INVESTIGATION INTO METHODS AND ANALYSIS OF COMPUTER AIDED DESIGN OF VLSI CIRCUITS

J. A. NOONAN, B. E.(Hons.)

A thesis submitted for the Degree of Master of Engineering Science

in

The Department of Electrical and Electronic Engineering,

The University of Adelaide.

Awarded 1987

TABLE OF CONTENTS

Synopsis

This thesis analyses a range of Computer Aided VLSI design tools that have been developed at the University of Adelaide. The work reviews different IC technologies and summarises some of the overall philosophies behind the development of VLSI CAD tools. In addition, Multi Project Chip processes and their relationship to various CAD tools is examined and thoughts offered on the merits of full custom IC design as opposed to semi-custom design techniques. The work for this thesis has been completed over a four and a half year period beginning in early 1982, when MPC processes were first introduced to Australia. As a result, the comments made on VLSI design software and MPC processes are strongly associated with the problems experienced with these topics in the Australian environment.

An investigation of a *mask level* CAD tool set is made and a summary of the tools comprising the complete set, by comparison with some early generation tools that originated from other universities such as USC Berkeley and MIT, is then offered. The work also investigates a symbolic level CAD tool set, the VIVID system, which originated from the Microelectronics Center of North Carolina to run under the UNIX operating system. VIVID was developed for the VMS operating system by ongoing research at the University of Adelaide. In addition, other tools that are generally classified as higher level tools than either mask or symbolic level tools are also discussed for the purposes of comparing the directions of tool development.

The thesis presents a detailed analysis of the tools discussed by presenting one complete nMOS MPC design experience in detail (including testing of the fabricated product), and providing information on a second design experience for the purposes of comparison of the layout tools discussed. The results of this work also allow comparison of two different Australian MPC processes using the nMOS technology as both designs discussed were fabricated using different MPC processes.

A similar account of a detailed design experience using the CMOS technology in conjunction with the symbolic level tools is presented and new Australian CMOS MPC process foreshadowed. This experience, considered with the earlier mask level design experiences, provides the basis for some final comparitive results to be made between the relative efficiency of designing IC's with low level and high level tools. As a result, the work finally presents conclusions regarding the future development of VLSI CAD tool sets with particular reference being made to Silicon Compilation.

The thesis tries to draw an analogy between levels of VLSI CAD tool sets and levels of languages used to program computers in an effort to indicate possible future directions for development of these tools. In particular, the thesis examines reasons why the symbolic level tools may prove to be the first step on the way towards making true Silicon Compilation a reality.

Two different IC technologies, nMOS and CMOS, are used to allow the close analysis and investigation of these tool sets. A number of circuits have been fabricated using the tools and Australian MPC processes and the simulated and tested results of the designs are used to verify the conclusions regarding the tools.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university, and to the best of my knowledge and belief, contains no material previously published or written by any other person, except where due reference is made in the text.

J. A. Noonan

Acknowledgements

In the production of this thesis I must thank my supervisor, Dr. Kamran Eshraghian, for providing the boundless enthusiasm he constantly brims with. This enthusiasm has served as a source of encouagement to continue on with a very complex task. I also owe a great debt of thanks to Mr. Michael Liebelt who provided the highest quality technical assistance and at times helped me to solve seemingly insurmountable problems, not the least of which was the proof reading of this thesis. Thanks also must be offered to Theo Kermanidis for assisting with the proof reading. I would also like to say what a pleasure it has been working with the Electrical and Electronic Engineering Department staff and researchers at Adelaide University.

Finally, I would like to thank Andrew Trevorrow for his patience and assistance in helping me create this thesis using Donald Knuth's TEX typesetting system. I will find it hard ever to use pencil and paper again.

J. A. Noonan.

CHAPTER I

IC TECHNOLOGIES AND DESIGN

1.1 Introduction to IC's and a VLSI Technology Choice

Since the invention of the first transistor by Bardeen, Brattain and Shockley in December 1947 at Bell Telephone Laboratories, Murray Hill, N.J.[1][3], Microelectronics has progressed rapidly. Perhaps the best example of this progress is indicated by the attempts to classify the progress in computer technology.

Computers have so far been classified into four distinct generations, with the beginnings of a fifth currently being researched[2]. The first four generations are all characterised by their basic building blocks, (1) the ***vacuum tube***, (2) the ***transistor***, (3) the *Integrated Circuit* (**IC**) and (4) the *Large Scale Integrated* (**LSI**) or *Very Large Scale Integrated* (**VLSI**) Circuit. The fifth generation on the other hand is characterised rather by advances in software, in particular advances in *Artificial Intelligence* (**AI**)[2].

Integrated Circuits are combinations of basic circuit elements , such as diodes, transistors, capacitors and resistors interconnected on a common substrate or base material. They are fabricated by various processing techniques allowing simultaneous formation of large numbers of these devices. Development of the planar process techniques and IC's did not begin until 1958, 10 years after the invention of the transistor[3]. With these first IC's came the realisation of their potential and rapid development followed[3.1]. G. E. Moore, President of INTEL Corp., produces excellent plots of statistics which indicate accurately the precise extent of these developments. Fig 1.1 relates the number of components per chip to time since the beginning of the semiconductor industry, which is one simple measure of IC development.

The varying scales of integration are themselves classified into four categories. The invention of the IC in 1958 heralded *Small Scale Integration* (**SSI**- fewer than 100 transistors per chip) and the first commercial IC's entered the market in 1960 with a Minimum Feature Size of $25\mu m$[4]. Minimum Feature Size has progressively shrunk to the *Medium Scale Integrated* (**MSI**- less than 1000 transistors per chip) $10\mu m$, the early *Large Scale Integrated* (**LSI**- less than 10,000 transistors per chip) $5\mu m$, and now onto the *Very Large Scale Integrated* (**VLSI**- greater than 10,000 transistors per chip) $1\mu m$ and expected sub micrometer levels[5].

