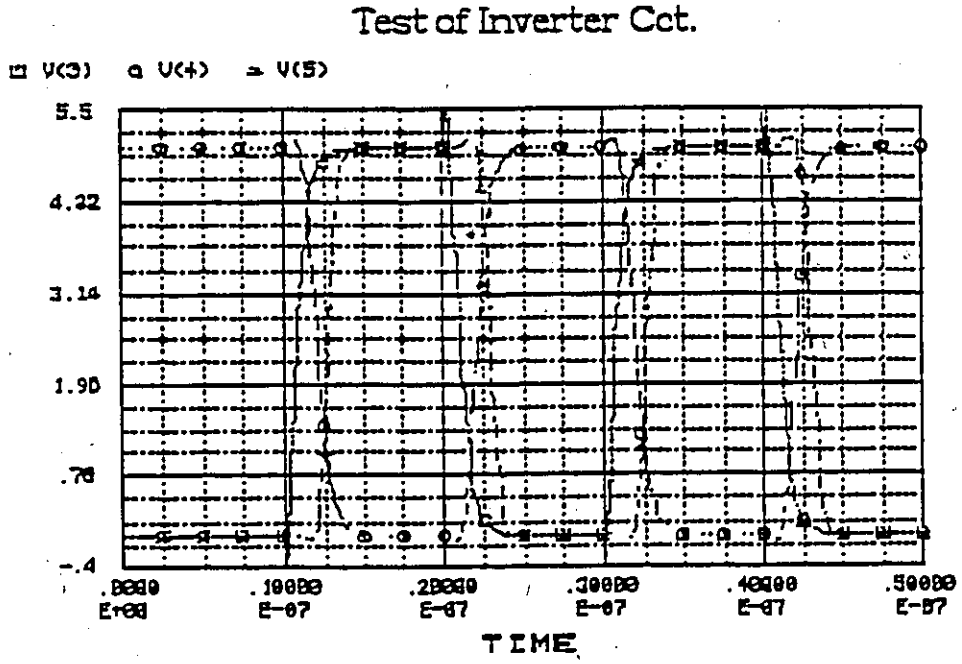


Fig 5 Complete graph with all 3 plots

Before we end this tutorial, we would like to show how to change the graphic setting for the graphical output. If you hit the N key, when you do not want to take the default graphic setting. the following dialog will occur. If you save the graphic setting, the graph will be drawn according to the new setting until you change the setting again.

Take default graphic setting? (Y/N) N
The following information will be required for the setting.
Grid? (Y/N) Y
Shade? (Y/N) N
Enter the number of cell on X-axis 5
Enter the number of subcell on X-axis 4
Enter the number of cell on Y-axis 5
Enter the number of subcell on Y-axis 4
Save the graphic setting? (Y/N) Y

It is much easier to quickly judge the result of the simulation graphically. The zoom feature allows close-up examination of portions of the result. The .PLOT output can also be consulted to obtain exact numbers. If the simulation is not satisfactory, the circuit description file has to be modified and the whole simulation process must be performed again until the design specifications are met.

Finally, the simulation process is completed, and the results of the simulation can be stored in a file to be used for future reference.

The GRAPHIC program has been used on a variety of graphical output, for example to generate plots for digital filter design. You can also prepare an input data file to generate your own drawings for reports or seminars. A similar program called TGRAPHIC has also been written to generate high quality graphic plots in many different colours and forms. The TGRAPHIC program is currently working on the Tektronix 4115 and Tektronix 4125 high resolution terminals in the VLSI design room. For further information, contact the author.