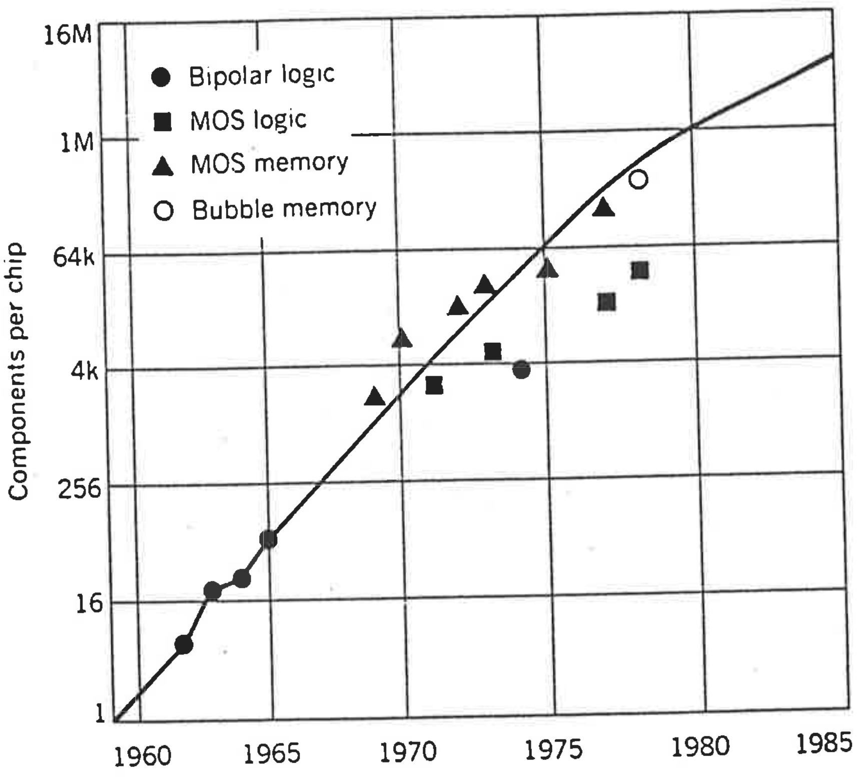


Figure 1.1: Gordon Moore, President of INTEL Corp., has produced much graphical evidence to support his predictions. Here is a plot he produced of Number of Components per Chip vs Time Since the Beginning of the Semiconductor Industry, produced in 1979.

Minimum Feature Size is commonly quoted as it is an indication of the sophistication of a particular technology. This measure describes the minimum width of any of the mask level geometries for a particular technology, see Appendix A. It is expected that the maximum scale of integration will be reached when Minimum Feature Size reduces to approximately the 0.1 μm level using currently favoured IC technologies[71].

A number of different types of IC structure exist. These include Thin-Film and Thick-Film IC's, Monolithic IC's, and Hybrid IC's. The advances in integration to the VLSI level have all been made using monolithic IC's.

Monolithic IC's begin with an extremely pure ***monolithic*** (single crystal) semiconductor base , which could be Germanium, Silicon, or Gallium-Arsenide, but is usually Silicon. This base is exposed to a number of chemical deposition, etching, and diffusion phases, patterned by photolithographic masking techniques. The result of this complex sequence of materials-processing steps is a layered, three dimensional circuit atop and within the monolithic base. The specific structural features required for IC's include the basic circuit elements, transistors, diodes, capacitors, resistors as well as isolation and interconnection. The PN junction is the basis of operation of all semiconductor devices with the two most important active devices being, (1) the *Metal-Oxide Semiconductor* (**MOS**) or Insulated Gate *Field Effect Transistor* (**FET**) and (2) the *Bipolar Junction Transistor* (**BJT**).

The mask patterns, see Fig. 1.2, forming IC layouts are prepared in arrays of squares with dimensions typically in the order of $5mm$ by $5mm$. This array of mask patterns is designed to cover the whole area of a semiconductor wafer with a diameter in the order of 3 to 4 inches. The individual squares are referred to as *dice* (**chips**). Each *die* (**chip**) carries the designed circuit and can be separated and packaged individually to perform its required function, see Fig. 1.3.

There are many different technologies available for use in fabricating monolithic IC's. Bipolar or MOS technologies, *Transistor Transistor Logic* (**TTL**), *Integrated Injection Logic* (**I^2L**), *Emitter-Coupled Logic* (**ECL**), *n-channel* or *p-channel* MOS (**nMOS, PMOS**), *Complementary* MOS (**CMOS**), or *Silicon On Insulator* (**SOI**) MOS IC's to name some of the more common technologies. The circuits that can be designed using these technologies are advantageous because of their low power consumption, small physical size and resultant low weight and cost[9]. Not all of these are suitable for VLSI design. Factors that must be considered in choosing a particular

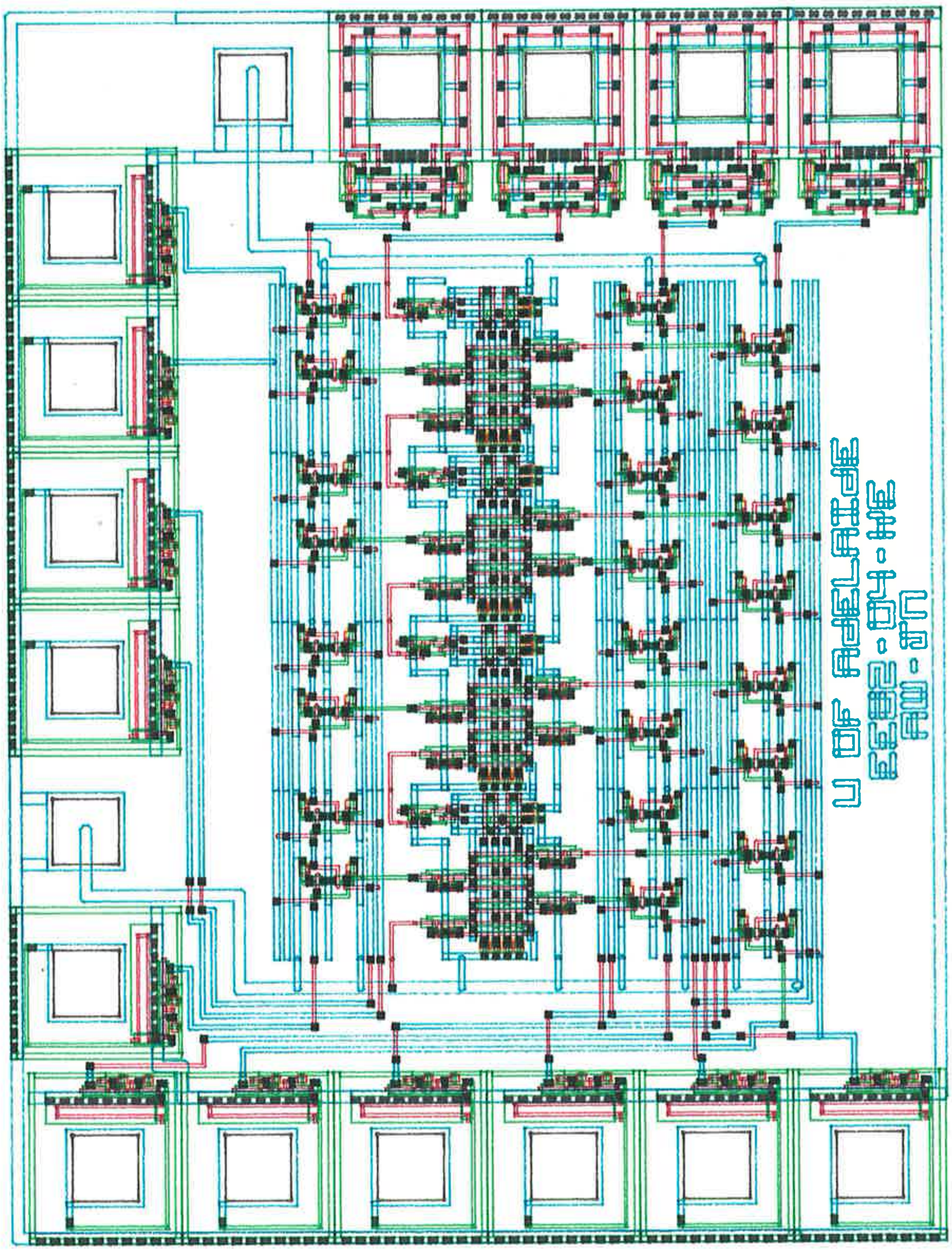
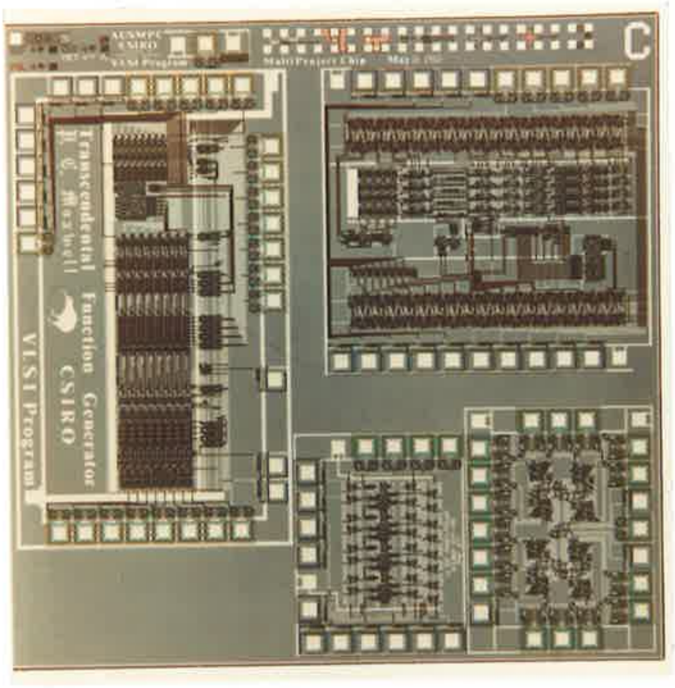


Figure 1.2 (a): A computer generated plot of the mask level layout of project C4, an nMOS MPC design fabricated on AUSMPC May 1982. C4 implies project number 4 on die C. Included is a photomicrograph of project C4 on die C.



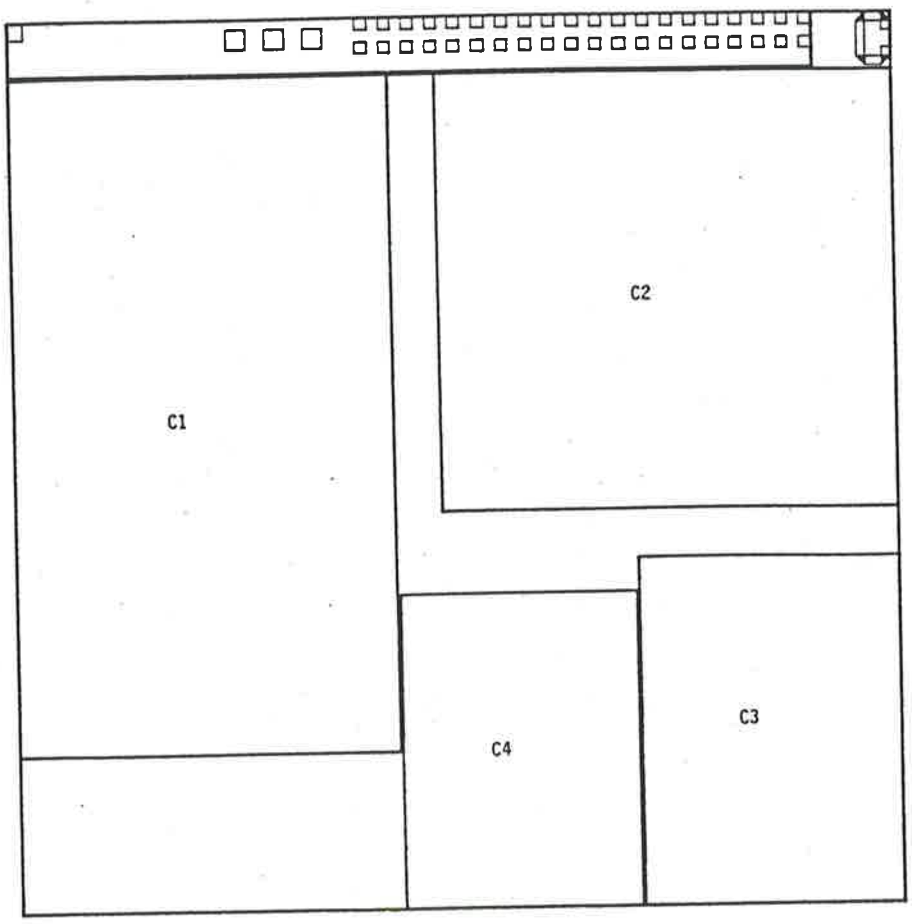


Figure 1.2 (b): A photomicrograph of the entire die C, and a map of die C indicating the position of each of the projects on die C.