Alger W. Yeung
CAD/CAM Centre
September 1987

```
C~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
C       This program generates plot of the SPICE simulation results, ~
C       It works closely with the GSPICE program. Users can select a ~
C       number of options such as number of curves, zooming feature  ~
C       hardcopy output. The graphic setting can be changed as       ~
C       desired. The setting includes grid, shade, number of division~
C       , number of subdivision.                                     ~
C       The program can also work with other application programs,   ~
C       like filter design programs.                                 ~
C~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       Author: Alger W.K. Yeung                 C
C       Version: 2.01                            C
C       Date: January 1987                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        character*40 file_name
        character toggle,string*3
        common /data/ x(1000),y(1000,8),small_value(8),big_value(8),
     1  small_y,big_y,small_x,big_x,numfun,numpt
        character*8 y_title(8)
        common /name/ p_title,x_title,y_title
        character*72 p_title
        character*8 x_title
        real temp_y(1000)
        character*8 temp_title,string1
        character*40 a_title
        logical go_on,zoom
        common /graphic/ix_cell,ix_scell,iy_cell,iy_scell,ifill_pattern
     * ,ig_color,it_color,layout,num_curve,grid,shade,marker_type,
     * icurve,icurve2

c       common /mscel/ grid,ix_cell,ix_scell,iy_cell,iy_scell,icurve,
c     1 icurve2,shade
        logical grid
        data grid/.false./
        logical shade
        data shade/.false./

        call clear_text
        call move_to(1,1)
        goto 11
9       type 12
12      format(/,' File does not exist.'/)

11      type 10
10      format(' Enter the input file name: ',$)
        read 1,file_name
1       format(A)
c check the existence of the file name entered.
        call clear_text
        open(unit=7,file=file_name,status='old',err=9)
        close(unit=7,status='keep')
```

```
        open(unit=7,file=file_name,status='old')
        idefault=0

        read(7,20) p_title
        if (p_title.eq.'END') then
           type *,' No data is in the file!!!..... Bye.'
        end if
        if (p_title.eq.'END') goto 838
99      read(7,21) a_title
        read(7,21) dc_ac
        read(7,30) numpt
        read(7,30) numfun
        read(7,33) x_title,(y_title(j),j=1,numfun)

        do i=1,numpt
           read(7,40) x(i),(y(i,j),j=1,numfun)
        end do

20      format(A72)
21      format(A40)
30      format(I8)
33      format(T3,A8,T17,8(A8,4X))
40      format(T2,E10.3,4X,8(E10.3,2X))

c to find the min. and the max. values from the input data arrays.
        call set_nodebug
        call find_minmax(numpt,x,small_x,big_x)
        TYPE *,SMALL_X,BIG_X
        small_y=y(1,1)
        big_y=y(1,1)

        do j=1,numfun
           do i=1,numpt
                temp_y(i)=y(i,j)
           end do
           call find_minmax(numpt,temp_y,small_value(j),big_value(j))
           if (small_value(j).lt.small_y) then
                small_y=small_value(j)
           end if
           if (big_value(j).gt.big_y) then
                big_y=big_value(j)
           end if
        end do
        TYPE *,SMALL_Y,BIG_Y

80      call text_scroll(1,24)
        call move_to(4,1)

C TO TAKE THE DEFAULT VALUE FOR THE GRAPHICAL SETTING.

        CALL DEFAULT_DATA


905     type *,' '
```

```
      idefault=1
      icurve=1
      icurve2=2
      if (numfun.gt.2) then
        type 70,numfun
70      format(/,' There are ',I1,' curves in the input data.',//)

        do k=1,numfun
          type 123,k,y_title(k)
123       format(10x,i1,'. ',A8)
        end do
        k=numfun+1
        type 333,k
333     format(10x,i1,'. All curves',/)

299     type 75,numfun+1
75      format(/,' Which two or all curves you want? (1-',i1,')',$)

355     type 350
350     format(/,' 1st curve: ',$)
        call convert_real(sum,iok)
        if (iok.eq.1) goto 355
        icurve=sum
        iall=numfun+1
        if (icurve.eq.iall) goto 510

360     type 365
365     format(/,' 2nd curve: ',$)
        call convert_real(sum,iok)
        if (iok.eq.1) goto 360
        icurve2=sum
        if (icurve2.eq.iall) goto 510

        if ((icurve.lt.0).or.(icurve.gt.8).or.(icurve2.lt.0).or.
     *    (icurve2.gt.8)) goto 299

        if ((icurve.gt.k).or.(icurve2.gt.k)) goto 299
510     continue
      end if

      if ((icurve.eq.0).or.(icurve2.eq.0)) goto 199

      call init_graphics( )
      if ((icurve.eq.iall).or.(icurve2.eq.iall)) then
4561    izoom=0
        call plot_all(izoom)
        zoom=.false.
        go_on=.false.
c to hit key Z to zoom in any portion of the plotting area.

        call get_key(j)
        if ((j.eq.90).or.(j.eq.122)) then
c to get the coord. of the zooming area.
```

```
            call cursor(x1,y1,x2,y2,zoom)
          end if

          if (zoom) then
            call zoom_in_all(x1,y1,x2,y2,go_on)
          end if

c if the user wants to draw the original graph.
          if (go_on) goto 4561


        end if

        if ((icurve.eq.iall).or.(icurve2.eq.iall)) goto 580

456     izoom=0
        call plot_curve(izoom)
        zoom=.false.
        go_on=.false.
c to hit key Z to zoom in any portion of the plotting area.

        call get_key(j)
        if ((j.eq.90).or.(j.eq.122)) then
c to get the coord. of the zooming area.

            call cursor(x1,y1,x2,y2,zoom)
        end if

        if (zoom) then
          call zoom_in(x1,y1,x2,y2,go_on)
        end if

c if the user wants to draw the original graph.
        if (go_on) goto 456

580     if (numfun.gt.2) THEN
          call text_scroll(23,24)
          istay_on=0
255       type 230
230       format(' Stay? (Y/N) ',$)
          call get_key(i)
        if ((i.ne.89).and.(i.ne.121).and.(i.ne.78).and.(i.ne.110))
     *  goto 255
          if ((i.eq.89).or.(i.eq.121)) then
              istay_on=1
              call clear_text
              call clear_screen
          end if

        end if

        if (istay_on.eq.1) goto 80

199     read (7,20) string
```

```
      if (string.ne.'END') then
        call clear_text
        call clear_screen
       end if
       if (string.ne.'END') goto 99
838    call text_scroll(1,24)
       end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                              C
C to change the default value for graphic paper setting.                       C
C                                                                              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       subroutine default_data
       common /graphic/ix_cell,ix_scell,iy_cell,iy_scell,ifill_pattern
     *  ,ig_color,it_color,layout,num_curve,grid,shade,marker_type,
     *  icurve,icurve2
       logical grid,shade
       character g,s
       type 30
30     format(//,' Take the default graphic setting? (Y/N) ',$)

       goto 31
32     type *,char(7)
31     call get_key(i)
       if ((i.ne.121).and.(i.ne.89).and.(i.ne.78).and.(i.ne.110))
     *  goto 32

C hit 'Y'
       if ((i.eq.121).or.(i.eq.89)) then

          open(unit=20,file='default.def',status='old',err=70)
          read(20,100) g
          read(20,100) s
C         read(20,101) marker_type
          read(20,101) ix_cell
          read(20,101) ix_scell
          read(20,101) iy_cell
          read(20,101) iy_scell
C         read(20,102) ifill_pattern
C         read(20,101) ig_color
C         read(20,101) it_color
C         read(20,101) layout
C         read(20,101) num_curve
          close(unit=20)
          if (g.eq.'Y') then
             grid=.true.
          else
             grid=.false.
          end if
          if (s.eq.'Y') then
```

```
70            shade=.false.
C             marker_type=-1
              ix_cell=5
              iy_cell=5
              ix_scell=4
              iy_scell=4
C             ifill_pattern=8
C             ig_color=4
C             it_color=7
C             layout=2
C             num_curve=1
120           continue
        else


        type 45
45      format(/,
     *  ' The following information will be required for the setting.'/)

55      type 50
50      format(/,' Grid? (Y/N) ',$)
        call get_key(i)
        if ((i.ne.89).and.(i.ne.121).and.(i.ne.78).and.(i.ne.110))
     *  goto 55

        if ((i.eq.89).or.(i.eq.121)) then
           grid=.true.
           g='Y'
        else
           grid=.false.
           g='N'
        end if

545     type 540
540     format(/,' Shade? (Y/N) ',$)
        call get_key(i)
        if ((i.ne.89).and.(i.ne.121).and.(i.ne.78).and.(i.ne.110))
     *  goto 545

        if ((i.eq.89).or.(i.eq.121)) then
           shade=.true.
           s='Y'
        else
           shade=.false.
           s='N'
        end if

c enter the number of cells and subcells of the graph paper.
```

```
310     type 60
60      format(/,' Enter the number of cell on X-axis ',$)
        call convert_real(sum,iok)
        if (iok.eq.1) goto 310
        if (sum.lt.1.0) goto 310
        ix_cell=sum


315     type 320
320     format(/,' Enter the number of subcell on X-axis ',$)
        call convert_real(sum,iok)
        if (iok.eq.1) goto 315
        if (sum.lt.1.0) goto 315
        ix_scell=sum

325     type 65
65      format(/,' Enter the number of cell on Y-axis ',$)
        call convert_real(sum,iok)
        if (iok.eq.1) goto 325
        if (sum.lt.1.0) goto 325
        iy_cell=sum

330     type 335
335     format(/,' Enter the number of subcell on Y-axis ',$)
        call convert_real(sum,iok)
        if (iok.eq.1) goto 330
        if (sum.lt.1.0) goto 330
        iy_scell=sum



C390    type 395
C395    format(/,' Enter the number of curve/graph (1/2) ',$)
c       call convert_real(sum,iok)
c       if (iok.eq.1) goto 390
c       if ((sum.lt.1.0).or.(sum.gt.2.)) goto 390
c       num_curve=sum


202     type 201
201     format(/,' Save the graphic setting? (Y/N) ',$)
        call get_key(1)
        if ((l.ne.121).and.(l.ne.89).and.(l.ne.78).and.(l.ne.110))
     *  goto 202
        if ((l.eq.121).or.(l.eq.89)) then
           open(unit=20,file='default.def',status='new')
             write(20,100) g
             write(20,100) s
c            write(20,101) marker_type
             write(20,101) ix_cell
             write(20,101) ix_scell
             write(20,101) iy_cell
             write(20,101) iy_scell
c            write(20,102) ifill_pattern
```

```
c            write(20,101) ig_color
c            write(20,101) it_color
c            write(20,101) layout
c            write(20,101) num_curve
             close(unit=20)
          end if
          end if
100       format(A)
101       format(I2)
102       format(I3)
          return
          end



          subroutine plot_curve(izoom_in)
          common /data/ x(1000),y(1000,8),small_value(8),big_value(8),
     1    small_y,big_y,small_x,big_x,numfun,numpt
          character*8 y_title(8)
          common /name/ p_title,x_title,y_title
          character*72 p_title
          character*8 x_title
          real temp_y(1000)
          character*8 temp_title,string1
          character*40 a_title
          common /graphic/ix_cell,ix_scell,iy_cell,iy_scell,ifill_pattern
     *    ,ig_color,it_color,layout,num_curve,grid,shade,marker_type,
     *    icurve,icurve2
c         common /mscel/ grid,ix_cell,ix_scell,iy_cell,iy_scell,icurve,
c     1    icurve2,shade
          logical grid
          logical shade
          common /curve/ s_big_l,s_small_l,s_big_r,s_small_r,s_big_x,
     1    s_small_x
          call set_nodebug
          call clear_screen
          call clear_text
C         TYPE *,SMALL_X,BIG_X
c to draw the graph paper

          if (grid) then
           call dpaper('GLIN',ix_cell,ix_scell,'GLIN',iy_cell,iy_scell,
     *     'WHITE')
          else
           call dpaper('LIN',ix_cell,ix_scell,'LIN',iy_cell,iy_scell,
     *     'WHITE')
          end if

c to label the axis

          j=72
          i=1
```

```
      do while(p_title(i:i).eq.' ')
          i=i+1
      end do
      do while(p_title(j:j).eq.' ')
          j=j-1
      end do

      call lnaxis('XT',p_title(i:j),,,)

      j=8
      i=1
      do while(x_title(i:i).eq.' ')
          i=i+1
      end do
      do while(x_title(j:j).eq.' ')
          j=j-1
      end do

      if (izoom_in.eq.1) then
        call lnaxis('XB',x_title(i:j),s_small_x,s_big_x,.false.)
      else
        call lnaxis('XB',x_title(i:j),small_x,big_x,)
      end if

c to label the left y-axis for the 1st selected curve.

      temp_title=y_title(icurve)
      j=8
      i=1
      do while(temp_title(i:i).eq.' ')
          i=i+1
      end do
      do while(temp_title(j:j).eq.' ')
          j=j-1
      end do

      if (izoom_in.eq.1) then
        call lnaxis('YL',temp_title(i:j),s_small_1,s_big_1,.false.)
      else
        call lnaxis('YL',temp_title(i:j),small_value(icurve),
     *  big_value(icurve),.false.)
      end if

c     call lnaxis('YL',temp_title(i:j),,,.false.)
      legendx1=i
      legendy1=j
      string1=temp_title

c to label the right y-axis for the 2nd curve.

      if (numfun.eq.2) then
       temp_title=y_title(2)
       j=8
       i=1
```

```
        do while(temp_title(i:i).eq.' ')
           i=i+1
        end do
        do while(temp_title(j:j).eq.' ')
           j=j-1
        end do

        if (izoom_in.eq.1) then
         call lnaxis('YR',temp_title(i:j),s_small_r,s_big_r,.false.)
        else
         call lnaxis('YR',temp_title(i:j),small_value(2),
     *    big_value(2),.false.)
        end if

c       call lnaxis('YR',temp_title(i:j),,,,.false.)
        legendx2=i
        legendy2=j
       end if

c to label the right y-axis for more than 2 curves.

       if (numfun.gt.2) then
        temp_title=y_title(icurve2)
        j=8
        i=1
        do while(temp_title(i:i).eq.' ')
           i=i+1
        end do
        do while(temp_title(j:j).eq.' ')
           j=j-1
        end do

        if (izoom_in.eq.1) then
         call lnaxis('YR',temp_title(i:j),s_small_r,s_big_r,.false.)
        else
         call lnaxis('YR',temp_title(i:j),small_value(icurve2),
     *    big_value(icurve2),.false.)
        end if
c       call lnaxis('YR',temp_title(i:j),,,,.false.)
        legendx2=i
        legendy2=j
       end if

c to plot the curve against the left y-axis.

        i=icurve
        k=100+i
        do j=1,numpt
              temp_y(j)=y(j,i)
        end do
        call pdata(numpt,x,temp_y,'L','RED',k, , ,shade,)
        k=k-100
        call marker(k,-2.0,36.0)
        call move(0.0,37.0)
```

```
        call set_textsize(1,)
        call text(string1(legendx1:legendy1))

c to plot the curves against the right y-axis the 2nd curve.

        if (numfun.eq.2) then
         i=2
         k=100+i
         do j=1,numpt
               temp_y(j)=y(j,i)
         end do
         call pdata(numpt,x,temp_y,'R','GREEN',k, , ,shade,)
         k=k-100
         call marker(k,37.0,36.0)
         call move(39.0,37.0)
         call set_textsize(1,)
         call text(temp_title(legendx2:legendy2))

        end if

c to plot the curve against the right y-axis for more than 2 curves.

        if (numfun.gt.2) then
         i=icurve2
         k=100+i
         do j=1,numpt
               temp_y(j)=y(j,i)
         end do
         call pdata(numpt,x,temp_y,'R','GREEN',k, , ,shade,)
         k=k-100
         call marker(k,37.0,36.0)
         call move(39.0,37.0)
         call set_textsize(1,)
         call text(temp_title(legendx2:legendy2))

        end if

        return
        end



        subroutine plot_all(izoom_in)
        common /data/ x(1000),y(1000,8),small_value(8),big_value(8),
     1  small_y,big_y,small_x,big_x,numfun,numpt
        character*8 y_title(8)
        common /name/ p_title,x_title,y_title
        character*72 p_title
        character*8 x_title
        real temp_y(1000)
        character*8 temp_title,string1
        character*40 a_title
        character*5 color_var
```

```
      common /graphic/ix_cell,ix_scell,iy_cell,iy_scell,ifill_pattern
     *  ,ig_color,it_color,layout,num_curve,grid,shade,marker_type,
     *  icurve,icurve2

c     common /mscel/ grid,ix_cell,ix_scell,iy_cell,iy_scell,icurve,
c    1 icurve2,shade
      logical grid,color,shade
      common /curve/ s_big_l,s_small_l,s_big_r,s_small_r,s_big_x,
     1  s_small_x
      data color/.false./
      call set_nodebug
      call clear_screen
      call clear_text

c to draw the graph paper

      if (grid) then
       call dpaper('GLIN',ix_cell,ix_scell,'GLIN',iy_cell,iy_scell,
     *  'WHITE')
      else
       call dpaper('LIN',ix_cell,ix_scell,'LIN',iy_cell,iy_scell,
     *  'WHITE')
      end if

c to label the axis

      j=72
      i=1
      do while(p_title(i:i).eq.' ')
          i=i+1
      end do
      do while(p_title(j:j).eq.' ')
          j=j-1
      end do

      call lnaxis('XT',p_title(i:j),,,)

      j=8
      i=1
      do while(x_title(i:i).eq.' ')
          i=i+1
      end do
      do while(x_title(j:j).eq.' ')
          j=j-1
      end do

      if (izoom_in.eq.1) then
        call lnaxis('XB',x_title(i:j),s_small_x,s_big_x,.false.)
      else
        call lnaxis('XB',x_title(i:j),small_x,big_x,.false.)
      end if

      if (izoom_in.eq.1) then
```

```
          call lnaxis('YL',' ',s_small_1,s_big_1,.false.)
        else
          call lnaxis('YL',' ',small_y,big_y,.false.)
        end if

c to plot the curve against the left y-axis.

        do i=1,numfun
          k=100+i
          do j=1,numpt
               temp_y(j)=y(j,i)
          end do
          color=.not.color
          if (color) then
            color_var='RED   '
          else
            color_var='GREEN'
          end if

          call pdata(numpt,x,temp_y,'L',color_var,k,i, ,shade,)
        end do
        call set_linepattern(1,)

c to draw the legend of the curves.

        call set_textsize(1,)
        xmarker=-8.0
        xmarker2=-6.0
        k=2
        do l=1,numfun
           stringl=y_title(l)
           jj=8
           ii=1
           do while(stringl(jj:jj).eq.' ')
                jj=jj-1
           end do
           do while(stringl(ii:ii).eq.' ')
                ii=ii-1
           end do
           call set_color(' ',k)
           k=k+1
           if (k.eq.4) then
             k=2
           end if
           call marker(1,xmarker,36.0)
           call move(xmarker2,37.0)
           call text(stringl(ii:jj))
           xmarker2=xmarker2+9.0
           xmarker=xmarker+9.0
        end do


        return
```

```
      end

      subroutine zoom_in(x1,y1,x2,y2,next)
      common /data/ x(1000),y(1000,8),small_value(8),big_value(8),
     1 small_y,big_y,small_x,big_x,numfun,numpt
      character*8 y_title(8)
      common /name/ p_title,x_title,y_title
      character*72 p_title
      character*8 x_title
      common /graphic/ix_cell,ix_scell,iy_cell,iy_scell,ifill_pattern
     * ,ig_color,it_color,layout,num_curve,grid,shade,marker_type,
     * icurve,icurve2

c     common /mscel/ grid,ix_cell,ix_scell,iy_cell,iy_scell,icurve,
c    1 icurve2
      logical grid,next
      common /curve/ s_big_1,s_small_1,s_big_r,s_small_r,s_big_x,
     1 s_small_x

c use the ratio to find the value of y and x at different coord.
      t_length_1=big_value(icurve)-small_value(icurve)
      yy1=t_length_1/33.0*y1
      yy2=t_length_1/33.0*y2

      yy1=yy1+small_value(icurve)
      yy2=yy2+small_value(icurve)

      t_length_r=big_value(icurve2)-small_value(icurve2)
      yy3=t_length_r/33.0*y1
      yy4=t_length_r/33.0*y2

      yy3=yy3+small_value(icurve2)
      yy4=yy4+small_value(icurve2)


      t_length_x=big_x-small_x
      xx1=t_length_x/61.0*x1
      xx2=t_length_x/61.0*x2

      xx1=xx1+small_x
      xx2=xx2+small_x

      if (yy1.gt.yy2) then
        s_big_1=yy1
        s_small_1=yy2
      else
        s_big_1=yy2
        s_small_1=yy1
      end if

      if (yy3.gt.yy4) then
        s_big_r=yy3
        s_small_r=yy4
      else
```

```
          s_big_r=yy4
          s_small_r=yy3
        end if

        if (xx1.gt.xx2) then
          s_big_x=xx1
          s_small_x=xx2
        else
          s_big_x=xx2
          s_small_x=xx1
        end if

c to plot the curves within the range

        izoom=1
        call plot_curve(izoom)

        call text_scroll(23,24)

        accept *
c to check whether the user wants to go back the original graph.

25      type 22
22      format(' Zoom back? (Y/N) ',$)
        call get_key(j)
        if ((j.ne.89).and.(j.ne.121).and.(j.ne.110).and.(j.ne.78))
     *  goto 25

        if ((j.eq.89).or.(j.eq.121)) then
           next=.true.
        else
           next=.false.
        end if

        return
        end


        subroutine zoom_in_all(x1,y1,x2,y2,next)
        common /data/ x(1000),y(1000,8),small_value(8),big_value(8),
     1  small_y,big_y,small_x,big_x,numfun,numpt
        character*8 y_title(8)
        common /name/ p_title,x_title,y_title
        character*72 p_title
        character*8 x_title
        common /graphic/ix_cell,ix_scell,iy_cell,iy_scell,ifill_pattern
     *  ,ig_color,it_color,layout,num_curve,grid,shade,marker_type,
     *  icurve,icurve2

c       common /mscel/ grid,ix_cell,ix_scell,iy_cell,iy_scell,icurve,
c     1 icurve2
        logical grid,next
        common /curve/ s_big_1,s_small_1,s_big_r,s_small_r,s_big_x,
     1  s_small_x
```

```
c use the ratio to find the value of y and x at different coord.
        t_length_1=big_y-small_y
        yy1=t_length_1/33.0*y1
        yy2=t_length_1/33.0*y2

        yy1=yy1+small_y
        yy2=yy2+small_y


        t_length_x=big_x-small_x
        xx1=t_length_x/61.0*x1
        xx2=t_length_x/61.0*x2

        xx1=xx1+small_x
        xx2=xx2+small_x

        if (yy1.gt.yy2) then
          s_big_1=yy1
          s_small_1=yy2
        else
          s_big_1=yy2
          s_small_1=yy1
        end if


        if (xx1.gt.xx2) then
          s_big_x=xx1
          s_small_x=xx2
        else
          s_big_x=xx2
          s_small_x=xx1
        end if

c to plot the curves within the range

        izoom=1
        call plot_all(izoom)

        call text_scroll(23,24)

        accept *
c to check whether the user wants to go back the original graph.

25      type 22
22      format(' Zoom back? (Y/N) ',$)
        call get_key(j)
        if ((j.ne.89).and.(j.ne.121).and.(j.ne.110).and.(j.ne.78))
     *  goto 25

        if ((j.eq.89).or.(j.eq.121)) then
           next=.true.
        else
           next=.false.
```

```
        end if

        return
        end


        subroutine cursor(x,y,xl,yl,zoom)
        byte key
        logical zoom

        zoom=.false.

        call set_writemode('CO')
        goto 10
13      call marker(4,x,y)

10      call locate(x,y,key)
        if(key.eq.'177) goto 200
        if ((x.lt.0.0).or.(x.gt.61.).or.(y.lt.0.).or.(y.gt.33.))
     *   goto 10
        call marker(4,x,y)
        if ((key.eq.'C').or.(key.eq.'c')) goto 13
        xl=x
        yl=y
        call box(x,y,xl,yl)
        change=2.0
51      call get_key(key_code)
        call move_to(1,1)
        tx=xl
        ty=yl
        if (key_code.eq.127) goto 200
        if ((key_code.eq.67).or.(key_code.eq.99)) then
           call box(x,y,xl,yl)
        end if
        if ((key_code.eq.67).or.(key_code.eq.99)) goto 13

        if ((key_code.eq.99).or.(key_code.eq.122)) then
           i=key_code
        end if
        if ((key_code.eq.99).or.(key_code.eq.122)) goto 87

        if (key_code.eq.256) then
           change=0.15
        else if (key_code.eq.257) then
           change=1.0
        else if (key_code.eq.258) then
           change=2.5
        else if (key_code.eq.259) then
           change=5.0
        end if
        if (.not.((xl.lt.0.0).or.(xl.gt.61.).or.(yl.lt.0.).or.
     *  (yl.gt.33.))) then
c to check the arrow cursor movement.
         if (key_code.eq.274) then
```

```
          yl=yl+change
         else if (key_code.eq.275) then
          yl=yl-change
         else if (key_code.eq.276) then
          xl=xl-change
         else if (key_code.eq.277) then
          xl=xl+change
         end if
        if ((xl.lt.0.0).or.(xl.gt.61.).or.(yl.lt.0.).or.(yl.gt.33.))
     *  then
          xl=tx
          yl=ty
        end if
         call box(x,y,tx,ty)
         call box(x,y,xl,yl)
        end if
        i=key_code
        if (((i.ge.274).and.(i.le.277)).or.(i.eq.258).or.
     *  (i.eq.259).or.(i.eq.256).or.(i.eq.257)) goto 51

c       call set_writeMode('CO')
        call marker(4,x,y)
c       call box(x,y,xl,yl)

30      call get_key(i)
        call move_to(1,1)
c to change the zooming area.
        if ((i.eq.67).or.(i.eq.99)) then
           call box(x,y,xl,yl)
        end if

        if ((i.eq.67).or.(i.eq.99)) goto 10

        if (i.eq.127) then
           call box(x,y,xl,yl)
           call set_writemode('OV')
           zoom=.false.
        end if

        if (i.eq.127) goto 200

87      if ((i.eq.90).or.(i.eq.122)) then
           call box(x,y,xl,yl)
           call set_writemode('OV')
           zoom=.true.
        else
           type *,char(7)
           zoom=.false.
        end if


        if (.not.zoom) goto 30
```

```
200     return
        end



cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                       c
c       To convert a text string to numeric value.                      c
c                                                                       c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

        subroutine convert_real(sum,icheck)
        character*80 string
        icheck=1
        read 1,string
1       format(A)
        if (string(1:1).eq.' ') goto 2
        len=80
        do while(string(len:len).eq.' ')
                len=len-1
        end do
        iicheck=0
        do i=1,len
        ik=ichar(string(i:i))
        if ((ik.lt.45).or.(ik.gt.57).or.(ik.eq.47)) then
           iicheck=1
        end if
        end do
        if (iicheck.eq.1) goto 2
        ii=1
        sum=0.0
        do i=len,2,-1
        if (string(i:i).eq.'.') then
            sum=sum/ii
            ii=1
        else
            temp=ii*(ichar(string(i:i))-48)
            sum=sum+temp
            ii=ii*10
        end if
        end do
        if (string(1:1).eq.'-') then
           sum=-sum
        else
           sum=sum+ii*(ichar(string(1:1))-48)
        end if
        icheck=0

2       return
        end
```

```
c  ************************************************************************
c
c       GET_KEY
c               Returns the integer value of a keystroke.
c               Will accept shifted keys and all control keys
c               except backspace, which it will ignore.
c
c  ************************************************************************
c
c    INTERNAL:
c       text            - not used
c       term_set        - terminator set mask for smg$read_string
c       data_str        - dummy variable for smg$read_string
c       enable          - .true. if virtual keyboard is enabled
c       status          - error warning for smg$ routines
c
c    OUTPUT:
c       ii              - key press returned to calling program
c
c  ************************************************************************

        subroutine get_key(ii)

c       calling sequence:       call get_key(ii)

        implicit integer*4 (s)
        include '($trmdef)'
        include '($smgdef)'
        character*16 term_set
        character*20 data_str
        logical enabled

        data enabled/.false./
        data term_set/''/

        smax=1

10      if(enabled) then
                status=smg$read_string(keyboardl,
     1           data_str, ,smax,trm$m_tm_noecho.or.trm$m_tm_noedit,
     1            , , ,sterm_char)


                if(.not.status) call lib$stop(%val(status))
                if (ichar(data_str).eq.32) ii=sterm_char
                if (sterm_char.eq.510) ii=ichar(data_str)
                return
        end if

c set cursor key to RESET
c       type *,''
```

```
c create virtual keyboard

        status = smg$create_virtual_keyboard(keyboardl)
        if(.not.status) call lib$stop(%val(status))
        enabled = .true.
        go to 10
        end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C MENU CREATER PROGRAM-VERSION M2                                      C
C DEPARTMENT OF ELECTRICAL ENGINEERING                                 C
C UNIVERSITY OF WINDSOR                                                C
C WRITTEN IN FORTRAN-77                                                C
C INDEPENDENT SUBROUTINES                                              C
c new program which echoes the selection                               C
C SENT BY RAJA TO TEKALIKE ON DR.MILLER'S REQUEST                      C
C FOR THE EXCLUSIVE USE OF MR.ALGER YEUNG                              C
C EXTRACTED FROM U.OF.W. TIME SERIES ANALYSIS PACKAGE WITH             C
C THE PERMISSION OF RAJA AND DR.MILLER                                 C
C AUTHORIZED                                                           C
C-----------------------------------------------------------------
C This subroutine prints the text starting at pos row=ix,col=iy
C-----------------------------------------------------------------
        subroutine print_at(ix,iy,text)
        integer ix,iy
        character*(*) text
        character*3 bufx,bufy

        write(bufx,'(i2)')ix
        if(bufx(1:1).eq.' ') bufx=bufx(2:)
        write(bufy,'(i2)')iy
        if(bufy(1:1).eq.' ') bufy=bufy(2:)

        write(*,'(a)') char(0)//char(155)
        1 //bufx(1:index(bufx,' ')-1)//';'
        1 //bufy(1:index(bufy,' ')-1)//'H'//text

        return
        end

        subroutine tcom(text)
        character*(*) text
        write(*,'(a)') char(0)//text
        return
        end

        subroutine move_to(ix,iy)
        character*3 bufx,bufy

        write(bufx,'(i2)')ix
        if(bufx(1:1).eq.' ') bufx=bufx(2:)
        write(bufy,'(i2)')iy
        if(bufy(1:1).eq.' ') bufy=bufy(2:)
```

```
      write(*,'(a)') char(0)//char(155)
     1 //bufx(1:index(bufx,' ')-1)//';'
     1 //bufy(1:index(bufy,' ')-1)//'H'

      return
      end
```

Appendix V

Macro Function Facility

Appendix V

Macro Functions

One of the interesting and useful features of Electric design system is the **macro** facility which allows users to customize the user-interface environment. It involves creating new commands to tailor the user commands precisely. A macro is new, user-defined command that contains a collection of other Electric commands. This appendix will describe the essential commands for defining macro functions. The commands include **macbegin**, **macend**, **var**, **echo**, **terminal input** and **if**, and these commands can be combined to form a powerful user-command facility. A number of macro functions have been written for speeding up the mask layout and simulation processes, and are listed at the end of this appendix.

The **macbegin** command starts the definition of a macro function. If a name is given, a new macro is created. Otherwise, an old macro is redefined. All commands issued after this command until a **macend** command are remembered as part of the macro. Examples can be found at the end of this appendix. The **macro** command can be invoked by hitting a single character **%**. Other macros are invoked by simply typing the macro name. There are two options to **macbegin** that may follow the macro name. If **no-execute** option is used after the macro name, the macro will not be executed while the macro is defined. This option is useful when the macro is defined in a command file as the examples in this appendix.

The other option to macbegin is **verbose**, which requests that each command of the macro be displayed when the macro is invoked.

One of powerful feature of macro facility is that macros can be parameterized. The use of the construct **%1** in any of the macro commands means substitute the first parameter of the macro at this point. Since macros have no parameters when they are first being defined, an initial and default value can be supplied for these parameter constructs by placing the value in square brackets following the parameter declaration.

The **var** command is used to create and to manipulate variables, both within the Electric internal database and locally for use in macros. This command is very powerful as it is able to examine any Electric internal variables. Hence, this command should be used with caution. Three special characters, **$**, **%** and **~**, are normally associated with this command. If the **$** appears in an Electric command, it references to an internal database variable. Following the $ comes a reference to an object, followed by qualifiers of that object. The **~** sign references the currently selected object. For instance, the x and y coordinates of the cursor can be accessed using both **$** and **~** signs such as **$~x** and **$~y**. There are many possible object references such as the technology variables, node attributes, and library information. To describe all the possible qualifiers available in the database is beyond the scope of this appendix. A complete list is provided by Part I of the Electric internals manual. On the other hand, users can define their own user-variables. To address this need, there is a set of 52 command-interpreter variables

with single letters a through z and A through Z. Each is referenced with a % in front.

The function of **var** command is to examine, set, and manipulate variables. The **var examine** takes a single variable specification and prints its value. The **var delete** takes a variable specification and removes it from the variable list. The **var set** creates and changes variables. For instance, to set variable **a** to 17, type

var set a 17

Besides setting variables, we can performed arithmetic by using +, -, *, /, and **mod** in place of the word **set**. Bit-wise operations can also be achieved with **and** and **or** operators String concatenation can be done with | operator. For example, if we wish to append time to clock for a node's name, type

var set a clock

var / a time

var / $~.node_name a redisplay

Some macros can be structured to communicate with users. For output purpose, there is an **echo** command, which simply repeats its parameters on the status display window. For example, to display the current the XY coordinates of the cursor, use the followings:

echo Cursor at $~x, $~y

For input purpose, the **terminal input** command takes two parameters: a variable letter and a prompt string. For instance,

terminal input c Port name:

prompts with the message Port name: and then places the typed response into the variable %c.