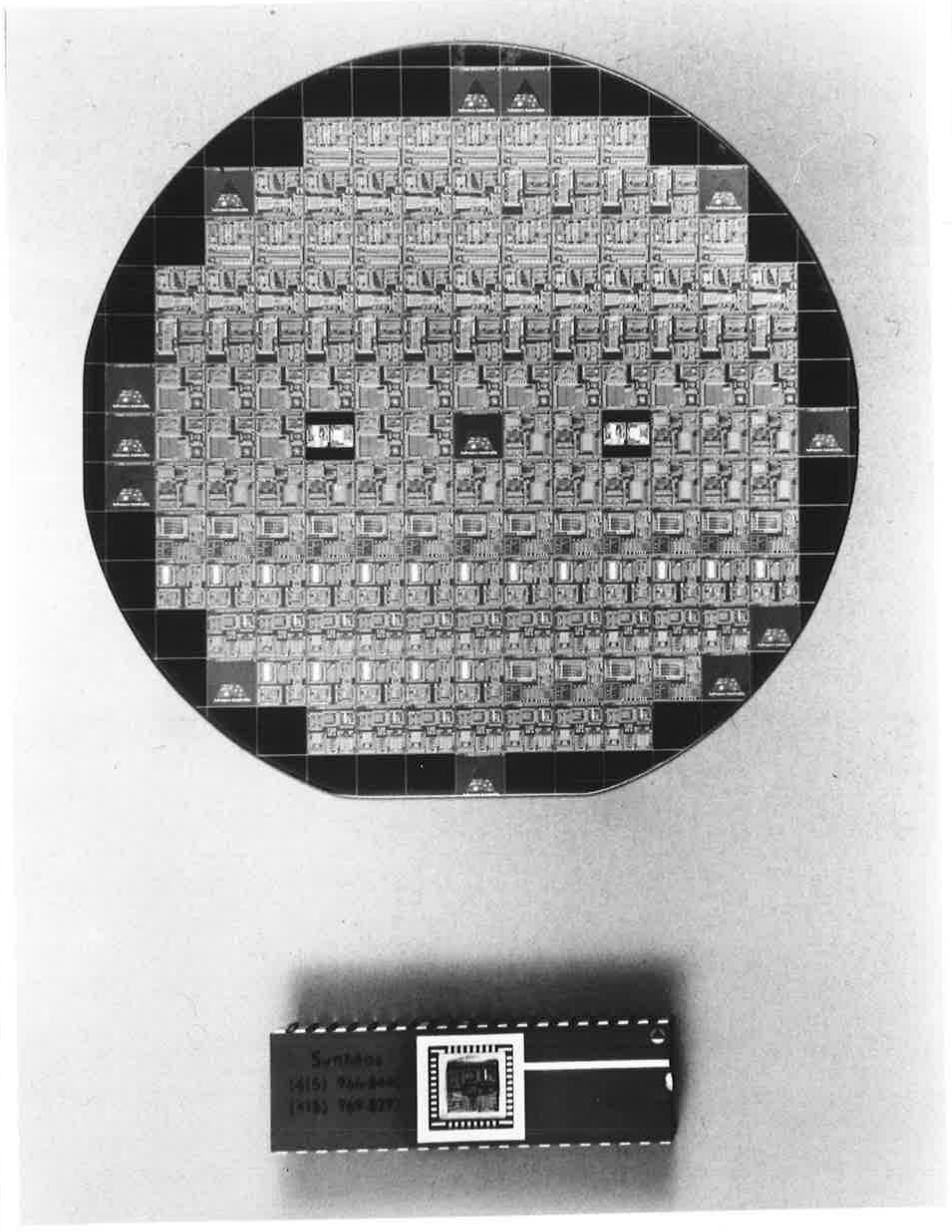
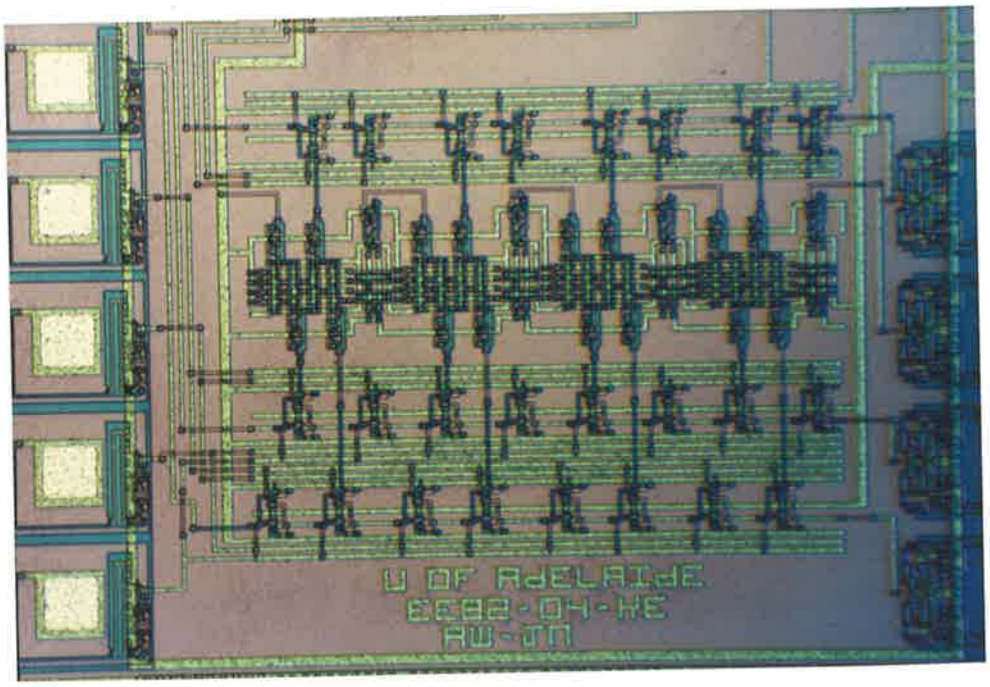


Figure 1.3 (a): A photograph of the 4 inch AUSMPC May 1982 Multi Project Wafer and a 40 pin package with the lid off, exposing the bonded die. The bonded die can actually be recognised as die H on the wafer, using the map in Fig. 1.3 (b) and the photograph of the wafer above.



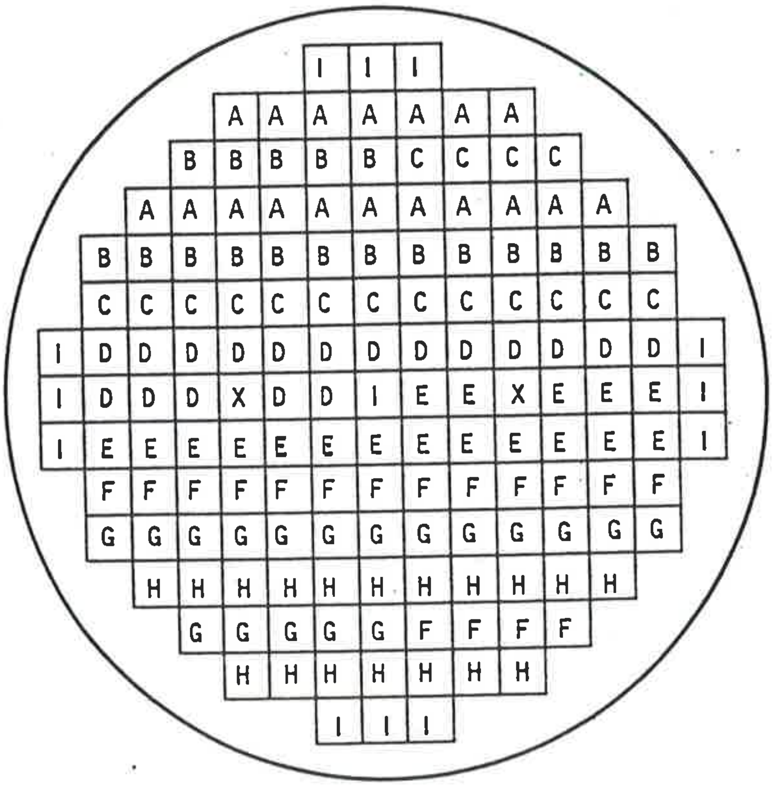

I I I
A A A A A A A
B B B B B C C C C
A A A A A A A A A A A
B B B B B B B B B B B B B
C C C C C C C C C C C C C
I D D D D D D D D D D D D D I
I D D D X D D I E E X E E E I
I E E E E E E E E E E E E E I
F F F F F F F F F F F F F
G G G G G G G G G G G G G
H H H H H H H H H H H
G G G G G F F F F
H H H H H H H
I I I

Figure 1.3 (b): A map of the 4 inch AUSMPC May 1982 wafer and a close up of an unbonded version of project C4.

technology for design are circuit density, circuit functionality, power consumption and performance, topological properties of the circuit for interconnection, suitability for total system implementation, and availability of processing facilities[8].

These points considered, the MOS technologies, in particular nMOS and CMOS are attractive. MOS devices are particularly well documented and easy to design, as long as the designer uses a set of design rules and stays away from the smallest geometries feasible[7]. With a relaxed set of design rules, the performance of standard MOS circuit blocks are as predictable as TTL circuits[7][8]. Finally, both nMOS and CMOS circuits can be fabricated in Australia. As a result, the two monolithic IC technologies used to illustrate the ***Computer Aided Design*** (**CAD**) systems discussed in this text will be nMOS and CMOS. It should be understood at this point however that while these CAD systems are discussed in relation to these two technologies, most of the CAD tools are easily modified to handle new technologies as they become available.

1.2 The VLSI Design Philosophy

Just as programming was done throughout the 1950's by small groups of highly qualified people solving problems in their own style, IC design in the 1970's was still done by small groups of "layout wizards"[7][12]. The advances in computer technology have however introduced powerful aids for design, and with the passage of time IC processing techniques are beginning to become reasonably standardised. As a result, IC design is very quickly becoming a routine engineering step in the development of a special purpose system[7].

As IC process technology has moved from the LSI to the VLSI generation, new problems for the circuit designer have rapidly emerged. These problems can perhaps be most conveniently grouped under the one heading, Complexity Management[5][11][12]. An excellent understanding of these problems can be gained by considering a layman's analogy as presented by Moralee[5] relating geographical feature sizes to circuit feature sizes.

This analogy relates a $1km$ square street map of London to the complexity involved in a $1mm$ square die size, $25\mu m$ feature size SSI chip. Moralee then indicates that a typical LSI $5mm$ square die with a $5\mu m$ feature size, is of similar complexity to a $25km$ square street map of London, or almost the whole of the Greater London area covered by the A-Z book of maps. While it is possible for anyone to hold in their minds the details of a $1km$ square street map of London, only a London taxi driver could ever be expected to keep mental track of a street network 625 times the size.

The LSI designer is faced with a similar daunting task if he is to know his circuit layout. The VLSI designer however is confronted by die sizes of around $1cm$ square with feature sizes of around $1\mu m$ which translates to a map covering a $250km$ square, or a map which would fill 100 volumes of the A-Z book of maps and cover most of the southern half of England. Even this scale of complexity is dwarfed by the expected mature VLSI devices of the late 1980's with die sizes of $2cm$ square and feature sizes of $0.25\mu m$, or a map of a street network which would be $2000km$ square, covering practically the whole of central and western Europe. Obviously, a "layout wizard" becomes lost at the LSI stage of design and only new design philosophies developed specifically to handle such complexity will provide solutions to the problems presented by LSI and VLSI design.

Mudge[11] indicates clearly some paths towards the solution of this complexity problem by the implementation of a structured design methodology, using a coordinated design team aided by sophisticated (CAD) tools. Sequin[12] provides a much more detailed scientific analogy to the VLSI design complexity problem. This analogy was touched on by Mudge[11] also. Sequin relates in detail the VLSI design complexity problem to the complexity problem faced by software engineers when unstructured programs started to grow to lengths in excess of 10,000 lines of code. The software complexity problem was alleviated by developing and adopting suitable design methodologies, structuring techniques and documentation styles. Many of the lessons learned from this work are also applicable in the VLSI system design domain, although the VLSI designer faces even more problems than encountered in the software design domain.