For more control of macro execution, there is the **if** command. The **if** command requires four parameters: a first value, a comparison, a second value, and a command to execute if the comparison succeeds. The two values can be integers or strings, and the comparison is one of the "C" language conditions, ==, !=, >,>=,<,<=. For example,

. if $~.proto == .transistor echo It is a transistor

If the selected object is a transistor, the message will be echoed.

Combined with all commands discussed above, powerful and flexible macro functions can be created. For simplifying the mask layout and the SPICE deck creation processes, a number of macro functions have been created and listed below. Functions include measuring a distance between two points, reading file into database, creating MOS transistors with various length and width in logic technology, and setting up the SPICE input file. A number of other existing macro files are located in the */users/local/electric/lib* directory in the workstation.

```
*** Macro functions created for customizing user environment ***
macbegin simplelayer no-execute
visiblelayers abdehiylmz
macend

macbegin showlayer no-execute
visiblelayers %1
macend

macbegin xy no-execute
echo .Cursor at $(~x),$(~y)
macend
```

```
macbegin printport no-execute
echo Port'name is $~p
macend

macbegin distance no-execute
var set c $(~x)
var set d $(~y)
var set e $tech:~.deflambda
var / c %e
var / d %e
echo Cursor at %c, %d
var - a %c
var - b %d
echo deltaX is %a,deltaY is %b
var set a %c
var set b %d
macend

macbegin cifin no-execute
offaid drc
library read %1 cif
library use %1
macend


macbegin spiceplot no-execute
tellaid simulation spice parse-output %1
macend


macbegin spiceresist no-execute
tellaid simulation spice resistance-toggle
macend

macbegin usefile no-execute
library read %1
library use %1
show cell
macend

macbegin spicecom
command spice.mac
macend
```

```
macbegin spicedc no-execute
var set a "DC "
var | a %1[5]
setsource v %a
macend

macbegin spicemeter no-execute
var set a "("
var | a %1[0]
var | a ","
var | a %2[5]
var | a ")"
setmeter %a
macend

macbegin spicepulse no-execute
var set a "PULSE("
var | a %1
var | a " "
var | a %2
var | a " "
var | a %3
var | a "NS "
var | a %4
var | a "NS "
var | a %5
var | a "NS "
var | a %6
var | a "NS "
var | a %7
var | a "NS)"
setsource v %a
macend

macbegin spicetrans no-execute
var set a " "
var | a %1
var | a "NS "
var | a %2
var | a "NS"
setsource t %a
macend
```

```
macbegin pmos no-execute
var set a "pmos"
var | a %1[3]
var | a "/"
var | a %2[3]
echo %a
settrans %a
macend

macbegin nmos no-execute
var set a "nmos"
var | a %1[3]
var | a "/"
var | a %2[3]
echo %a
settrans %a
macend
```

Appendix VI

Plotting Facilities

Appendix VI

Plotting Facilities

This appendix describes how to use the plotting programs developed in this research project. The plotting facility consists of four programs: **plot.c**, **vt2x.c**, **splot.c**, and **con_post.c**. The program listings are attached at the end of this appendix. The **plot** program is used to display the **Relax** simulation results on the GPX workstation. The *vt2x* converts the data format of regular SPICE output files into a format supported by the **plot** program. The **splot** is used to generate hardcopy on the Apple LaserWriter Plus printer and the HP-7585 colour plotter. The **con_post** simplifies PostScript files generated by the **splot** program since some PostScript files involving complicated graphic patterns require a large memory to be processed and the Apple laser printer has limited memory.

The **plot** program allows users to select window sizes for display, plotting range, and number of curves. The desired window size is obtained by using the mouse, as follows:

* Press the left mouse button to create a 200 by 200 pixels window at the cursor position.

* Press the right mouse button to create a 200 by 200 pixels window at the centre of the screen.

* Press the centre mouse button to create a window that we can choose the height and width. Move the mouse horizontally and vertically until the size indicator located at the upper left-hand

corner of the screen displays the desired height and width. Release the mouse button and the window appears on the screen.

For example, three files, node1, node2 and node3, are generated by the **Relax2** simulator by typing the command with a Relax input file, as follows:

% **relax2** input-file

These files representing the simulation results can be displayed on the GPX workstation by invoking the plotting program as follows:

% **plot** node1 node2 node3

The **plot** program determines the maximum and minimum values of the curve, node1, and then the user can specify the plotting range. Then, the user uses the mouse to create a graphic window with appropriate window size. This process is repeated until all curves have been displayed.

The **vt2x** converts a regular SPICE output file into a set of new files which can be plotted by using the **plot** program. The **vt2x** program is based on the fact that carriage control characters X and Y are used to start and end a table created by the *.PRINT* command in SPICE input file. For instance, if a SPICE output file tabulates three curves: V(1), V(2) and V(3), the **vt2x** is then invoked as shown in the following to create three different files: *V1*, *V2* and *V3*. Then, the **plot** can be used to display the curves on the screen.

% **vt2x** spice-output-file

The **splot** program generates hardcopy of mask layout and schematic of circuit design on a number of hardcopy devices: The program originally came with the Electric design system. However, one or more modules were missed so that a lot of modifications must

be done to the program in order to generate hardcopy on the Apple. laser printer and the HP-7585 colour plotter. Since most PostScript files generated by the **splot** program are huge, the files cannot be handled efficiently by the Apple LaserWriter Plus printer, especially when graphic patterns are used extensively for drawing mask layout. Hence, a program called **con_post**, was written to simplify the files. More information about **splot** can be learned by simply typing **splot** on the terminal. The **con_post** is invoked by typing as follows:

% **con_post** input-file output-file

All plotting programs are written in C language and the listings are shown in the following.

```
/* ------------------------------------------------------------
  This plotting program is primarily designed to display
  Relax2 output file. The program can display more than one file
  at a time. All the file names are specified on the command line.

  Written by Alger Yeung
  Date: October 31, 1987
  Used on X-window environment
  ------------------------------------------------------------*/

#include <stdio.h>
#include <X/Xlib.h>

#undef CURSOR /* to remove clash with Xlib.h */

#define GMINSIZE        100     /* minimum display in pixels    */
#define MAXENTRY       2000     /* max entry in the array   */
/* color constants */
#define GMAXCOLORS      254     /* NOTE: this is 254, not 256   */
#define GMAXPLANES      8       /* device dependant             */

/* button identifiers */
#define Grightbutton    0
#define Gmiddlebutton   1
#define Gleftbutton     2


struct {
    FontInfo *font;
    char *fontname;
    int   width;
} gra_font[] =
{
    (FontInfo *)0, "6x10", 6,   /* 0 */
    (FontInfo *)0, "6x13", 6,   /* 1 */
    (FontInfo *)0, "8x13", 8,   /* 2 */
    (FontInfo *)0, "9x15", 9,   /* 3 */
    (FontInfo *)0, 0, 0
};

/* array that associates GRAPHICS font sizes with the above fonts */
static gra_fontassociate[9] =
{
    0,          /* TXTVTINY */
    0,          /* TXTTINY */
    1,          /* TXTVSMALL */
    1,          /* TXTSMALL */
    2,          /* TXTMEDIUM */
    2,          /* TXTLARGE */
    3,          /* TXTVLARGE */
    3,          /* TXTHUGE */
    3           /* TXTVHUGE */
};
Window w;
```

```
int sc_width,sc_height;
FontInfo *gra_curfont;

main(argc, argv)

int argc;
char *argv[];

{
   FILE *infile, *outfile;
   double time, voltage;
   double maxx, minx, maxy, miny;
   int numpt;
   int i;
   short sx,sy;
   double originx, originy;
   double gdistance, realxdist, realydist;
   double x[MAXENTRY], y[MAXENTRY], mapx[MAXENTRY], mapy[MAXENTRY];
   char eof,display[128];
   short ox,oy,gra_width, gra_height, num_dividx, num_dividy;

/* window variables */
   Vertex vlist[MAXENTRY];
   OpaqueFrame frame;
/*
* routine to initialize the device in 'us_display'
*/
    /* default number,              0    1    2    3    4    5    6    7 */
    /* default colors             bla  whi  red  grn  blu  yel  mag  cya */
    static int normalred[8]   = {  0, 255, 255,   0,   0, 255, 255,   0};
    static int normalgreen[8] = {  0, 255,   0, 255,   0, 255,   0, 255};
    static int normalblue[8]  = {  0, 255,   0,   0, 255,   0, 255, 255};
   int  plane, xpixels[GMAXCOLORS];
   int color, numfile;
   char default_geometry[32], normalletter[8];
   char ch, filename[40];
   WindowInfo info;
   int gra_dev_no, gra_sdep;
   int gra_maxcolors;


   for (numfile = 1; numfile < (argc); numfile++)
   {


/* check the existence of the file */
   if ((infile = fopen(argv[numfile], 'r')) == NULL)
     {
       fprintf(stderr, 'File %s cannot be opended.\n', argv[1]);
       fprintf(stderr, 'Usage: plot filename(s)\n');
       exit(1);
     }
/* read in the data from the file and set up the array */
   numpt = 0;
```

```
   do {
     numpt++;
     fscanf(infile, "%e", &time);
     fscanf(infile,"%e" , &voltage);
     x[numpt] = time; y[numpt] = voltage;
/*     printf("numbr= %d,time=%e voltage=%e.\n", numpt, time, voltage); */
   } while ( getc(infile) != EOF);
     numpt--;
     maxmin(x, numpt, &maxx, &minx);
     maxmin(y, numpt, &maxy, &miny);
     printf("MaxY= %lf, MinY= %lf \n", maxy, miny);
     printf("Enter the desired Y range: maxy--miny\n");
     scanf("%f", &maxy);
     scanf("%f", &miny);

   fclose(infile);


/* setup the window parameters */
/*   if (XOpenDisplay(argc ? argv[1] : " ") == NULL)*/
   if (XOpenDisplay("unix:0") == NULL)
     fprintf(stderr, "Could not open Display!0");
   frame.bdrwidth = 1;
   frame.border = WhitePixmap;
   frame.background = BlackPixmap;
   strcpy(default_geometry, "=412x412+200+200");
   w = XCreate("PLOT", argv[0] ,"", default_geometry,
         &frame, 100, 100);
   XSelectInput(w, ExposeRegion);
   sc_width = frame.width;
   sc_height = frame.height;
   XMapWindow(w);

/* set up the graph origin and it's length */
   realxdist =  (0.8) * (double) sc_width;
   realydist =  (0.8) * (double) sc_height;
   originx = (double) sc_width - realxdist -30;
   originy = realydist +30;
/* set up the color and display planes 8 */
   gra_sdep = 8;

    if (numfile == 1)
    {
    /* get depth of screen */
    if (gra_sdep > 1)
    {
        if (gra_sdep >= GMAXPLANES)
        {
            gra_sdep = GMAXPLANES;
            if (XGetColorCells(0, GMAXCOLORS-2, 0, &plane, xpixels) == 0)
                fprintf(stderr,"Can't get colors\n");
            gra_maxcolors = GMAXCOLORS;
        } else
```

```
        {
            if (XGetColorCells(0, (1<<gra_sdep)-2, 0, &plane, xpixels) == 0)
                fprintf(stderr,"Can't get colors\n");
            if (xpixels[0] != 2) printf("Color map in wrong place\n");
            gra_maxcolors = 1 << gra_sdep;
        }
    } else
    {
        gra_sdep = 1;
        gra_maxcolors = 2;
    }
    }


 /* initialize the text */
    for (i=0; gra_font[i].fontname != 0; i++)
       gra_font[i].font = XOpenFont(gra_font[i].fontname);
    gra_curfont = gra_font[2].font;

    /* initialize colors */
    us_loadmap(normalred, normalgreen, normalblue, normalletter, 0,7);


 /* set up the window env. for the plot. */




 /* do the actual mapping into the graph paper */
    mappingh(mapx, numpt,maxx, minx, realxdist, originx,x);
    mappingv(mapy, numpt,maxy, miny, realydist, originy,y);
 /* CREATE THE array for plot routine */
    color=5;
    sx= (short) originx;
    sy=  sc_height- (short) originy;
    for (i=1; i<numpt; i++)
     {
      vlist[i-1].x = (short) mapx[i];
      vlist[i-1].y = ((short) mapy[i]);
      vlist[i-1].flags = 0;
     }
    vlist[numpt-1].x = (short) mapx[numpt];
    vlist[numpt-1].y = ((short) mapy[numpt]);
    vlist[numpt-1].flags = VertexDrawLastPoint;

    ox = (short) originx;
    oy = (short) originy;
    gra_width = (short) realxdist+ 0.5;
    gra_height = (short) realydist +0.5;
    num_dividx = num_dividy = 5;
    color = 2;
```

```
   graphpaper(ox, oy, gra_width,gra_height,color,num_dividx, num_dividy);
   labelx(originx, originy, maxx, minx, num_dividx, realxdist);
   labely(originx, originy, maxy, miny, num_dividy, realydist);
/* set the drawing color to yellow */
  color = 5;
   XDraw(w, vlist, numpt, 1, 1, color ,GXcopy, AllPlanes);
   text(argv[numfile], (int) (sc_width/2), 4, 5);
   XTextMaskPad(w, (int) (sc_width/2), oy+3*gra_curfont->height, 'Time',5,
     gra_curfont->id,0,
    gra_curfont->width,1, GXcopy, AllPlanes);
   XFlush();
 }

   printf('Enter q to quit\n');
   while ((ch=getchar()) != 'q');

}



/* subroutine starts from here */

/* print a text at location x, y in color */
text(string, x, y, color)
char string[];
short x,y;
int color;
{
  int slen;

  slen = strlen(string);
  XTextMaskPad(w, x, y, string, slen+1, gra_curfont->id, 0,
      gra_curfont->width, color, GXcopy, AllPlanes);
}


maxmin(data, n, max, min)
double *max, *min;
double data[];
int n;
{   int i;
    *max = *min = data[1];
    for (i=2; i <= n; i++)
      {
        *max = ( data[i] > *max)? data[i]: *max;
        *min = (data[i] < *min) ? data[i] : *min;
      }
}

mappingh(map,numpt, max, min,realxdist, origin, value)
double map[], value[];
double max, min, realxdist;   /* max and min value of the input data */
```

```
int numpt;   /* number of input data */
double origin;   /* the location of the starting point */
{
   double distance;
   int i;
   distance = max -min;
   for (i=1; i<=numpt; i++)
    map[i] = ((value[i]-min)*realxdist/distance)+ origin;
}

mappingv(map,numpt, max, min,realxdist, origin, value)
double map[], value[];
double max, min, realxdist;   /* max and min value of the input data */
int numpt;   /* number of input data */
double origin;   /* the location of the starting point */
{
   double distance;
   int i;
   distance = max -min;
   for (i=1; i<=numpt; i++)
    map[i] = -((value[i]-min)*realxdist/distance)+ origin;
}



us_loadmap(red, green, blue, letter, low, high)
 int *red, *green, *blue;
char *letter;
int low, high;
{
  int i;
  Color def;
  for (i=low; i<=high; i++)
    {
      if (i < 0 || i >= 254) continue;

      if (red[i-low] < 0 || red[i-low] > 255) continue;
      if (green[i-low] < 0 || green[i-low] > 255) continue;
      if (blue[i-low] < 0 || blue[i-low] > 255) continue;

      def.pixel = i;
      def.red = red[i-low] * 256;
      def.green = green[i-low] * 256;
      def.blue = blue[i-low] * 256;
      XStoreColor(&def);

    }

}


printstringh(string, x, y, color)
```

```
char string[];
double x,y;
int color;
{
  int slen,i,k, slen2;
  char string1[10];
  char string2[5];
  short sx,sy,sx2,sy2;

/* find the length of the string */
  slen = strlen(string);
  if (slen > 10)
/* make sure that the string consists of 10 character; */
    for (i=1; i<= 4; i++)
      string[i+5] = string[slen-5+i];
  string[10] = '\0';

 if (slen > 10 )
  {
  i = 0;
  while ((string[i] != 'e') && (string[i] != '\0'))
   {
     string1[i] = string[i];
     i++;
   }
  string1[i] = '\0';
  }
  else
  {
  i = 0;
  while (string[i] != '\0')
   {
     string1[i] = string[i];
     i++;
   }
  string1[i] = '\0';
  }
  slen2 = strlen(string1);
  if ( slen2 <= 6)
   for (k=0; k<= 4; k++)
     string2[k] = string[i+k];

  sy2 = (short) y + gra_curfont->height;
  sx2 = (short) x;
  sx = (short) x - 2*gra_curfont->width;
  sy = (short) y;
  slen = strlen(string1);
  XTextMaskPad(w, sx, sy, string1, slen+1,gra_curfont->id, 0,
     gra_curfont->width, color, GXcopy, AllPlanes);
  if (slen2 > 5)
    XTextMaskPad(w, sx2, sy2, string2, 5 ,gra_curfont->id, 0,
     gra_curfont->width, color,
       GXcopy, AllPlanes);
}
```

```
printstringv(string, x,y, color)
char string[];
double x,y;
int color;
{
  int slen,i,k,slen2;
  char string1[10];
  char string2[5];
  short sx,sy,sx2,sy2;

/* find the length of the string */
  slen = strlen(string);
  if (slen > 10)
/* make sure that the string consists of 10 character; */
    for (i=1; i<= 4; i++)
      string[i+5] = string[slen-5+i];
  string[10] = '\0';
  if (slen > 10)
  {
  i = 0;
  while ((string[i] != 'e') && (string[i] != '\0') )
   {
     string1[i] = string[i];
     i++;
   }
  string1[i] = '\0';
  }
  else
   {  i = 0;
  while (string[i] != '\0')
   {
     string1[i] = string[i];
     i++;
   }
  string1[i] = '\0';
   }
  slen2 = strlen(string1);
  if (slen2 <= 6)
    for (k=0; k<= 4; k++)
      string2[k] = string[i+k];

 /* print text vertically */
   sx = (short) x - (slen2+1)*(gra_curfont->width);
   sy = (short) y;
   sy2 = sy - gra_curfont->height;
   XTextMaskPad(w, sx, sy2, string1, slen2,gra_curfont->id,0,
    gra_curfont->width, color,
      GXcopy, AllPlanes);
   if (slen2 > 5)
   XTextMaskPad(w, sx, sy, string2, 5 ,gra_curfont->id, 0,gra_curfont->width,
```

```
    color,
    GXcopy, AllPlanes);
}

labelx(ox, oy, maxx, minx, num_dividx, realxdist)
double maxx, minx, realxdist;
double ox, oy;
int num_dividx;
{
  double xdistance, divid, x, value;
  int i, color=5;
  char string[20];

  xdistance = maxx - minx;
  divid = (xdistance) / ((double) num_dividx);

  for (i=0; i <= num_dividx; i++)
   {
    x = ox + (double) ((i*realxdist) / ((double) num_dividx));
    value = minx + (i * divid);
    gcvt(value, 10, string);
    printstringh(string, x, oy, color);
   }
}

labely(ox, oy, maxy, miny, num_dividy, realydist)
double maxy, miny, realydist;
double ox, oy;
int num_dividy;
{
  double ydistance, divid, y,value;
  int i, color=6;
  char string[20];

  ydistance = maxy - miny;
  divid = (ydistance) / ((double) num_dividy);

  for (i=0; i <= num_dividy; i++)
   {
    y = oy - (double) ((i*realydist) / ((double) num_dividy));
    value = miny + (i* divid);
    gcvt(value, 10, string);
    printstringv(string, ox, y,color);
   }
}

graphpaper(ox,oy, width, height, cclor, num_dividx, num_dividy)
short ox, oy, width, height, num_dividx, num_dividy;
int color;

{ short x,y, cellx,celly,subcellx,subcelly, endx, endy;
```

```
  Pattern line_pattern;
  Vertex vlist[2];
  int i, k;
  short oxx, subx, subsubx;
  line_pattern = XMakePattern(0x8888, 16, 1);

  endx = ox + width;
  endy = oy - height;

/* the outer bound of the paper */
  XLine(w,ox ,oy, ox, endy, 1,1,color,GXcopy, AllPlanes);
  XLine(w,ox ,endy, endx, endy, 1,1,color,GXcopy, AllPlanes);
  XLine(w,endx ,endy, endx, oy, 1,1,color,GXcopy, AllPlanes);
  XLine(w,endx ,oy, ox, oy, 1,1,color,GXcopy, AllPlanes);

/* draw the dotted  line */

  cellx = (short) (width/num_dividx +0.5);
  celly = (short) (height/num_dividy +0.5);

/* draw the vertical dotted line */
  oxx = ox;
  for (i=1; i<= num_dividx; i++)
   {
     x = ox + i*cellx;
     vlist[0].x = vlist[1].x = x;
     vlist[0].y = oy;
     vlist[1].y = endy;
     vlist[0].flags = 0;
     vlist[1].flags = VertexDrawLastPoint;
     XDrawDashed(w, vlist,2, 1,1,color, line_pattern, GXcopy, AllPlanes);
     subx = x - oxx;

     subsubx = (short) (subx/4 + 0.5);
     for (k=1; k <= 3; k++)
       {
         x =oxx + k*subsubx;
         XLine(w, x, oy, x, oy-5,1,1,color,GXcopy, AllPlanes);
       }
     oxx = ox + i*cellx;
   }
/* draw the hori dotted line */

  oxx = oy;
  vlist[0].x = ox; vlist[1].x = endx;
  for (i=1; i <= num_dividy; i++)
   {
     y = oy - (i*celly);
     vlist[0].y = vlist[1].y = y;
     XDrawDashed(w, vlist,2, 1,1,color, line_pattern, GXcopy, AllPlanes);
     subx = oxx - y;
     subsubx = (short) (subx/4 + 0.5);
     for (k=1; k <= 3; k++)
       {
```

```
        y =(short) (oxx - k*subsubx);
        XLine(w, ox,  y, ox+5, y,1,1,color,GXcopy, AllPlanes);
        }
      oxx = oy - (i*celly);
    }

/* draw the small subcell-dividson */

}
```

```
/* ------------------------------------------------------
   This program is used to convert the regular SPICE data file
   to plot.c input data format. That is  to put the time axis
   data and one of the Y-axis data into a single file.
   The output data file will read into plot.c program to display
   the output in Unix (X-Window) environment .
   If a file has more than one set of data, the next data set will be
   appended with '.x', where x will be 2, 3, 4, 5..... to indicate
   that which set of data is.


   Author: Alger W. K. Yeung
   Date: November 9. 1987 1:30 AM
   operating System: Ultrix
   Graphic Driver: X-Window Ver. 10
 ------------------------------------------------------ */

#include <stdio.h>
#include <strings.h>
#include <math.h>

main(argc, argv)
int argc;
char *argv[];

{
  FILE *infile, *outfile;
  char string[132], string1[132];
  char filename[9][20], ext[10][10];
  double array[9][1500];
  double sharray[9];
  int i,j,k,l, index;
  int numset=0;
  char ch;

  if (argc < 2 )
   {  printf('Usage: vt2x input_file\n');
      exit(1);
   }
  strcpy(ext[0],'.0');
  strcpy(ext[1],'.1');
  strcpy(ext[2],'.2');
  strcpy(ext[3],'.3');
  strcpy(ext[4],'.4');
  strcpy(ext[5],'.5');
  strcpy(ext[6],'.6');
  strcpy(ext[7],'.7');
  strcpy(ext[8],'.8');
  strcpy(ext[9],'.9');


  infile =fopen(argv[1],'r');

  while (feof(infile) == NULL)
```

```
{

 numset++;

while ( (fgets(string, 132, infile) != NULL) && (string[0] != 'X') )
 {
     strcpy(string1, string);
 }
if (feof( infile) )
  {
    fclose(infile);
    exit(0);
  }


i= 0;
j= 0;
while( string1[i] != '\n')
{
 if (string1[i] == ' ')
   { i++; k =i; }
 else
  { while ( (string1[i] != ' ') && (string1[i++] != '\n') );
     {
       j++; index =0;
       for (l=0; l< (i-k); l++)
        if( string1[k + l] != ')' && string[k+l] != '(' )
            {
              filename[j][index] = string[k+l];
              index++;
            }
       filename[j][index] = '\0';
       if (numset != 1)
          strcat(filename[j], ext[numset] );

     }
  }
 }
 printf("Number of function curves in the file = %d\n", j -1);
 printf("They are as follows:\n");

/*
 for(l=2; l<=j; l++)
   printf("%s\n", filename[l]);
*/

 fgets(string, 132, infile);


 i=0;

while ( (fgets(string, 132, infile) != NULL) && (string[0] != 'Y') )
 {
```

```
    string2double(string, j, sharray);
    for (k=0; k < j; k++)
         array[k][i] = sharray[k];
    i++;

   }

   for(l=1; l <j; l++)
   {
    outfile = fopen(filename[l + 1],'w');
    for (k=0; k < i; k++)
       fprintf(outfile, '%e %e\n', array[0][k], array[l][k]);
    fclose(outfile);
    printf('%s is created.\n', filename[l +1]);
   }


}


} /* main */
/* convert a string of 132 chars to a number of double-type value=array */

string2double(string, numdata, sharray)
int numdata;
char string[];
double sharray[9];

{
 int i,k,j,l;
 int numchar, index;
 char shortdata[40];
 char shortdata1[40];

 i = 0;

 for (l=1; l <= numdata; l++)
 {

  while (string[i++] == ' '); /* determine the beginning of a string */
  k = i;
  k--;
  /* determine the position of the end of the string */
  while ( (string[i] != ' ') && (string[i++] != '\n') );

  numchar = i - k;

 /* convert the string fron index k to index i to a double value */
  for (index=0; index < numchar; index++)
   shortdata[index] = string[k + index];

  shortdata[numchar] = '\0';

  strcpy(shortdata1, shortdata);
```

```
    sharray[l-1] = ( double ) (atof( shortdata1));

  }
}

/* for debugging purpose. */

tell(string)
double string;
{
  printf("%e", string);
 }

show(string)
char string[];
{
 printf("%s", string);
}
```

```
#include <stdio.h>
FILE *infile,*outfile;

main(argc, argv)
int argc; char *argv[];
{
  char string[90];
  int numchar,j, dummy, spnum;
  int kk;

  if ((infile = fopen(argv[1],"r")) == NULL) {
    printf("Usage:con_post infile outfile\n");
    exit(1);
  }
  outfile = fopen(argv[2],"w");

   while (fgets(string, 90, infile) != NULL) {
        dummy = 0;
     if(string[0]=='1' && string[2]=='s' && string[9] =='c')
        {  fprintf(outfile,"%s", string);
           fprintf(outfile,"/cpoly { newpath moveto lineto lineto\n");
           fprintf(outfile,"         lineto closepath stroke } def\n");
           dummy =1;
          }

       if (string[0]=='P' && string[1]=='a' && string[2]=='t' &&
           string[3]=='t' && string[4]=='e' && string[5]=='r')
         {
           j=1;
           while(string[j] != '[')
                j++;
           spnum = 0;
           kk = j;
           while(string[++kk] != ']')
            if (string[kk] == ',')
               spnum++;
           if (spnum == 7)
           {
           while(string[++j] != ']')
             putc(string[j], outfile);
           fprintf(outfile, " cpoly\n");
           }
           dummy =1;
         }

       if (string[0]=='F' && string[1]=='i' && string[2]=='l' &&
           string[3]=='l' && string[4]=='e' && string[5]=='d')
         {  dummy = 1;
         }
       if (string[0]=='/' && string[1]=='P' && string[2]=='a' &&
           string[3]=='t' && string[4]=='t' && string[5]=='e')
        .{  dummy = 1;
         }
       if (string[0]=='d' && string[1]=='e' && string[2]=='f')
```

```
        {  dummy = 1;
        }

        if (dummy == 0)
          fprintf(outfile,'%s', string);


   }
  fclose(infile);
  fclose(outfile);
}
```

Appendix VII

ECIFIN/DCIFIN Translator

Appendix VII

ECIFIN/DCIFIN Programs

This appendix describes how to use the ECIFIN/DCIFIN programs for translating files between different data formats so that files generated by the Electric design system can be used by the Phoenix Data System software package and the Daisy workstation, and vice versa. The programs are written in C language and can be run on both VAX-VMS and Ultrix operating systems.

To convert a CIF file generated by the Daisy's CIF_OUT program into Electric's CIF format, type

% *ecifin* Daisy-CIF-file output-file

To convert a CIF file generated by the Electric design system into the Daisy's CIF format, type

% *dcifin* Electric-CIF-file output-file

If the programs are intended to run on the VMS operating system, two foreign commands must be created before invoking the programs. The foreign commands are created as follows.

$ *ecifin* :==$ dua1:[user.directory]ECIFIN.EXE

$ *dcifin* :==$ dua1:[user.directory]DCIFIN.EXE

Then, use the commands, *ecifin* and *dcifin*, as the regular VMS commands. The reason of creating the foreign commands is that the input and output files can be specified followed the command on the same line.

The programs are currently located in the directory, '/users/yeung/programs', in the workstation and the listings are included in the following.

```
/*""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
  This program is written for converting Daisy generated CIF file into
  a new CIF file which can be understood by Electric. In fact, it is
  CIF file translator. The program primarily changes the format of
  stating the name of each cell, comment command and call commands used
  in the file. This program can run in VAX-VMS or Unix OS environment.
  If used in VMS, a foreign command must be created first in the
  following way:

  $ ecifin :==$ sys$user:[username.subdir]ecifin.exe

  To run the program, type

  $ ecifin input-file output-file

  """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""*/


/*""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
  Author: Alger Yeung
  Version: 1.0
  Date: November 1987
  Purpose: a module program for MOSC silicon compiler
  """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""*/

#include <stdio.h>
FILE *infile, *outfile1, *outfile;

main(argc, argv)
int argc; char *argv[];
{
  char string[82];
  int numchar;
  int checkcells, numcell=0;   /* number of define cell in the file */

  if ((infile = fopen(argv[1],"r")) == NULL) {
    printf("Usage: cifin infile outfile\n");
    exit(1);
  }
  outfile1 = fopen(argv[2],"w");
  outfile = fopen("temp101.dat","w");

   while (fgets(string, 82, infile) != NULL) {
     switch(string[0]) {

     case 'D': { if (string[1] == 'S')
                  {
                   numcell++;
                   addones(string);
                  }
                else
                   fprintf(outfile,"%s", string);
                break; }
```

```
    case '(':  commentname(string);
               break;

    case 'L': { if (string[2]=='L' && string[3]=='0' )
                    fprintf(outfile,'( The name of the port:- );\n');
                else
                    fprintf(outfile,'%s', string);
                break; }

    case '1': { putc('9', outfile);
                string[0] = '4';
                fprintf(outfile,'%s', string);
                break; }


    default: fprintf(outfile,'%s', string);
     }
}
  fclose(infile);
  fclose(outfile);
  infile = fopen('templ01.dat','r');
  checkcells = 0;
  while (fgets(string, 82, infile) != NULL) {
     switch(string[0]) {

     case 'D': {
                fprintf(outfile1,'%s',string);
                if (string[1] == 'F')
                 {
                  checkcells++;
                  if ( checkcells == numcell)
                     fprintf(outfile1,'DS %d 1 1;\n', checkcells+1);
                 }

                break;
               }
     case 'E': { fprintf(outfile1,'DF;\n');
                 fprintf(outfile1,'C %d;\n', checkcells+1);
                 fprintf(outfile1,'%s',string);

                 break;
               }

     default: fprintf(outfile1,'%s', string);

   }  }
   fclose(infile);
   fclose(outfile1);
   system('rm templ01.dat');  /* used only in UNix envi */
  /*  delete(infile);     used in VAX-11 C only */


}
```

```
commentname(string)
char string[];
{
   int i, k;
   fprintf(outfile, "%s", string);
   if ( string [2]=='B' && string[3]=='e' && string[4]=='g' && string[5]=='i'
      && string[6]=='n' )
     { i = 0;
       while (string[i] != ':')
           i++;
       i++;
       string[0] = '9';
       k = 1;
       while (string[i] != ')' )
         {   string[k] = string[i];
             i++; k++;
         }
      string[k++] = ';';
      string[k++] = '\n';
      i=0;
      while (string[i] != '\n')
        putc(string[i++], outfile);
      putc('\n', outfile);

     }

}


addones(string)
char string[];
{
   int i=0;

   while (string[i++] != ';');
   i--;
   string[i++] = ' ';
   string[i++] = '1';
   string[i++] = ' ';
   string[i++] = '1';
   string[i++] = ';';
   string[i++] = '\n';
   string[i++] = '\0';
   string[i++] = '\n';
   fprintf(outfile, "%s", string);
}
```

```
/*********************************************************************
  This program is written for converting CIF file generated by
  Electric into a new CIF file for Phoenix Data System and Daisy
  system. This program looks at the cell name, call command and
  comment command used in CIF.
  *******************************************************************/


/*********************************************************************
  Author: Alger Yeung
  Version: 1.0
  Date: November 1987
  Purpose: a module program for MOSC silicon compiler
  *******************************************************************/

#include <stdio.h>
FILE *infile, *outfilel, *outfile;

main(argc, argv)
int argc; char *argv[];
{
  char string[82];
  int k;

  if ( argc < 3 || (infile = fopen(argv[1],"r")) == NULL) {
    printf("Usage: dcifin infile outfile\n");
    exit(1);
  }
  outfile = fopen(argv[2],"w");

   while (fgets(string, 82, infile) != NULL) {
     switch(string[0]) {

     case 'D': { if (string[1] == 'S')
                  {
                   removeones(string);
                  }
                 else
                   fprintf(outfile,"%s", string);
                 break; }


      case '9':
               { if (string[1] == '4')
                   { fprintf(outfile,"L L0;\n");
                     string[0] = '1';
                     string[1] = ' ';
                     fprintf(outfile,"%s", string);
                    }
                  else if (string[1] == ' ')
                  {
                    fprintf(outfile,"( Begin symbol: ");
                    k = 2;
```

```
                    while (string[k] != ';')
                        putc( string[k++], outfile);
                    fprintf(outfile, " );\n");
                   }
                 break;
               }


     default: fprintf(outfile,"%s", string);
      }
  }
  fclose(infile);
  fclose(outfile);


}

commentname(string)
char string[];
{
   int i, k;
   fprintf(outfile, "%s", string);
   if ( string [2]=='B' && string[3]=='e' && string[4]=='g' && string[5]=='i'
      && string[6]=='n' )
     { i = 0;
       while (string[i] != ':')
           i++;
       i++;
       string[0] = '9';
       k = 1;
       while (string[i] != ')' )
         {   string[k] = string[i];
             i++; k++;
         }
      string[k++] = ';';
      string[k++] = '\n';
      i=0;
      while (string[i] != '\n')
        putc(string[i++], outfile);
      putc('\n', outfile);

     }

}


removeones(string)
char string[];
{
   int i=0;

  while (string[i++] != ';');
   i--;
   string[i-4] = ';';
```

```
    string[i-3] = '\n';
    string[i-2] = '\0';
    string[i-1] = '\n';
    fprintf(outfile, "%s", string);
}
```