Sequin[12] concisely sums up the VLSI design process with his three conceptual concerns:-

1 Functional design: guaranteeing proper behaviour.

2 Implementation: finding a suitable structure.

3 Optimization: fine tuning the physical arrangement.

It is important to note that these three steps in the design process may need to be iterated through a number of times to guarantee that the high level design decisions made in step 1 can be implemented with the available medium in 2.

Design methodologies are continually being developed to fit these steps in the design process. Sequin proposes that the more innovative methodologies so far developed result in design systems primarily produced by people with a strong background in Computer Science[12]. It is important to note that these innovative design systems in some cases also owe their origins to people with backgrounds at least as strong in Electrical Engineering[13], as in the case for Weste and Ackland's **MULGA** system and subsequently developed symbolic layout[13.1][14] systems.

The trend in the development of these methodologies centres around a desire to design fully custom developed IC's as opposed to semi-custom devices. However, because of the restrictions on the sets of design rules that can be used with such design systems, it is found that the standard processes commonly used for fabricating custom chips for remote designers can involve as much as a two-generation (factor of four) deficiency in performance [11] as compared to the leading edge processes used by "in house" designers. Subsequently, as Sequin[12] indicates, when rising development costs for VLSI systems become comparable to the fabrication costs of tens of thousands of devices, a faster development cycle at the expense of a less efficient implementation becomes an attractive alternative for many systems. Hence the increasing popularity of gate arrays and other semi-custom alternatives to LSI and VLSI design such as Standard Cell systems.

Semi-custom as opposed to *Full Custom* design offers a "TTL Databook" type approach to system design using gates and library cells. Generally semi-custom IC design limits the number of layers processed by the designer in order to reduce the complexity faced by the designer. In the case of gate arrays, a pre-processed wafer is reconfigured using only a few mask levels. In the case of Standard Cell systems, apart from the reduced number of layers being processed, pre-defined and pre-laid out function blocks are used, further decreasing the design complexity faced by the designer. These pre-defined blocks are connected using routing and placement routines. Both of these approaches to VLSI design allow the designer to customize an underlying matrix of cells. In the case of gate arrays, these cells are basic, while more complex cells up to the MSI scale, are used in the case of Standard, or Library cell systems[14.1][14.2]. In both the Standard Cell and Full Custom VLSI design methodologies, an entirely new, full set of masks are produced to process an uncommitted wafer, however, Standard Cell design remains less efficient than full custom design in terms of circuit density and performance[13.1][14.2]. The ultimate VLSI design methodology will revolve around a single tool which will comprise many other tools. That tool will be a Silicon Compiler. A Silicon Compiler will take a high level system description and automatically translate it into correct circuit implementations in a particular technology. According to Sequin[12], such tools are still in their infancy and it should not be expected that correct, economically viable solutions for arbitrary systems for all combinations of input parameters will be found before the end of this century. However, early Silicon Compilation tools based on the full custom approach to VLSI design as applied to highly constrained architectures are showing promise, notably Macpitts[14.3]. Fig 1.4 gives a Macpitts example and is inserted at this point to briefly introduce the characteristics of such tools to the reader. Weste[48] explains that most current Silicon Compilers are still characterised by a fixed floor plan. This leaves such tools with no ability to deal with the complexity involved in contemporary full custom IC's. It is expected that, in the future, silicon compilers will synthesise floor plans and circuits in the same manner as a human designer[48].