Appendix VIII

Program Listing of MOSC

```
/*************************************************************************
  This is a main program of the MOSC silicon compiler. It duty is to
  select the proper operator and invoke the appropriate subprograms.
  This program is supposed to run on the Unix OS environment because
  it makes use of a Unix command:system. The command string is formed
  by using string commands.

  This program handles the first memory structure cell only but it can
  easily expanded to take care the second structure.

  ************************************************************************/

/*************************************************************************
  Author: Alger Yeung
  Version: 1.0
  Date: November 1987
  Purpose: a module program for MOSC silicon compiler
  ************************************************************************/

#include <stdio.h>
#include <strings.h>

main(argc, argv)
int argc;
char *argv[];
{
  char ch;
  char string[60];

  if (argc < 2)
   {
     printf("Usage: silicon output-filename\n");
     exit(1);
   }

   printf("\n\n\n\n******************************");
   printf("\n*     SILICON COMPILER       *");
   printf("\n******************************\n\n\n");
   printf("Adder/Multipler/Constant? (A/M/C) ==> ");
   while ( (ch=getchar()) !='a' && ch !='A' && ch !='m' && ch !='M' &&
        ch !='c' && ch != 'C');
   printf("\n\n\n");

   if (ch =='m' || ch == 'M')
    {
     strcpy(string,"petermultipler.com ");
     strcat(string, argv[1]);
     system(string);
    }
   else
   if ( ch =='c' || ch =='C')
    {
     strcpy(string,"one_data.com ");
     strcat(string, argv[1]);
```

```
    system(string);
    }
  else
  {
    strcpy(string, "multadder ");
    strcat(string, argv[1]);
    system(string);
   }

printf("\n\n*** Silicon compilation finishes...\n");

}
```

```
/* ------------------------------------------------------------
   This silicon compiler is designed to generate adder and substrator
   by a VLSI desinger without concerning the actual layout process.
   The basic cell structure is defined by Mr. Peter Bird. This compiler
   creates different ROM content for the arith. operations.
   This compiler generates CIF codes for the content. A modulo number is
   supplied by the user to specify what content will be mapped into the
   ROM.
   The cell number 20 is basically a CF (diffusion) layer which contributes
   the content of the ROM.

   The compiler packs all the created cells into a single CIF data files.
   The CIF file is created in heirch fashion.

   Created on November 4, 1987
   By
      Alger Yeung
   University of Windsor.
   ------------------------------------------------------------ */


#include <stdio.h>
#include <strings.h>
#define BIT 5
#define MAXNUM 200
#define STARTX -17750
#define STARTY 8350   /* starting location of the whole cell */
#define BLK_DELTA_X 9400
#define BLK_DELTA_Y 2200  /* size of a block 1X5 */


FILE *infile, *outfile;
FILE *infilem, *outfilem;  /* used in mask subroutine */
int tnumcell;
int basiccellnum;
char argname[40][40];
char fileorder[6][2];
int cellnum, cellnamenum;

main(argc, argv)
int argc;
char *argv[];
{  int i,j,k,l;
   int numadder;

   strcpy(fileorder[0], "0");
   strcpy(fileorder[1], "1");
   strcpy(fileorder[2], "2");
   strcpy(fileorder[3], "3");
   strcpy(fileorder[4], "4");
   strcpy(fileorder[5], "5");
   if ( argc < 2 )
     {
```

```
      printf('Usage: multadder filename\n');
      exit(1);
    } ;

  if ( (outfile = fopen(argv[1],'w')) == NULL)
    {
      printf('Error in opening the file %s\n', argv[1]);
       exit(1);
    }
  printf(' ********************************\n');
  printf(' *** VLSI adders/substrators ***\n');
  printf(' ********************************\n\n');


  printf('Each adder/substractor consists of 5 modules.\n');
  printf('Each modulo has specified ROM content.\n\n');

  printf('Enter number of adder:==> ');
  scanf('%d', &numadder);/* read in number of adder interested */
  printf('\n\n');


  tnumcell = 0;
/* transfer and count the number of cells in the basic ROM cell */
  trans_count('/users/yeung/programs/peter/cif.cif');
  basiccellnum = tnumcell;
/* need the starting cell number :ie=21 */

  cellnum = basiccellnum + 1;
  cellnamenum = 501;

  for (i=1; i<=numadder; i++)
   {
     strcpy(argname[1],'bit01');
/* generate the bit data for adder or substractor */

     gen_data_bit(2);
     k = 2;
     strcpy(argname[k],'c1');      strcat(argname[k++],fileorder[i]);

     strcpy(argname[k],'c2');      strcat(argname[k++],fileorder[i]);

     strcpy(argname[k],'c3');      strcat(argname[k++],fileorder[i]);

     strcpy(argname[k],'c4');      strcat(argname[k++],fileorder[i]);

     strcpy(argname[k],'c5');      strcat(argname[k++],fileorder[i]);

/*   for (l=2; l<= 6; l++)
     printf('%s\n', argname[l]);
*/
/* create the mask and store it in a specify files */

     mask(6);
```

```
    }
/* merge all mask data into a single file specified in argv[1] */

    combine(numadder);


/* remove all intermediate files */
   strcpy(argname[1],"rm bit01");
   system(argname[1]);

   for (i=1; i<=numadder; i++)
    {
      k = 2;
      strcpy(argname[k],"rm c1");      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],"rm c2");      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],"rm c3");      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],"rm c4");      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],"rm c5");      strcat(argname[k++],fileorder[i]);

      for (l=2; l<= 6; l++)
         system(argname[l]);
    }
}



/*-------------------------------------------------------
  gen_data creates binary data 1 or 0 for adder or substractor in five
  different modules. Each modulo has 32 set of 5-bit data, two more bits
  are used in parity check. However, some programs requires only five
  bits data .
-------------------------------------------------------*/

gen_data_bit(argc)
int argc;
{
   FILE *outfile1;
   unsigned short mod[BIT][MAXNUM], number=1, adder, reset, check, modulo;
   char ch;
   register int i, ii;

   if (argc < 2)
     { printf("Usage: Program outfile-name\n");
       exit(1);
     }

/* determine the range of address 0--number */
```

```
   for (i=1; i<=BIT; i++)
      number *= 2;
   number--;


   printf("\nEnter the modulo number:==> ");
   scanf("%d", &modulo);


   printf("\nAdder/Substractor? (A/S)==> ");
   while((ch=getchar()) != 'A' && ch !='a' && ch!='s' && ch!='S');
/* addition parameter */
   adder = 1;
   reset = 0;
   check = modulo;

/* substraction parameter */
   if (ch == 's' || ch == 'S')
     { adder = -1;
       reset = modulo - 1;
       check = -1;
     }



/* determine the first number of each array */

   mod[0][0] = 1;
   for (i=1; i<= BIT-1; i++)
     mod[i][0] = mod[i-1][0] * 2;


   for (i=0; i<= BIT-1; i++)
     {
       for (ii=1; ii<=number; ii++)
         {   mod[i][ii] = mod[i][ii-1] + adder;
             if (mod[i][ii] == check)
                 mod[i][ii] = reset;
         }
   }
   outfile1 = fopen(argname[1],"w");

/* write out the number of bit deal here  */
   fprintf(outfile1,"%d\n", BIT);
/* least signi bit is taken care first */
/* therefore l.s.b is come first */
     for (i=0; i<=BIT-1; i++)
      {
       for (ii=0; ii<=number; ii++)
       {  register int j;
          int ebit1, ebit2, temp;


/* check number of '1' bit in the address number */
```

```
        ebit1 =0;
        for (j=0; j<= BIT-1; j++)
          { temp = ii;
            if (temp & 0x01)
              ebit1++;
            temp >>= 1;
          }
/* check number of '1' bit in the data (content ) */
        ebit2 = 0;
        for (j=0; j<= BIT-1; j++)
           {
             if ( mod[i][ii] & 0x01 )
              {
                fprintf(outfile1,"1 ");
                ebit2++;
              }
             else
                fprintf(outfile1,"0 ");
             mod[i][ii] >>= 1;
            }

 /* use even parity scheme */
        if (ebit1 & 0x01)        /* take care the address lines */
           fprintf(outfile1,"1 ");
        else
           fprintf(outfile1,"0 ");


        if (ebit2 & 0x01)        /* take care the data lines  */
           fprintf(outfile1,"1 ");
        else
           fprintf(outfile1,"0 ");

        putc('\n', outfile1);
       }
 }

     fclose(outfile1);

 }

 /* ------------------------------------------------------------
   Mask.c creates mask layer for the ROM implementation. The mask files
   will be stored in argv[2], argv[3],....argv[6]. The subroutine will
   prompt for starting cell number and cell name number .
   ------------------------------------------------------------ */


 /* mask.c is include in the following  */


 mask(argc)
 int argc;
```

```
{

 int i,j,k,l;
 int matrix[MAXNUM][BIT];  /* define the size of the 2D array for input data */
 int numset;    /* number of set of data in the infilem */
 int numfile;
 char ch;

 if (argc < 3)
  {
    printf("Usage: Program infilem outfilem(s)\n");
    exit(1);

  }

 if ( (infilem = fopen(argname[1],"r")) == NULL)
    {
       printf("Failure in open the file %s\n", argname[1]);
       exit(1);
    }

  fscanf(infilem,"%d", &numset);  /* find how many set of data in the file */
/*  printf("%d\n", numset);  */
/* position the file pointer to next line */
  while (ch=getc(infilem) != '\n');

  for (i=1; i<=numset; i++)  /* take care all cells ie. 5 cells */
   {
     readindata(matrix);
     outfilem = fopen(argname[i+1],"w");
     gen_data(matrix);
     fclose(outfilem);
     cellnum++; cellnamenum++;
   }

  fclose(infilem);

} /* main */



readindata(matrix)
int matrix[MAXNUM][BIT];

{
  register int i,l;
  char string[40];
  for (i=0; i<=31; i++)

   {

    fgets(string, 40, infilem);
     for (l=0; l<=4; l++)
      if (string[l*2] == '1')
```

```
        matrix[i][1] = 1;
      else
        matrix[i][1] = 0;
/*
     printf("%d %d %d %d %d\n", i, matrix[i][0], matrix[i][1], matrix[i][2],
                        matrix[i][3], matrix[i][4]);
*/

  }

}


/* take care of one whole cell */
gen_data(matrix)  /* take care of 8X4 rows */
int matrix[MAXNUM][BIT];
{

   int i, k, l;
   int x, y;
   l = -1;
   fprintf(outfilem,"DS %d 1 1;\n", cellnum);
   fprintf(outfilem,"9 DATAC_%d;\n", cellnamenum);

   for (i=0; i<=3; i++)
     {
       x = STARTX + i*BLK_DELTA_X;

       for (k=0; k<=7; k++)
        { l++;  /* specfy the index of the matrix */
          y = STARTY - k*BLK_DELTA_Y;
          onerow(matrix, l, x, y);
        }
       }
    fprintf(outfilem,"DF;\n");

  }


   onerow(matrix, l, x,y)  /* take care of one row of 5 bit data */
   int matrix[MAXNUM][BIT];
   int l, x,y;

   {
     int i, temp;
     int smal_delta_x = 1600;  /* 16 micron between diffusion CF */

     y = y-650;  /* shift 6.5 micron down */
     for (i=0; i<=4; i++)  /* 5 bit data cif */
      {
       temp = x + i*smal_delta_x + 450;  /* shift right 4.5 micron */
       if ( matrix[l][i] == 1 )  /* if "1" ,put a CF layer */
         fprintf(outfilem,"C 20 R 100 0 T %d %d;\n", temp,y);
```

```
    }

}



/* ------------------------------------------------------------
Combine.c combine all cif code in single file which is a hierch file
   The basic cell is defined once, and the data cells are called
   individually. The  final cell is in CHIP.
------------------------------------------------------------ */


/* combine.c file is included in the following */


combine(numadder)
int numadder;

{

  char ch;
  int i,j,k, l, ii;
  int cell_height;
  int longl, lengy;

/* transfer all the data from argv[i] cell to the new output file */
  tnumcell =basiccellnum;

  for (i=1; i<= numadder; i++)
   {   k = 2;
      strcpy(argname[k],'c1');      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],'c2');      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],'c3');      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],'c4');      strcat(argname[k++],fileorder[i]);

      strcpy(argname[k],'c5');      strcat(argname[k++],fileorder[i]);
      for (j=2; j<= 6; j++)
          trans_count(argname[j]);
   }
 if (numadder > 1)
  { trans_count('/users/yeung/programs/peter/connect2.cif');
    tnumcell--;
  }
 if (numadder > 2)
  { trans_count('/users/yeung/programs/peter/connect1.cif');
```

```
    tnumcell--;
  }


 for (ii=0; ii < numadder; ii++)
 {
/* group the rom cell and the data cell together */
  for (i=1; i <= 5 ; i++)
   { int ftr;
     ftr = ii*5 + i;
     j = ii*5 + i+tnumcell;    /* built more new cells */
     l = ii* 5 + i+basiccellnum;   /* call from the basic cell */
     fprintf(outfile,"DS %d 1 1;\n", j);
     fprintf(outfile,"9 ROM_FTR%d;\n", ftr);
     fprintf(outfile,"C %d R -100 0 M Y T 0 0;\n", basiccellnum-1);
     fprintf(outfile,"C %d R 100 0 T 14050 -21100;\n",1);
     fprintf(outfile,"DF;\n");
   }
 }



   k = tnumcell + 1+ numadder*5;
 /* make up the final call cell */
  fprintf(outfile,"DS %d 1 1;\n", k);
  fprintf(outfile,"9 CHIP;\n");



 cell_height = 0;
 for (ii=0; ii < numadder; ii++)
   {
    for (i=0; i < 5; i++)
    {
     if ( ii & 0x01)  /* do not flip */
       fprintf(outfile,"C %d T %d %d;\n", (tnumcell+1+i +ii*5), (i*69900),
          cell_height);
     else  /* flip about X-axis */
       fprintf(outfile,"C %d M Y T %d %d;\n", (tnumcell+1+i +ii*5), (i*69900),
          cell_height);
    }

    if ( ii & 0x01)
       cell_height += 63400;  /* for odd row adder */
    else
       cell_height += 62400;  /* for even row adder */

    }
   /* add the long connect between two row of adders */

    lengy = 0;
    for (longl= 2; longl <= numadder; longl++)
     {
```

```
    longl++;
    fprintf(outfile,"C 900 T 316650 %d;\n", (lengy*125800) + 31200 );
    lengy += 1;
    }
  /* add the short connect between two row of adders */

   lengy = 1;
   for (longl= 3; longl <= numadder; longl++)
    {
    longl++;
    fprintf(outfile,"C 901 T -36550 %d;\n", (lengy*94100));
    lengy += 1;
    }


   fprintf(outfile,"DF;\n");
   fprintf(outfile,"C %d;\n", k);
   fprintf(outfile,"E\n");


fclose(outfile);

}

/* Count the number of cells in the file and transfer from the old file
   to new file */

trans_count(filename)
char filename[];
{
  char string[82];

  infile = fopen(filename, "r");
  /* transfer all the data from filename cell to the new output file */

  while (fgets(string, 82, infile) != NULL)  /* not end of file */
   {
    if ( string[0]=='D' && string[1] =='F' ) /* count how many cell */
      tnumcell++;

    fprintf(outfile,"%s", string);
   }

  fclose(infile);
}
```

```
#include <stdio.h>
#define BIT 5
#define MAXNUM 200

main(argc, argv)
int argc;
char *argv[];
{
   FILE *outfile;
   unsigned short mod[BIT][MAXNUM], number=1, adder, reset, check, modulo;
   char ch;
   register int i, ii;

   if (argc < 2)
     { printf("Usage: Program outfile-name\n");
       exit(1);
     }

 /* determine the range of address 0--number */
   for (i=1; i<=BIT; i++)
      number *= 2;
   number--;


   printf("\nEnter the modulo number: ");
   scanf("%d", &modulo);


   printf("\nAdder/Substractor? (A/S)\n");
   while((ch=getchar()) != 'A' && ch !='a' && ch!='s' && ch!='S');
 /* addition parameter */
   adder = 1;
   reset = 0;
   check = modulo;


 /* determine the first number of each array */

   mod[0][0] = 1;
   for (i=1; i<= BIT-1; i++)
     mod[i][0] = mod[i-1][0] * 2;



 /* substraction parameter */
   if (ch == 's' || ch == 'S')
     {
       mod[0][0] = 31;
       mod[1][0] = 30;
       mod[2][0] = 28;
       mod[3][0] = 24;
       mod[4][0] = 16;
     }
```

```
   for (i=0; i<= BIT-1; i++)
     {
       for (ii=1; ii<=number; ii++)
         {   mod[i][ii] = mod[i][ii-1] + adder;
             if (mod[i][ii] == check)
                 mod[i][ii] = reset;
/* testing  printf("i=%d, ii=%d, data=%d\n",i,ii, mod[i][ii]);  */
         }
   }
   outfile = fopen(argv[1],"w");

/* write out the number of bit deal here  */
   fprintf(outfile,"%d\n", BIT);
/* least signi bit is taken care first */
/* therefore l.s.b is come first */
     for (i=0; i<=BIT-1; i++)
      {
       for (ii=0; ii<=number; ii++)
       {  register int j;
          int ebit1, ebit2, temp;


/* check number of '1' bit in the address number */
        ebit1 =0;
        for (j=0; j<= BIT-1; j++)
          { temp = ii;
            if (temp & 0x01)
              ebit1++;
            temp >>= 1;
          }
/* check number of '1' bit in the data (content ) */
        ebit2 = 0;
        for (j=0; j<= BIT-1; j++)
           {
            if ( mod[i][ii] & 0x01 )
             {
               fprintf(outfile,"1 ");
               ebit2++;
             }
            else
               fprintf(outfile,"0 ");
            mod[i][ii] >>= 1;
           }

/* use even parity scheme */
        if (ebit1 & 0x01)        /* take care the address lines */
          fprintf(outfile,"1 ");
        else
          fprintf(outfile,"0 ");
```

```
      if (ebit2 & 0x01)       /* take care the data lines  */
         fprintf(outfile,"1 ");
      else
         fprintf(outfile,"0 ");

      putc('\n', outfile);
     }
}

   fclose(outfile);

}
```

```
#include <stdio.h>
#define MAXCOL 300
#define BIT 5
#define STARTX -17750
#define STARTY 8350   /* starting location of the whole cell */
#define BLK_DELTA_X 9400
#define BLK_DELTA_Y 2200  /* size of a block 1X5 */

FILE *infile, *outfile;



main(argc, argv)
int argc;
char *argv[];

{
  char ch;
  int i,j,k,l;
  int matrix[MAXCOL][BIT];  /* define the size of the 2D array for input data */
  int numset;    /* number of set of data in the infile */
  int numfile, cellnum, cellnamenum;

  if (argc < 3)
   {
     printf("Usage: Program infile outfile(s)\n");
     exit(1);

   }

  if ( (infile = fopen(argv[1],"r")) == NULL)
     {
        printf("Failure in open the file %s\n", argv[1]);
        exit(1);
     }

  fscanf(infile,"%d", &numset);  /* find how many set of data in the file */
/*  printf("%d\n", numset); */

  while( (ch = getc(infile)) != '\n');


  printf("Enter the starting cell number DS #: ");
  scanf("%d", &cellnum);
  printf("\nEnter the starting cellname number CEL_#: ");
  scanf("%d", &cellnamenum);

  for (i=1; i<=numset; i++)  /* take care all cells ie. 5 cells */
   {
     readindata(matrix);
     outfile = fopen(argv[i+1],"w");
     gen_data(matrix, cellnum, cellnamenum);
     fclose(outfile);
     cellnum++; cellnamenum++;
```

```
    }

  fclose(infile);

} /* main */


readindata(matrix)
int matrix[MAXCOL][BIT];

{
  register int i,l;
  char string[40];
  for (i=0; i<=31; i++)

    {

     fgets(string, 40, infile);
      for (l=0; l<=4; l++)
       if (string[l*2] == '1')
         matrix[i][l] = 1;
       else
         matrix[i][l] = 0;
/*
     printf("%d %d %d %d %d\n", matrix[i][0], matrix[i][1], matrix[i][2],
                    matrix[i][3], matrix[i][4]);
*/
    }

}


/* take care of one whole cell */
gen_data(matrix, cellnum, cellnamenum)  /* take care of 8X4 rows,*/
int cellnum, cellnamenum;
int matrix[MAXCOL][BIT];
{

  int i, k, l;
  int x, y;
  l = 0;
  fprintf(outfile,"DS %d 1 1;\n", cellnum);
  fprintf(outfile,"9 DATAC_%d;\n", cellnamenum);

  for (i=0; i<=3; i++)
    {
      x = STARTX + i*BLK_DELTA_X;

      for (k=0; k<=7; k++)
       {   /* specfy the index of the matrix */
         y = STARTY - k*BLK_DELTA_Y;
         onerow(matrix, l, x, y);
         l++;
```

```
      }
    }
  fprintf(outfile,"DF;\n");

}


onerow(matrix, l, x,y)  /* take care of one row of 5 bit data */
int matrix[MAXCOL][BIT];
int l, x,y;

{
  int i, temp;
  int smal_delta_x = 1600;  /* 16 micron between diffusion CF */

  y = y-650;  /* shift 6.5 micron down */
  for (i=0; i<=4; i++)  /* 5 bit data cif */
   {
    temp = x + i*smal_delta_x + 450;  /* shift right 4.5 micron */
    if ( matrix[l][i] == 1 )  /* if "1" ,put a CF layer */
      fprintf(outfile,"C 20 R 100 0 T %d %d;\n", temp,y);
   }

}
```

```
#include <stdio.h>
#define BIT 5

FILE *infile, *outfile;
int tnumcell;

main(argc, argv)
int argc;
char *argv[];

{

  char ch;

  int i,j,k, l;
  int basiccell;

  outfile = fopen(argv[1], "w");
  tnumcell = 0;

/* transfer all the data from argv[i] cell to the new output file */

/* transfer the basic cell first */
  trans_count(argv[2]);
  basiccell = tnumcell;

  for (i=3; i<= (argc-1); i++)
    trans_count(argv[i]);


/* group the rom cell and the data cell together */
  for (i=1; i <= (argc-3); i++)
   {
     j = i+tnumcell;    /* built more new cells */
     l = i+basiccell;   /* call from the basic cell */

     fprintf(outfile,"DS %d 1 1;\n", j);
     fprintf(outfile,"9 ROM_ETR%d;\n", i);
     fprintf(outfile,"C %d R 100 0 T 1600 0;\n", basiccell);
     fprintf(outfile,"C %d R 100 0 T -43900 44400;\n",l);
     fprintf(outfile,"DF;\n");
   }
 k = tnumcell + argc -2;
/* make up the final call cell */
  fprintf(outfile,"DS %d 1 1;\n", k);
  fprintf(outfile,"9 CHIP;\n");
  for (i=0; i< (argc-3); i++)
   {
    fprintf(outfile,"C %d T %d00 0;\n", (tnumcell+1+i), (i*1375));
/* 40 is used for overlap the cells to make connection */
   }
  fprintf(outfile,"DF;\n");
  fprintf(outfile,"C %d;\n", k);
```

```
  fprintf(outfile,"E\n");


fclose(outfile);

}

/* Count the number of cells in the file and transfer from the old file
   to new file */

trans_count(filename)
char filename[];
{
  char string[82];

  infile = fopen(filename, "r");
  /* transfer all the data from filename cell to the new output file */

  while (fgets(string, 82, infile) != NULL)  /* not end of file */
   {
    if ( string[0]=='D' && string[1] =='F' ) /* count how many cell */
      tnumcell++;

    fprintf(outfile,"%s", string);
   }

  fclose(infile);
}
```

```
#include <stdio.h>
#define BIT 5

FILE *infile, *outfile;
int tnumcell;

main(argc, argv)
int argc;
char *argv[];

{

  char ch;

  int i,j,k, l;
  int basiccell;
  int deltax=137500, deltay=145100;   /* distance between cells */

  outfile = fopen(argv[1], "w");
  tnumcell = 0;

/* transfer all the data from argv[i] cell to the new output file */

/* transfer the basic cell first */
  trans_count(argv[2]);
  basiccell = tnumcell;

  for (i=3; i<= (argc-1); i++)
    trans_count(argv[i]);



/* group the rom cell and the data cell together */
  for (i=1; i <= (argc-3); i++)
   {
     j = i+tnumcell;    /* built more new cells */
     l = i+basiccell;   /* call from the basic cell */

     fprintf(outfile,"DS %d 1 1;\n", j);
     fprintf(outfile,"9 ROM_PTR%d;\n", i);
/* 1600 0 and -43900 44400 can be changed for different basic cell */

     fprintf(outfile,"C %d R 100 0 T 1600 0;\n", basiccell);
     fprintf(outfile,"C %d R 100 0 T -43900 44400;\n",l);
     fprintf(outfile,"DF;\n");
   }
 k = tnumcell + argc -2;
/* make up the final call cell */
  fprintf(outfile,"DS %d 1 1;\n", k);
  fprintf(outfile,"9 CHIP;\n");
/* specfy the floor planning for multipler cell */

  for (j=1; j<=6; j++)  /* specfy the bottom row */
    fprintf(outfile,"C %d T %d 0;\n", 38-j, deltax*(j-1));
```

```
/* specfy the substractor in the middle row */
  for (j=1; j<=5; j++)
     fprintf(outfile,"C %d T %d %d;\n", 31+j, j*deltax, deltay);
/* specfy the top row for adder */
/* cell number 37 is the address content cell calculated by multaddss */

  fprintf(outfile,"C 37 MY T 0 %d;\n", 2*deltay);
  for (j=1; j<=5; j++)
    fprintf(outfile,"C %d MY T %d %d;\n", 32-j, j*deltax, 2*deltay);


  fprintf(outfile,"DF;\n");
  fprintf(outfile,"C %d;\n", k);
  fprintf(outfile,"E\n");


fclose(outfile);

}

/* Count the number of cells in the file and transfer from the old file
   to new file */

trans_count(filename)
char filename[];
{
  char string[82];

  infile = fopen(filename, "r");
  /* transfer all the data from filename cell to the new output file */

  while (fgets(string, 82, infile) != NULL)  /* not end of file */
   {
    if ( string[0]=='D' && string[1] =='F' ) /* count how many cell */
      tnumcell++;

    fprintf(outfile,"%s", string);
   }

  fclose(infile);
}
```

```
/*********************************************************************

  This program will generate memory content for the quarter square
  multiplier. The equation is based upon the following

  content = ( |address^2| / 4 ) mod m

  m = modulo-number

  Users invokes the program by typing

  % qsquare output-file

  *********************************************************************/


/*********************************************************************
  Author: Alger Yeung
  Version: 1.0
  Date: November 1987
  Purpose: a module program for MOSC silicon compiler
  *********************************************************************/

#include <stdio.h>
#define BIT 5
#define MAXNUM 200

main(argc, argv)
int argc;
char *argv[];
{
   FILE *outfile;
   unsigned short mod[MAXNUM], number=1, adder, reset, check, modulo;
   register int i, ii;

   if (argc < 2)
     { printf("Usage: Program outfile-name\n");
       exit(1);
     }

 /* determine the range of address 0--number */
   for (i=1; i<=BIT; i++)
      number *= 2;
   number--;


   printf("\nEnter the modulo number: ");
   scanf("%d", &modulo);

   check = modulo;
   ii = 4;

 /* mod( int( (address**2)/ 4) )  */
```

```
      for (ii=0; ii<=number; ii++)
        {   mod[ii] = (int) ( (ii*ii) / 4);
            while (mod[ii] >= check)
                mod[ii] = mod[ii] - check;
            printf("%3d-->%3d\n", ii, mod[ii]);
         }


/* convert the decimal number into binary number
   l.s.b comes first
*/

   outfile = fopen(argv[1],"w");
   fprintf(outfile,"1\n"); /* write down one cell in the output file */


/* least signi bit is taken care first */
/* therefore l.s.b is come first */
   for (ii=0; ii<=number; ii++)
     {  int j, ebit1, ebit2, temp;


/* check number of '1' bit in the address number */
        ebit1 =0;
        for (j=0; j<= BIT-1; j++)
          { temp = ii;
            if (temp & 0x01)
              ebit1++;
            temp >>= 1;
          }
/* check number of '1' bit in the data (content ) */
        ebit2 = 0;
        for (j=0; j<= BIT-1; j++)
           {
            if ( mod[ii] & 0x01 )
             {
               fprintf(outfile,"1 ");
               ebit2++;
             }
            else
               fprintf(outfile,"0 ");
            mod[ii] >>= 1;
           }

/* use even parity scheme */
       if (ebit1 & 0x01)        /* take care the address lines */
          fprintf(outfile,"1 ");
       else
          fprintf(outfile,"0 ");


       if (ebit2 & 0x01)       /* take care the data lines  */
          fprintf(outfile,"1 ");
```

```
        else
           fprintf(outfile,"0 ");

        putc('\n', outfile);
       }

  fclose(outfile);

}
```

```
/*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  This program will generate memory content for one memory cell only.
  Users can add, subtract and multiply a constant value. The program
  is invoked by typing

  % gen_one_data output-filename

  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^*/

/*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  Author: Alger Yeung
  Version: 1.0
  Date: November 1987
  Purpose: a module program for MOSC silicon compiler
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^*/

#include <stdio.h>
#define BIT 5
#define MAXNUM 200

main(argc, argv)
int argc;
char *argv[];
{
   FILE *outfile;
   unsigned short mod[BIT][MAXNUM], number=1, adder, reset, check, modulo;
   char type[5];
   char ch;
   register int i, ii;
   unsigned short firstvalue;

   if (argc < 2)
     { printf('Usage: Program outfile-name\n');
       exit(1);
     }

/* determine the range of address 0--number */
   for (i=1; i<=BIT; i++)
      number *= 2;
   number--;
printf('\n*********************************\n');
printf('*       Constant                *\n');
printf('*********************************\n\n\n');


   printf('\nEnter the modulo number: ');
   scanf('%d', &modulo);
   printf('\nSelect Adder/Subtractor/Multiplier==>\n');
   scanf('%s', type);
   printf('\nEnter the constant value:==> ');
   scanf('%d', &firstvalue);

   printf('\n\n');
```

```
/* addition parameter */
   adder = 1;
   reset = 0;
   check = modulo;

/* determine the first number of each array */
   mod[0][0] = firstvalue;
/* take care of subtraction */
   if (type[1]=='u'&&type[2]=='b')
      mod[0][0] = modulo - firstvalue;

/* if multiplier is wanted */
   if (type[1]='u'&&type[2]=='l')
      {
          i = 0;
          for (ii=0; ii<=number; ii++)
          {
          mod[i][ii] = ii * firstvalue;
           while ( mod[i][ii] >= modulo )
              mod[i][ii] = mod[i][ii] - modulo;
          }
       }
      else      /* construct adder/subtractor */
       {
         i = 0;
         for (ii=1; ii<=number; ii++)
           {
            mod[i][ii] = mod[i][ii-1] + adder;
            if (mod[i][ii] == check)
              mod[i][ii] = reset;
           }
       }


   outfile = fopen(argv[1],"w");

 /* write out the number of bit deal here  */
   fprintf(outfile,"1\n");
 /* least signi bit is taken care first */
 /* therefore l.s.b is come first */
       for (ii=0; ii<=number; ii++)
       {  register int j;
          int ebit1, ebit2, temp;


 /* check number of '1' bit in the address number */
        ebit1 =0;
        for (j=0; j<= BIT-1; j++)
          { temp = ii;
            if (temp & 0x01)
              ebit1++;
            temp >>= 1;
```

```
        }
/* check number of '1' bit in the data (content ) */
       ebit2 = 0;
       for (j=0; j<= BIT-1; j++)
          {
           if ( mod[i][ii] & 0x01 )
            {
              fprintf(outfile,"1 ");
              ebit2++;
            }
           else
              fprintf(outfile,"0 ");
           mod[i][ii] >>= 1;
          }

/* use even parity scheme */
       if (ebit1 & 0x01)        /* take care the address lines */
          fprintf(outfile,"1 ");
       else
          fprintf(outfile,"0 ");


       if (ebit2 & 0x01)        /* take care the data lines  */
          fprintf(outfile,"1 ");
       else
          fprintf(outfile,"0 ");

       putc('\n', outfile);
      }

    fclose(outfile);

}
```

```
/*""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""


  """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""*/
/*""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
  Author: Alger Yeung
  Version: 1.0
  Date: November 1987
  Purpose: a module program for MOSC silicon compiler
  """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""*/

#include <stdio.h>
#define BIT 5

FILE *infile, *outfile;
int tnumcell;

main(argc, argv)
int argc;
char *argv[];

{

  char ch;

  int i,j,k, l;
  int basiccell;
  int deltax=69900, deltay=62700;   /* distance between cells */

  outfile = fopen(argv[1], 'w');
  tnumcell = 0;

/* transfer all the data from argv[i] cell to the new output file */

/* transfer the basic cell first */
  trans_count(argv[2]);
  basiccell = tnumcell;

  for (i=3; i<= (argc-1); i++)
    trans_count(argv[i]);


/* group the rom cell and the data cell together */
  for (i=1; i <= (argc-3); i++)
   {
     j = i+tnumcell;    /* built more new cells */
     l = i+basiccell;   /* call from the basic cell */

     fprintf(outfile,'DS %d 1 1;\n', j);
     fprintf(outfile,'9 ROM_FIR%d;\n', i);
/* 1600 0 and -43900 44400 can be changed for different basic cell */
```

```
      fprintf(outfile,"C %d R -100 0 M Y T 0 0;\n", basiccell -1);
      fprintf(outfile,"C %d R 100 0 T 14050 -21100;\n",1);
      fprintf(outfile,"DF;\n");
    }
 k = tnumcell + argc -2;
/* make up the final call cell */
  fprintf(outfile,"DS %d 1 1;\n", k);
  fprintf(outfile,"9 CHIP;\n");
/* specfy the floor planning for multipler cell */

  for (j=1; j<=6; j++)  /* specfy the bottom row */
   fprintf(outfile,"C %d R -100 0 T %d -700;\n", 43-j, deltax*(j-1));
/* specfy the substractor in the middle row */
  for (j=1; j<=5; j++)
     fprintf(outfile,"C %d R -100 0 T %d %d;\n", 36+j, j*deltax, deltay);
/* specfy the top row for adder */
/* cell number 42 is the address content cell calculated by multaddss */

  fprintf(outfile,"C 42 R -100 0 MY T 0 %d;\n", (2*deltay -300));
  for (j=1; j<=5; j++)
    fprintf(outfile,"C %d R -100 0 MY T %d %d;\n", 37-j, j*deltax,
        (2*deltay-300) );


  fprintf(outfile,"DF;\n");
  fprintf(outfile,"C %d;\n", k);
  fprintf(outfile,"E\n");


fclose(outfile);

}

/* Count the number of cells in the file and transfer from the old file
   to new file */

trans_count(filename)
char filename[];
{
  char string[82];

  infile = fopen(filename, "r");
  /* transfer all the data from filename cell to the new output file */

  while (fgets(string, 82, infile) != NULL)  /* not end of file */
   {
    if ( string[0]=='D' && string[1] =='F' ) /* count how many cell */
      tnumcell++;

    fprintf(outfile,"%s", string);
   }

  fclose(infile);
}
```

```
program rajarom2(input,output);

{ This program will compute the coord. of CIF code which will
 be stick on the Mr. Raja's ROM. The ROM is highly regular so that
 I will program single bit first, 7-bit next, one-block next and
 eight-block.
  The first data of the input file specfies number of singleblock.ie.5
  The input is a 32*7 matrix which composes of 8 blocks. Each block
 has four 7-bit value so that each block is 4*7 matrix.
  Inside each block, there are four columns of data. Each column has 7-bit
 If the input data of any particular bit is 0, a connection is made. Or
 the connection can be skipped.
 The connection consists of a diffusion layer, two contact cuts and one
 metal CM1 layer.
 The program will use Box command heavily in the CIF Code output.
 The width and length are almost constant inside a block.
}

const
  fwr=900;
  flr=900;
  fwl=900;
  fll=900;
  fwm=900;
  flm=1500;

  cw=500;
  cl=500;

  mw=900;
  ml=1400;

 type
  onecolumn=array [1..7] of integer;
  fourcolumn=array [1..4,1..7] of integer;
  column32=array [1..32,1..7] of integer;

 var
  cml:text;
  diff:text;
  cc:text;
  infile:text;
  i, numfile, stnum, cellnamenum:integer; { number of subfile in the infile}
  matrix:column32;
  datafile:packed array [1..40] of char;
  string: packed array [1..80] of char;

{The reading-file procedure will input the 32*7 matrix into the matrix
 variable }

procedure readingfile(var matrix:column32);

 var
  i,j:integer;
```

```
{  infile:text;
   datafile: packed array [1..40] of char;
}

 begin

  for i:=1 to 32 do
    begin
      for j:=1 to 7 do
        read(infile,matrix[i,j]);
      readln(infile);
    end;
  end;


  procedure intlength(value:integer; var length:integer);
  { To find the number of digits in the integer variable value }
   var
    constant:integer;
   begin
    if value < 0 then
     begin

      constant:=-9;
      length:=1;
      repeat
        length:=length+1;
        constant:=constant*10-9;
      until constant < value;
      length:=length+2;
      end


      else
       begin

       constant:=9;
       length:=1;
       repeat
         length:=length+1;
         constant:=constant*10+9;
       until constant > value;
       length:=length+1;
       end;
      end;


       procedure contactcut(cx,cy:integer);
       var
         lcx,lcy:integer;  { number of digits of cx,cy}

       begin
```

```
  intlength(cx,lcx);
  intlength(cy,lcy);
  writeln(cc,'B',cl:4,cw:4,cx:lcx,cy:lcy,';');
end;

procedure metalone(cx,cy:integer);
var
  lcx,lcy:integer;  { number of digits of cx,cy}

begin
  intlength(cx,lcx);
  intlength(cy,lcy);
  writeln(cml,'B',ml:5,mw:4,cx:lcx,cy:lcy,';');
end;

procedure diffusion(cx,cy,l,w:integer);
var
  lcx,lcy:integer;  { number of digits of cx,cy}

begin
  intlength(cx,lcx);
  intlength(cy,lcy);
  if l=1500 then
    writeln(diff,'B',l:5,w:4,cx:lcx,cy:lcy,';')
  else
    writeln(diff,'B',l:4,w:4,cx:lcx,cy:lcy,';');
end;


procedure onebito(x,y:integer);
{ x,y is the point of upper left corner of the transistor.}
var
 fcentrexr,fcentreyr:integer;  {centre of box in diffusion layer }
 fcentrexl,fcentreyl:integer;
 fcentrexm,fcentreym:integer;

 ccentrexr,ccentreyr:integer;  {centre of box in contact-cut layer }
 ccentrexl,ccentreyl:integer;

 mcentrex,mcentrey:integer;    {centre of box in matel layer }

begin
  fcentrexl:=x+450;
  fcentreyl:=y-450;

  fcentrexm:=x+1650;
  fcentreym:=fcentreyl;

  fcentrexr:=x+2850;
  fcentreyr:=fcentreyl;

  ccentrexr:=fcentrexr;
  ccentreyr:=fcentreyr;
```

```
  ccentrexl:=fcentrexl;
  ccentreyl:=fcentreyl;

  mcentrex:=x+3100;
  mcentrey:=fcentreyl;

  diffusion(fcentrexl,fcentreyl,fll,fwl);
  diffusion(fcentrexm,fcentreym,flm,fwm);
  diffusion(fcentrexr,fcentreyr,flr,fwr);

  contactcut(ccentrexr,ccentreyr);
  contactcut(ccentrexl,ccentreyl);

  metalone(mcentrex,mcentrey);

end;



procedure onebite(x,y:integer); {These one takes care the even cloumn }
{ x,y is the point of upper left corner of the transistor.}
var
 fcentrexr,fcentreyr:integer;  {centre of box in diffusion layer }
 fcentrexl,fcentreyl:integer;
 fcentrexm,fcentreym:integer;

 ccentrexr,ccentreyr:integer;  {centre of box in contact-cut layer }
 ccentrexl,ccentreyl:integer;

 mcentrex,mcentrey:integer;    {centre of bpx in matel layer }

begin
  fcentrexl:=x+450;
  fcentreyl:=y-450;

  fcentrexm:=x+1650;
  fcentreym:=fcentreyl;

  fcentrexr:=x+2850;
  fcentreyr:=fcentreyl;

  ccentrexr:=fcentrexr;
  ccentreyr:=fcentreyr;

  ccentrexl:=fcentrexl;
  ccentreyl:=fcentreyl;

  mcentrex:=x+200;       { Here is only change made in even column }
  { only shift 200 unit }
  mcentrey:=fcentreyl;   { The rest are the same as before.}

  diffusion(fcentrexl,fcentreyl,fll,fwl);
  diffusion(fcentrexm,fcentreym,flm,fwm);
```

```
   diffusion(fcentrexr,fcentreyr,flr,fwr);

   contactcut(ccentrexr,ccentreyr);
   contactcut(ccentrexl,ccentreyl);

   metalone(mcentrex,mcentrey);

 end;

{ define onecolumn=array [1..7] of integer }
 procedure sevenbito(x,y:integer; data:onecolumn);
  const
   distance2row=1600;

  var              { This takes care of odd column data. }
   i:integer;

   begin
    for i:=1 to 7 do
     begin
       if data[i] = 0 then
          onebito(x,y);
       y:=y-distance2row;
     end;
   end;



 procedure sevenbite(x,y:integer; data:onecolumn);
  const
    distance2row=1600;
  var          { this takes care of even column data. }
   i:integer;

   begin
    for i:=1 to 7 do
     begin
       if data[i] = 0 then
          onebite(x,y);
       y:=y-distance2row;
     end;
    end;



{ define fourcolumn= array [1..4 1..7] of integer }
 procedure oneblock(x,y:integer; datas:fourcolumn);
  const
    distance2column=7800;
    distancenextcolumn=5400;
  var
    k,j,l,i:integer;
    data: onecolumn;
```

```
    temp:integer;

  begin
    temp:=x;
    for i:=1 to 2 do
      begin
        j:=2*i-1;    { take the odd number of column first. }
        for k:=1 to 7 do
           data[k]:= datas[j,k];
        sevenbito(temp,y,data);
        temp:=temp+distance2column;
      end;

    x:=x+distancenextcolumn;
    for i:=1 to 2 do
      begin
        j:=2*i;      { Take care of even number of column data }
        for k:=1 to 7 do
          data[k]:=datas[j,k];
        sevenbite(x,y,data);
        x:=x+distance2column;
      end;
    end;


{ define column32=array [1..32 1..7] of integer; }
 procedure eightblock(x,y:integer; matrix:column32);
  const
    distancexblock=19600;
    distanceyblock=12900;

  var
    j,i,k,l,x1,x2:integer;
    datas:fourcolumn;

  begin
    x1:=x;
    x2:=x+distancexblock;

    for i:=1 to 4 do
      begin
       j:=(i-1)*8;
       for k:=1 to 4 do
          begin
            for l:=1 to 7 do
              datas[k,l]:=matrix[j+k,l];
          end;
          oneblock(x1,y,datas);



       for k:=1 to 4 do
          begin
            for l:=1 to 7 do
```

```
            datas[k,l]:=matrix[j+k+4,l];
          end;
          oneblock(x2,y,datas);

       y:=y-distanceyblock;
      end;
    end;


 procedure fixfile;
 var i, ii:integer;

 begin

{ to transfer the data in the the metal file to poly file}

  reset(cml, 'metal.dat');
  while not eof(cml) do
    begin
      i:=0;
      while not eoln(cml) do begin
        i:=i+1;
        read(cml,string[i]);
      end;
      readln(cml);
      for ii:=1 to i do
        write(diff,string[ii]);
      writeln(diff);
    end;

{ To transfer the contact cut data to poly file. }

  reset(cc, 'contact.dat');
  while not eof(cc) do
    begin
      i:=0;
      while not eoln(cc) do begin
        i:=i+1;
        read(cc,string[i]);
      end;
      readln(cc);
      for ii:=1 to i do
        write(diff,string[ii]);
      writeln(diff);
    end;
  writeln(diff,'DF;');
{  writeln(diff,'E'); }

  {close(cml);
  close(cc);
  close(diff);
}
end;
```

```
procedure singleblock(arg: integer);
 var
  i: integer;
  x,y:integer;
  number,leng, cellnumber:integer;
  cellname: packed array [1..20] of char;
 begin

  readingfile(matrix);
  argv(arg+1, datafile);

{  write('Enter the original point (x,y): ');
  readln(x,y);
  write('Enter the cell DS # :');
  readln(number);  }


  x:= 0;
  y:=0;
  number := arg+stnum;
  intlength(number,leng);
  cellnumber := arg+cellnamenum;
{  write('Enter the name of the cell: ');
  i:=1;
  while not eoln do begin
  read(cellname[i]);
  i:=i+1;
  end;
   i:=i-1;
}
{ open three files to store three different layers ie. diff,cc,cm }
  rewrite(diff, datafile);
  rewrite(cml, 'metal.dat');
  rewrite(cc, 'contact.dat');
{ write down the layer name }
  writeln(diff,'DS ',number:(leng-2),' 1 1;');
  writeln(diff,'9 CEL_',cellnumber:3,';');
  writeln(diff,'L CF;');
  writeln(cml,'L CM;');
  writeln(cc,'L CC;');

{ carry the actual work on the ROW }
  eightblock(x,y,matrix);
{  close(cml);i}

{ merge the three files into a single file. }
  fixfile;
{
  writeln(diff, 'C ',number:(leng-1),';');
  writeln(diff,'E');
}
 end;
```

```
begin
  argv(1, datafile);
  reset(infile, datafile);
  readln(infile, numfile);
  writeln('Enter the starting cell number #: ');
  readln(stnum);
  writeln('Enter the starting cellname number CEL_#: ');
  readln(cellnamenum);
  stnum := stnum - 1;
  for i:=1 to numfile do begin
    singleblock(i);
  end;
end.
```

```
#include <stdio.h>
main()
{ FILE *outfile;
  int mod;

  printf("\nEnter the modulo number:==> ");
  scanf("%d", &mod);
  putchar('\n');

  outfile = fopen("adder.dat", "w");
  fprintf(outfile,"%d\n", mod);
  fprintf(outfile,"a\n");
  fclose(outfile);
  outfile = fopen("substractor.dat", "w");
  fprintf(outfile,"%d\n", mod);
  fprintf(outfile,"s\n");
  fclose(outfile);
  outfile = fopen("modnumber.dat", "w");
  fprintf(outfile,"%d\n", mod);
  fclose(outfile);
}
```

```
#include <stdio.h>
#include <string.h>
main(argc, argv)
int argc;
char *argv[];

{  int i, ii;
  char  ch, pwd[100], logindir[40], fixname[40];
  FILE *outfile, *infile;

  if (argc < 2 )
   {
    printf("Usage: program filename\n");
    exit(1);
   }

  if ( getwd(pwd) == NULL)
  {
    printf("Error in reading your current directory!\n");
    printf("_Exit\n");
    exit(1);

  }


   i= 7;
   while(pwd[i] !=  '/' && pwd[i++] !='\0' ) ;
        i--;
   for (ii=0; ii<=i; ii++)
      logindir[ii] = pwd[ii];
  logindir[++i] = '\0';
  printf("%s\n", logindir);

  printf("%s\n", pwd);
   i=0;
   while(logindir[i++] != '\0');
   logindir[--i] ='/';
   i=0;
   while(pwd[i++] != '\0');
   pwd[--i] ='/';
   strcat(fixname, logindir);
   strcat(fixname,".fix.mac");  /* create file loginname/.fix.mac */

   strcat(logindir, ".cadrc");  /* create file loginname/.cadrc  */

   strcat(pwd, argv[1]);
   infile = fopen(fixname, "r");
   outfile = fopen(logindir, "w");
/* transfer all the basic commands from .fix.mac file to new .cadrc */

   while( (ch = getc(infile)) != EOF)
        putc(ch, outfile);
   fclose(infile);
```

```
    fprintf(outfile,'electric cifin %s\n', pwd);
    fprintf(outfile,'electric editcell chip\n');

    fclose(outfile);


}
```

```
/* ----------------------------------------
  This program find the user login directory
  and copy .fix.mac file to .cadrc file in the login directory
  . Hence , the user can issue this program in any diectory he is
  currently work */
#include <stdio.h>
#include <string.h>
main()

{  int i, ii;
  char pwd[100], logindir[40], fixname[40], move[150];


  if ( getwd(pwd) == NULL)
  {
    printf("Error in reading your current directory!\n");
    printf("_Exit\n");
    exit(1);

  }


   i= 7;
   while(pwd[i] !=  '/' && pwd[i++] !='\0' ) ;
        i--;
   for (ii=0; ii<=i; ii++)
      logindir[ii] = pwd[ii];
   logindir[++i] = '\0';

   i=0;
   while(logindir[i++] != '\0');
   logindir[--i] ='/';
   i=0;
   while(pwd[i++] != '\0');
   pwd[--i] ='/';
   strcpy(move, "cp ");
   strcat(fixname, logindir);
   strcat(fixname,".fix.mac");  /* create file loginname/.fix.mac */

   strcat(logindir, ".cadrc");  /* create file loginname/.cadrc  */

   strcat(move, fixname);
   strcat(move, " ");
   strcat(move, logindir);
 /* move = " cp fixname logindir" */
 /* i.e.: cp /users/yeung/.fix.mac /usrs/yeung/.cadrc  */
   system(move);  /* issue a command = cp fixname logindir */


}
```

```
# create those adder.dat, substractor.dat and modnumber.dat for modulo num.
modnum
# create adder bit data
gen_data bit01 < adder.dat
masknew bit01 c1 c2 c3 c4 c5 < cellnum21
# create substractor data
gen_data bit01 < substractor.dat
masknew bit01 cc1 cc2 cc3 cc4 cc5 < cellnum26
# create addresscontent data
multaddss bit01 < modnumber.dat
masknew bit01 ccc1 < cellnum31
# layout the floor planning for the multipler
petermult $1 cif.cif c1 c2 c3 c4 c5 cc1 cc2 cc3 cc4 cc5 ccc1
rm c1 c2 c3 c4 c5 cc1 cc2 cc3 cc4 cc5 ccc1
rm bit01
```