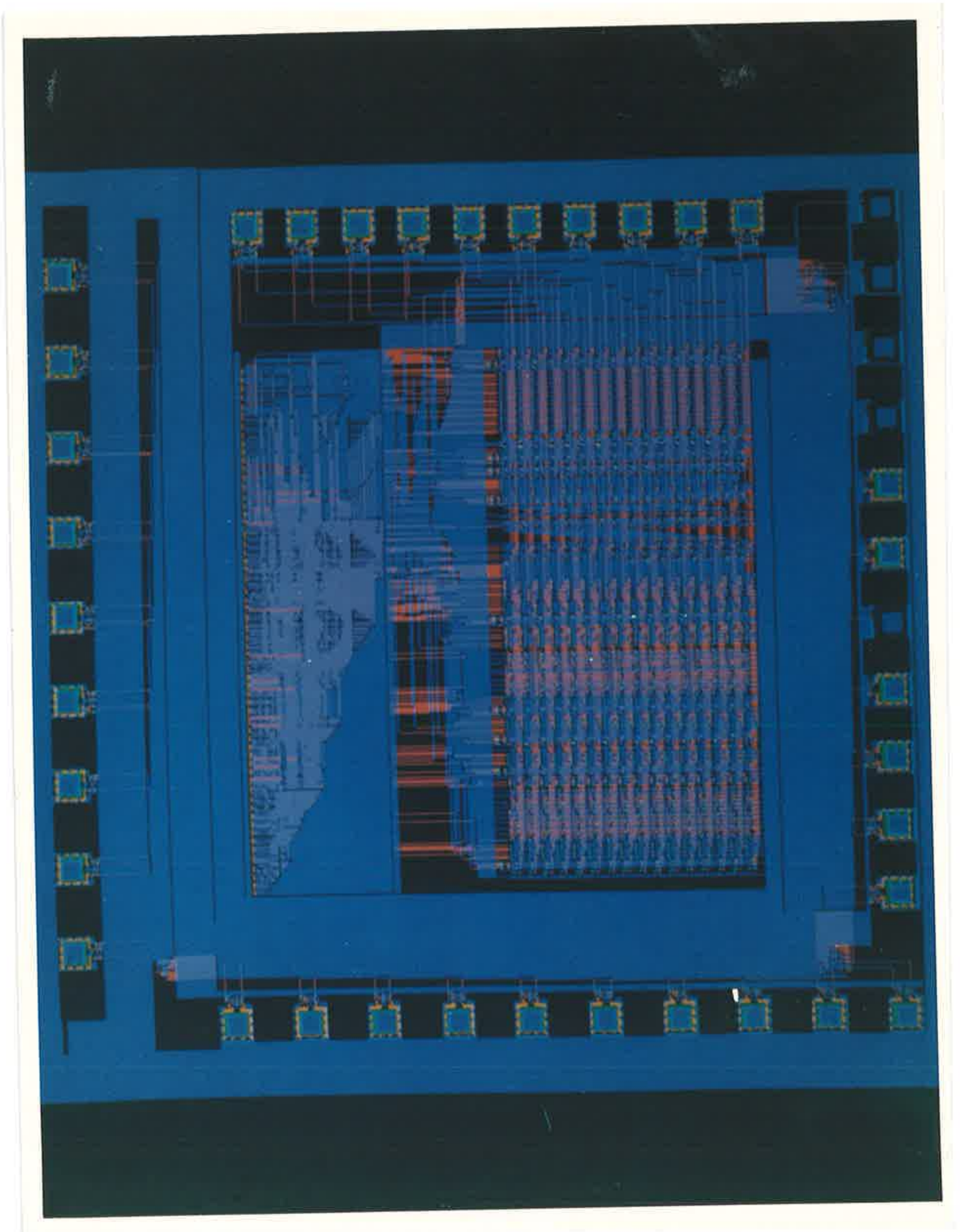


Figure 1.4: A circuit generated by the Macpitts Silicon Compiler. The circuits generated by current Silicon Compilers are still quite restricted and these tools cannot yet be considered completely versatile as far as Custom IC Design is considered. The code used to describe the circuit above is given in the following three pages.

```
(program frisc 16
 (def * power)
 (def * ground)
 (def * phia)
 (def * phib)
 (def * phic)
 (def reset signal input *)
 (def read signal output *)
 (def write signal output *)
 (def interrupt-request signal input *)
 (def interrupt-acknowledge signal output *)
 (def address port tri-state *)
 (def data port i/o *)
 (def p register)
 (def s register)
 (def m register)
 (def a register)
 (def b register)
 (def i register)

 (process cpu 2
reset-cpu
  (setq address 0)
  (par (setq address 0) (signal read))
  (par (setq address 0) (signal read) (setq p data))
  (setq address 1)
  (par (setq address 1) (signal read))
  (par (setq address 1) (signal read) (setq s data))

 instruction-fetch
  (cond ((not (eq? i 1 (3 2 1 0)))
         (setq i (> > i 4))
         (cond (( = 0 i) (go instruction-fetch))
               ((eq? i 2 (3 2 1 0)) (call constant))
               ((eq? i 3 (3 2 1 0)) (call get-s))
               ((eq? i 4 (3 2 1 0)) (call set-s))
               ((eq? i 5 (3 2 1 0)) (call get-m))
               ((eq? i 6 (3 2 1 0)) (call load))
               ((eq? i 7 (3 2 1 0)) (call store))
               ((eq? i 8 (3 2 1 0)) (call goto))
               ((eq? i 9 (3 2 1 0)) (call if))
               ((eq? i 10 (3 2 1 0)) (call end))
               ((eq? i 11 (3 2 1 0)) (call mark))
               ((eq? i 12 (3 2 1 0)) (call call))
               ((eq? i 13 (3 2 1 0)) (call return))
               ((eq? i 14 (3 2 1 0)) (call add))
               ((eq? i 15 (3 2 1 0)) (call increment))
        (t (setq i (> > i 8))
               ((eq? i 0 (7 6 5 4)) (call nand))
               ((eq? i 1 (7 6 5 4)) (call subtract))
               ((eq? i 2 (7 6 5 4)) (call shift)))))
  (go instruction-decode)
```

```
interrupt
  (call push)
  (call push)
  (par (setq b m) (setq a p) (setq m (+ s 1)) (setq address 2))
  (par (setq address 2) (signal interrupt-acknowledge))
  (par (setq address 2)
       (signal interrupt-acknowledge)
       (setq p data)
       (go instruction-fetch))

constant
  (call push)
  (setq address p)
  (par (setq address p) (signal read))
  (par (setq address p) (signal read) (setq a data) (setq p (+ p 1)) (return))

get-s
  (call push)
  (par (setq a s) (return))

set-s
  (par (setq s a) (call pop))
  (go pop)

get-m
  (call push)
  (par (setq a m) (return))

load
  (setq address a)
  (par (setq address a)(signal read))
  (par (setq address a)(signal read) (setq a data) (return))

store
  (par (setq address b) (setq data a))
  (par (setq address b) (setq data a) (signal write))
  (par (setq address b) (setq data a) (go pop))

goto
  (par (setq p a) (go pop))

if
  (par (cond ((<0 b) (setq p a))) (call pop))
  (go pop)

end
  (go pop)

mark
  (call push)
  (par (setq m (+ s 2)) (setq a m) (return))

call
```

```
  (par (setq p a) (setq a p) (return))

return
  (par (setq p b) (setq b a) (setq s m) (call pop))
  (par (setq m b) (setq b a) (go pop))

add
  (par (setq b (+ b a)) (go pop))

increment
  (par (setq a (+ a 1)) (return))

nand
  (par (setq b (word-nand b a)) (go pop))

subtract
  (par (setq b (- b a)) (go pop))

shift
  (par (setq a (> > a)) (return))

push
  (par (setq address (setq s (+ s 1))) (setq data b))
  (par (setq address s) (setq data b) (signal write))
  (par (setq address s) (setq data b) (setq b a) (return))

pop
  (par (setq address s) (setq a b))
  (par (setq address s) (signal read))
  (par
   (setq address s) (signal read) (setq s (- s 1)) (setq b data) (return))))
```

In conclusion, the VLSI design philosophy should take into account all the problems faced by the large software system designer, and many more. Additional problems faced by the VLSI designer are mainly based on the two-dimensional nature of IC's and the need for physical interconnections between modules. The design methodologies introduced in this thesis will be based on two different levels of the design philosophy. The first will be the lowest level, mask level layout description, analogous to machine code in the software domain. The second methodology introduced will be a higher level, symbolic level description, analogous to assembler code in the software domain.