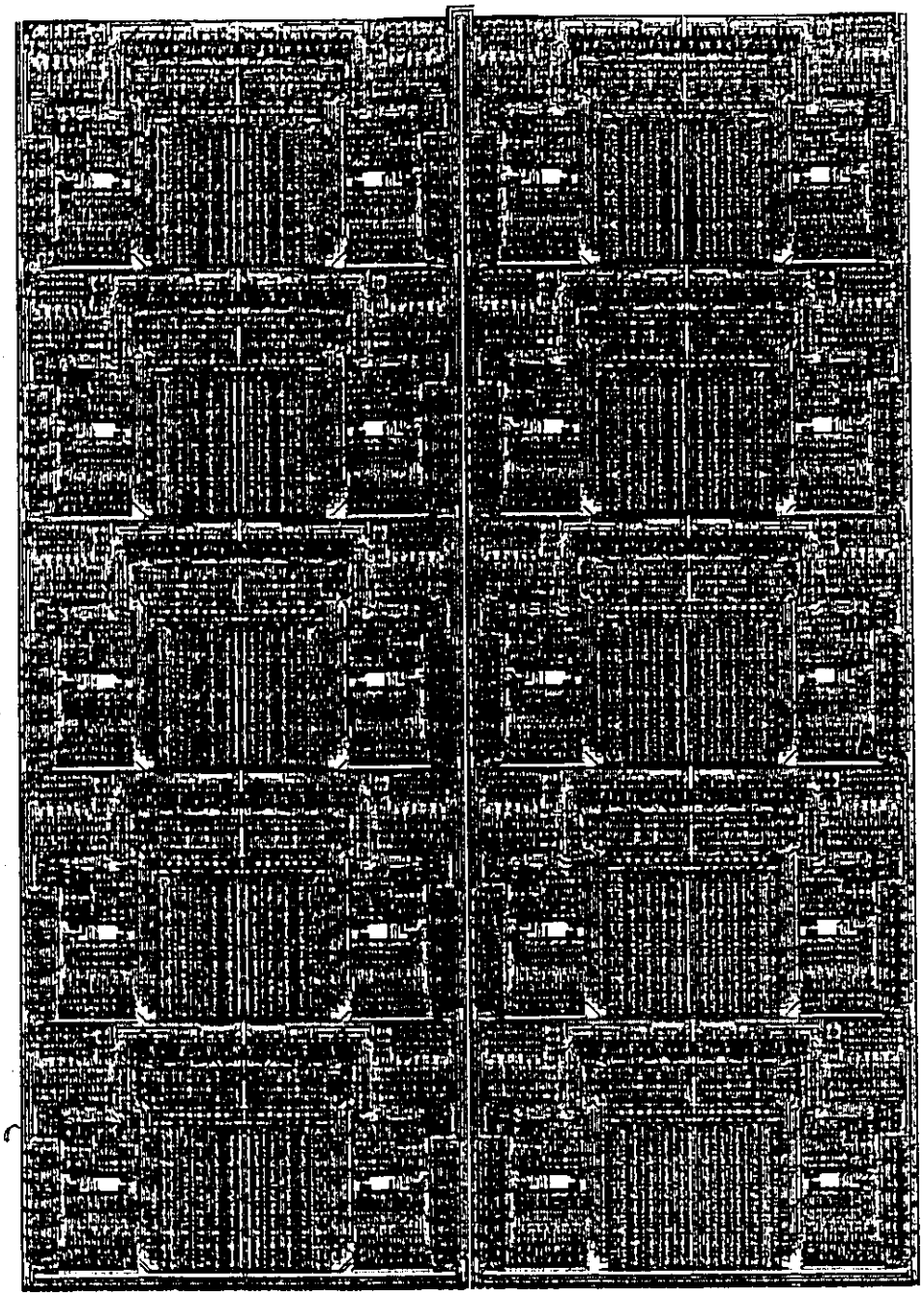
```
# one_data.com output file
#
# create numerical data
gen_one_data dummy11
# create the mask data from the dummy11
masknew dummy11 $1 < .onedata
# append the 'end' statment to the output file
cat .endata >> $1
# remove the dummy11 file
rm dummy11
```

```
# create a adder/substractor, anser the question of modulo number
gen_data bit01
systolic bit01 c1 c2 c3 c4 c5 < cellnum16
merge $1 rajaoneftr.cif c1 c2 c3 c4 c5
rm bit01
rm c1 c2 c3 c4 c5
rm contact.dat metal.dat
```

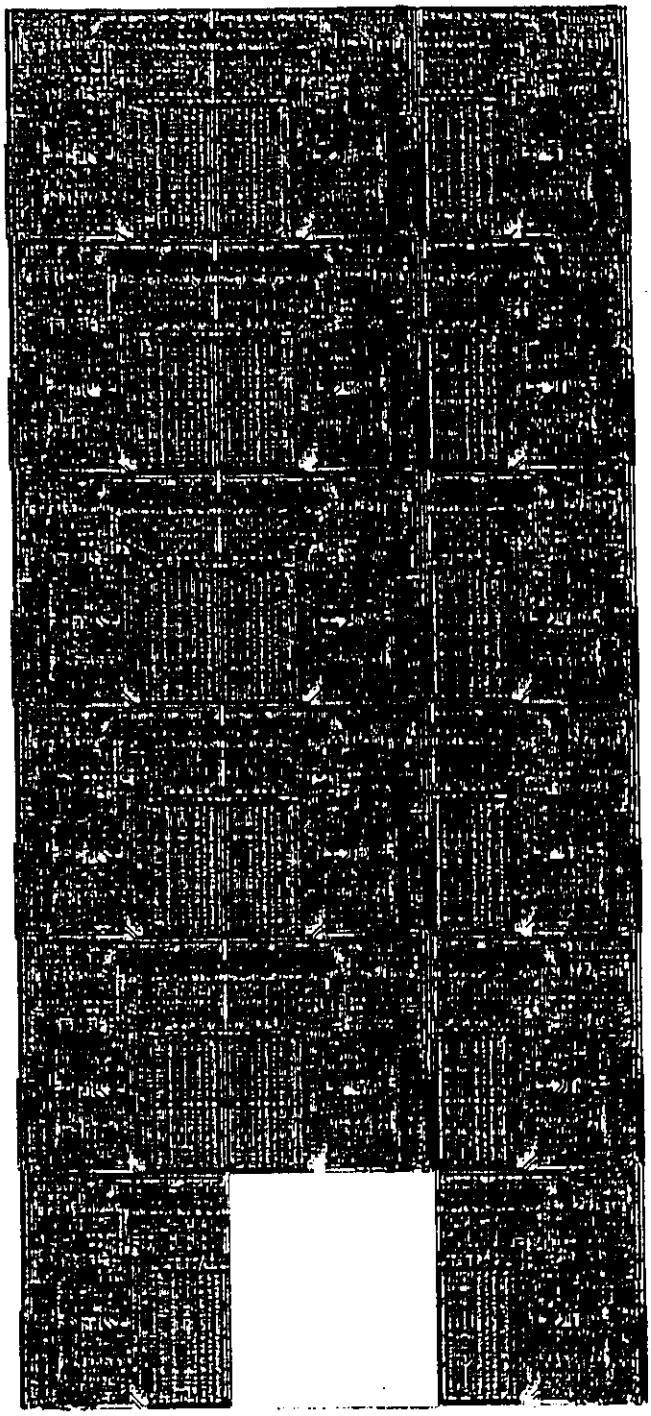
```
# create adder data
gen_data bit01 < adder.dat
systolic bit01 c1 c2 c3 c4 c5 < cellnum16
# create substractor data
gen_data bit01 < substractor.dat
systolic bit01 cc1 cc2 cc3 cc4 cc5 < cellnum21
# create addresscontent data
multaddss bit01 < modnumber.dat
systolic bit01 cccl < cellnum26
# layout the floor planning for the multipler
mergemult $1 rajaoneftr.cif c1 c2 c3 c4 c5 cc1 cc2 cc3 cc4 cc5 cccl
rm c1 c2 c3 c4 c5 cc1 cc2 cc3 cc4 cc5 cccl
rm bit01
rm contact.dat metal.dat
```

Appendix IX

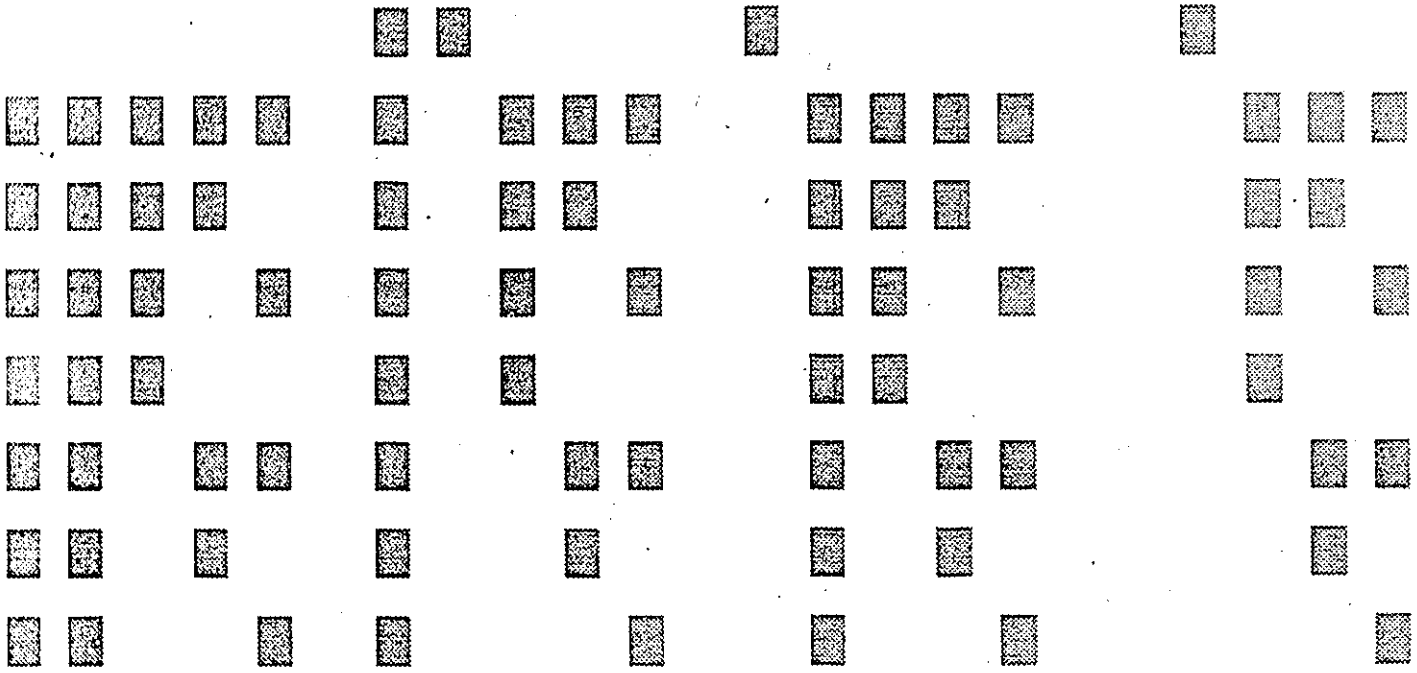
Design Examples of MOSC



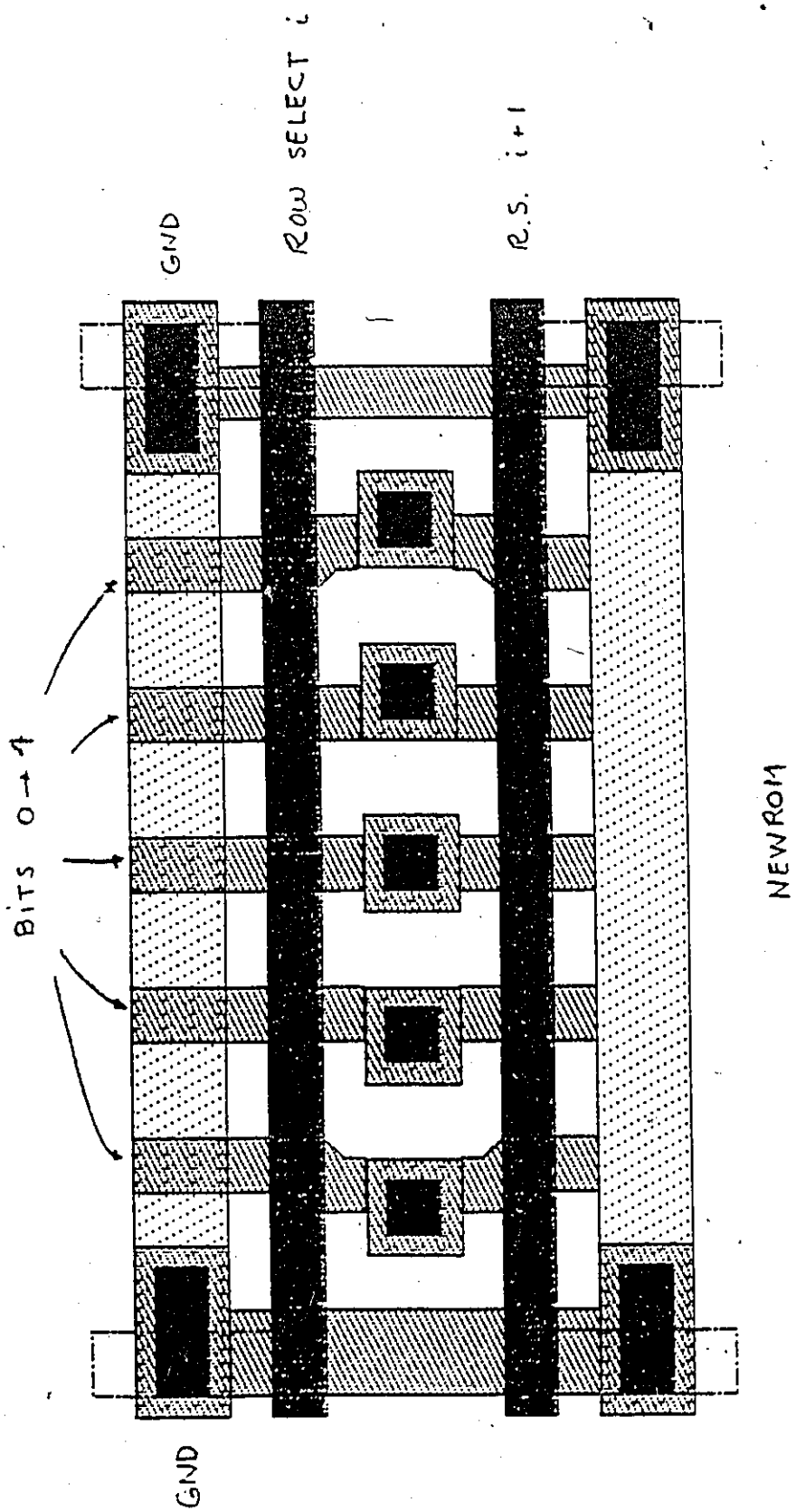
Mask layout of the 4-adder circuit



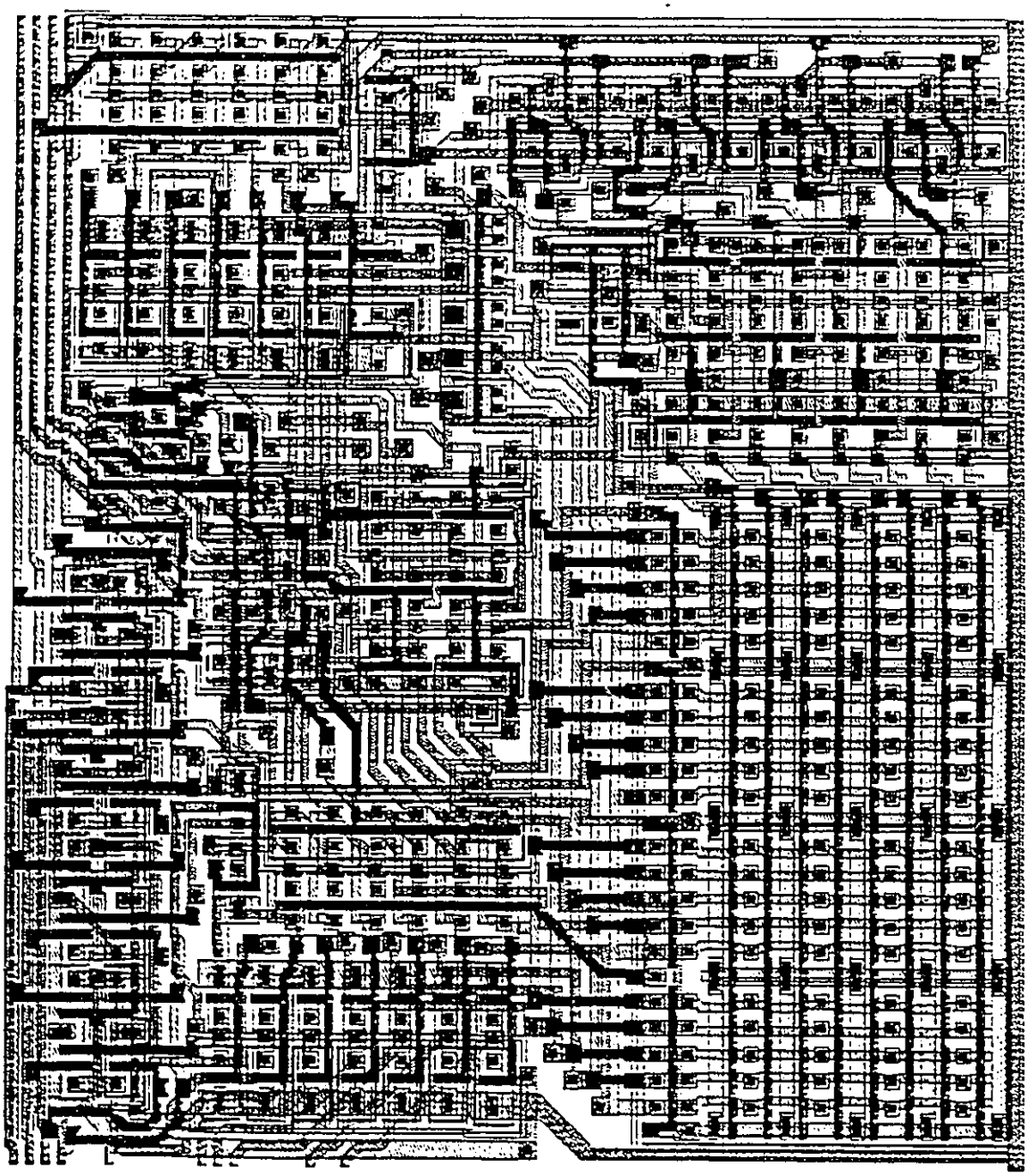
Mask layout of the multiplier circuit



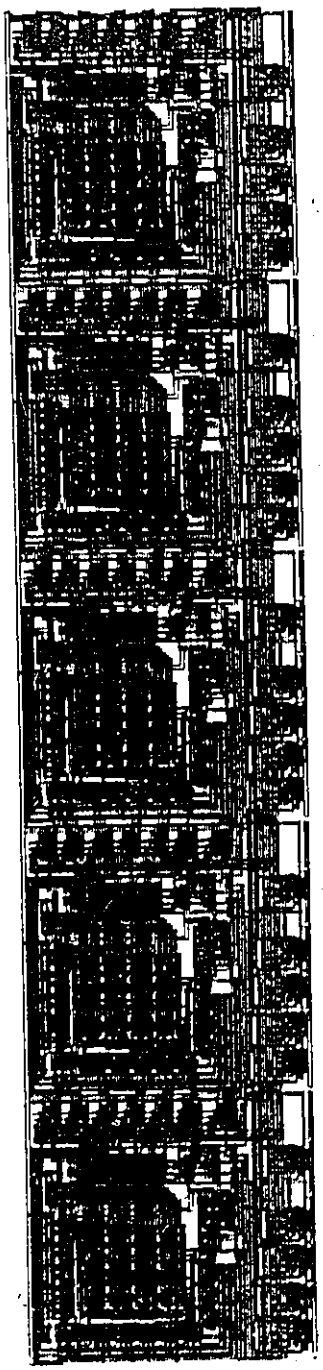
Mask layout of the memory content of the 1st cell



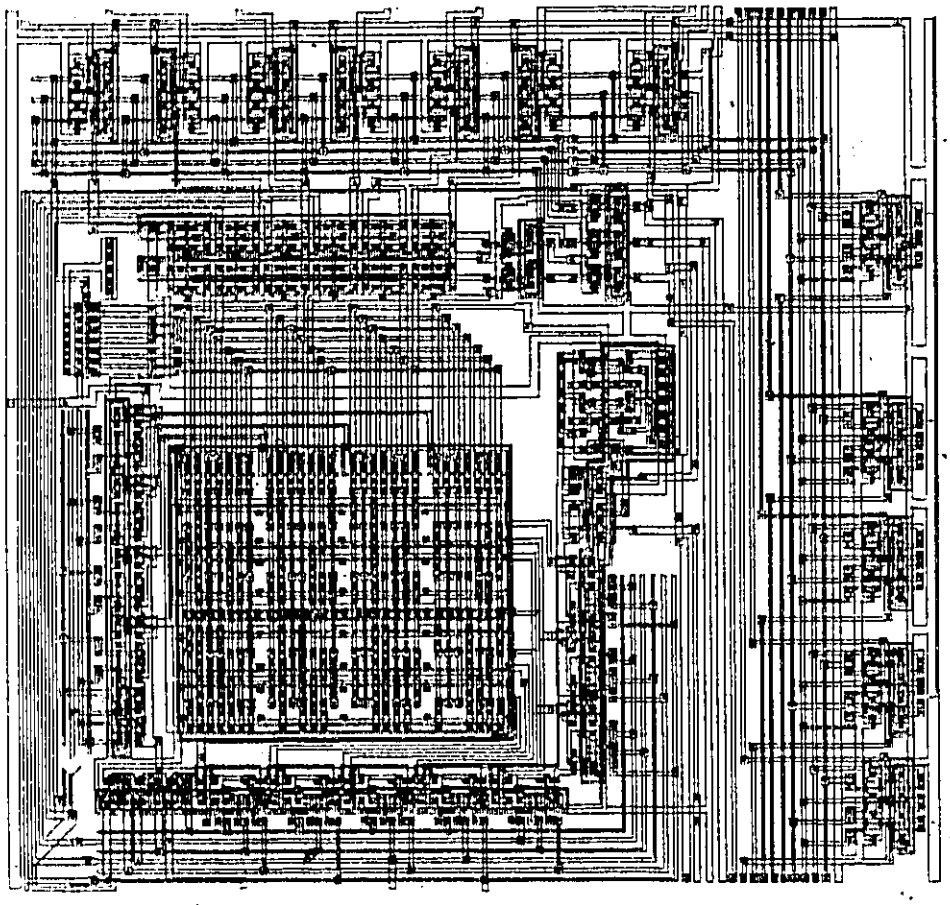
Mask layout of two rows of memory locations



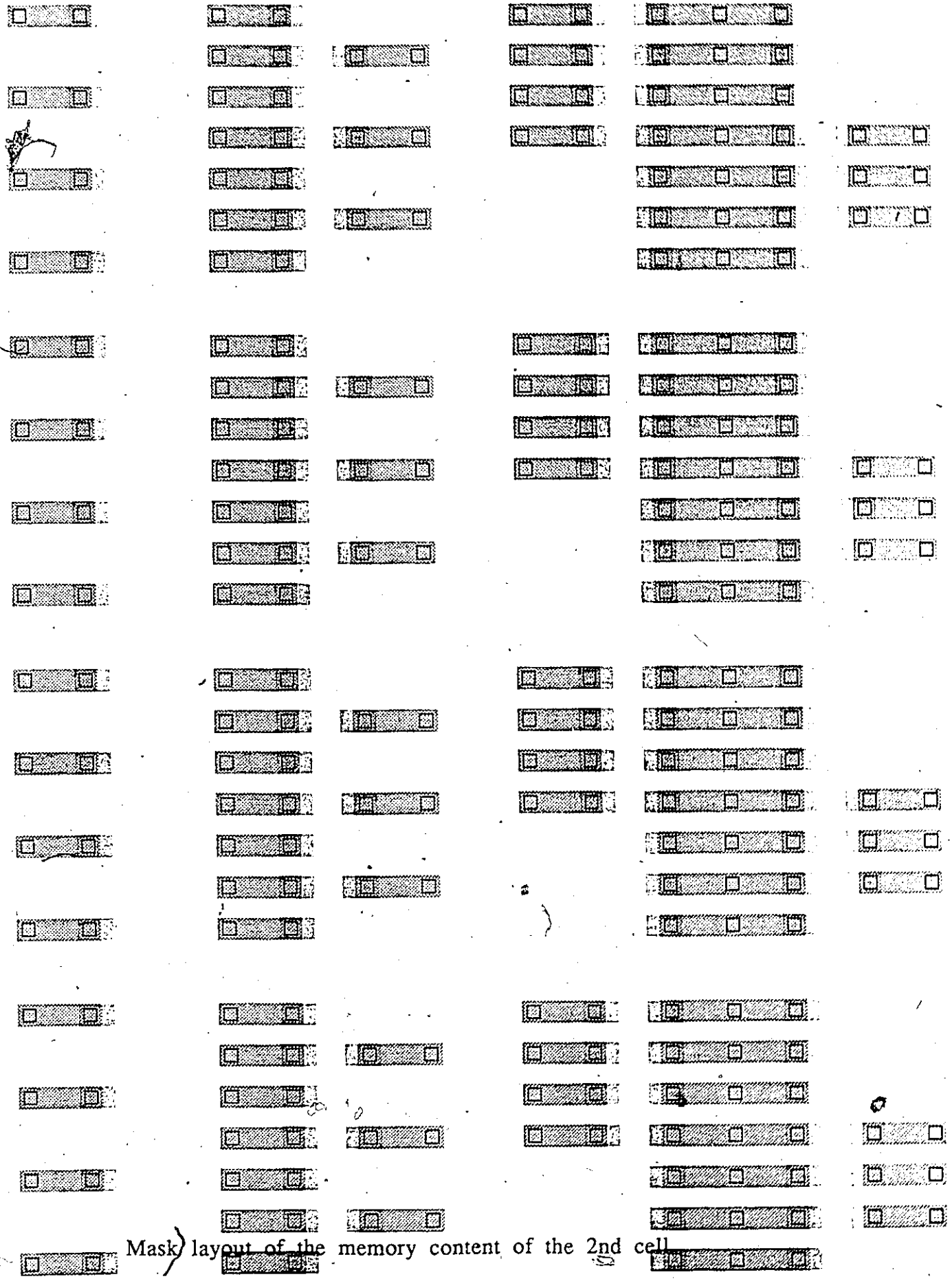
Mask layout of a single cell generated in Daisy



Mask layout of a single adder circuit



Mask layout of the second memory oriented cell



Mask layout of the memory content of the 2nd cell

VITA AUCTORIS

Date of birth: September 10, 1961

Place of birth: Hong Kong

1986: B.A.Sc. in Electrical Engineering, University of Windsor

1988: Candidate for M.A.Sc. in Electrical Engineering at University of Windsor.