1.3 Australian MPC's

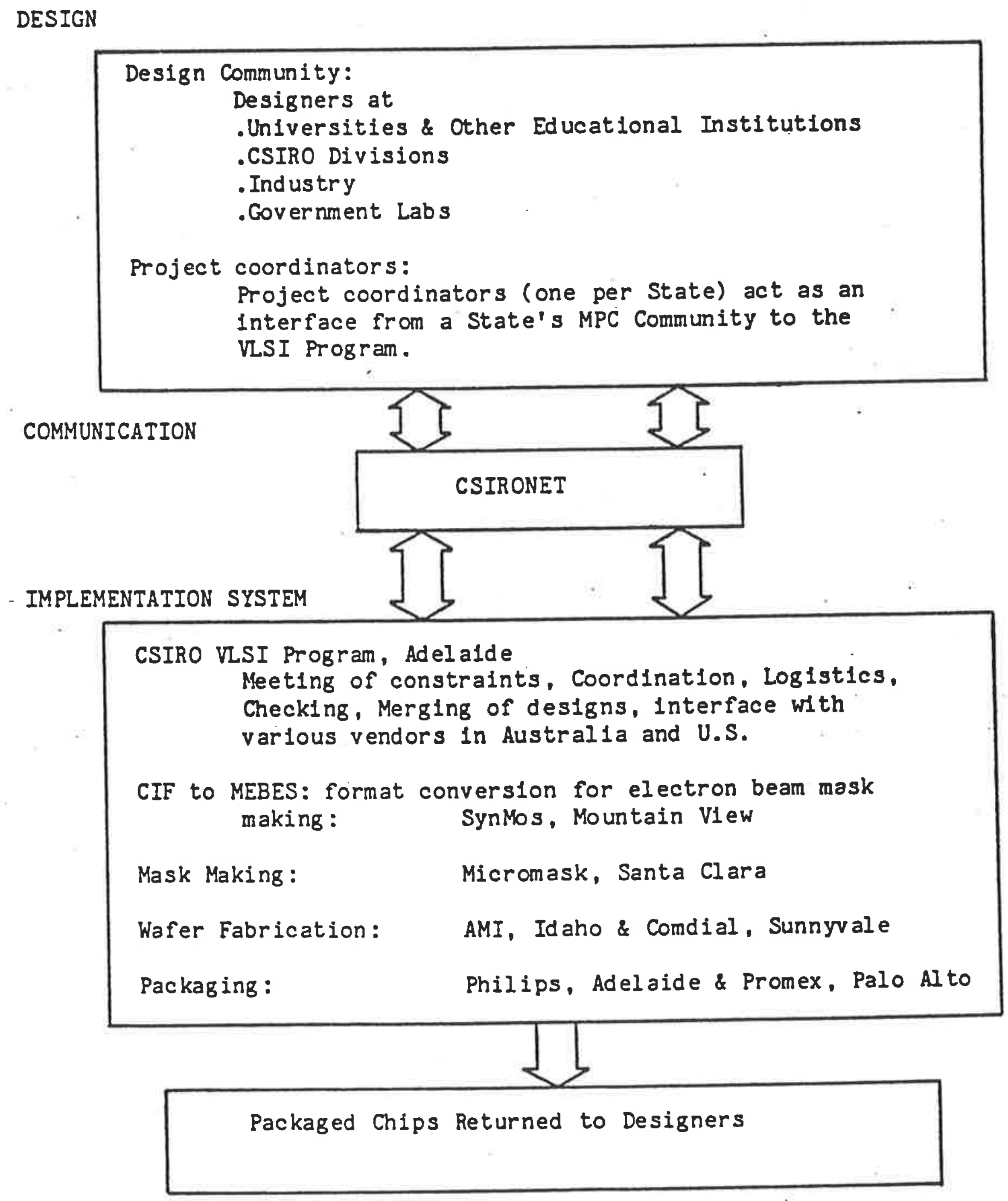
A *Multi-Project Chip* (**MPC**) is a chip containing more than one independent circuit design, with each design being considered an independent project. Usually there will be a number of independent MPC's all on the one wafer resulting in the term *Multi-Project Wafer* (**MPW**), see Fig's 1.2 (a) and (b) and Fig's 1.3 (a) and (b).

According to Bell[15], the structured design methodology introduced by Mead and Conway[8] specifically to handle the design of VLSI systems, required as an integral part of its plan a new implementation methodology for fabrication of IC's. The previously existing method of fabrication of one design per fabrication run could not economically handle the large number of designers trained by the Mead and Conway method. This new methodology still had to rely on existing Foundry services that had the equipment and knowledge to build wafers, but not to handle all the designs. The designer needs only to know that the implementation process is something to which designs must be sent and from which implemented and packaged chips will be received. This idea is consistent with the intention that the complex design problem be partitioned into manageable blocks.

The work done by Mead and Conway detailing the structured design methodology used in the generation of MPC's based on mask level layouts of nMOS designs, in conjunction with the book produced by Hon and Sequin[7], presented the excellent documentation required as an integral part of the methodology. By actually documenting the techniques involved in approaching a complex task such as VLSI design as well as documenting a set of tools based on this methodology, these researchers have now made it feasible for research organisations such as universities to participate in the design and construction of computing systems totally in silicon. According to Mudge[11], the last time universities could economically compete with systems designers was in the second (transistor) generation.

Simply put, the Mead and Conway methodology for design of VLSI systems requires a structured approach to the mask level design of nMOS circuits using a relaxed set of design rules where mask generation, fabrication and packaging are handled by a **Silicon Broker** or **Foundry**[19] which will ultimately return fabricated designs to the designer with a fast turnaround time, at a relatively low cost. The methodology has been used in the US and in Australia for separate MPC fabrication runs and has been shown to work successfully[15]–[18]. Most significantly, the designers creating such systems come from diverse scientific disciplines and have already produced significant designs. A good example of some useful designs proven on MPC runs are Richard Lyon's Optical Mouse Chip[20] (US MPC run), and a High Speed Digital Correlator designed by J. G. Ables and A. J. Hunt[21] (CSIRO Aust. MPC run).

The first Australian MPC effort was initiated by the *Commonwealth Scientific and Industrial Research Organisation* (**CSIRO**) VLSI group, an organisation funded federally by the Australian Government, and commenced in earnest with a course given by Dr. K. Eshraghian and Dr. D. Pucknell of Adelaide University[22][17] in Adelaide in February 1982. The results of the first and some subsequent MPC efforts can be seen in Mudge and Clarke's reports[17][18]. Over the period of the existance of the CSIRO's VLSI Group, mid 1981 to late 1984, 8 successful MPC runs, using both $5\mu m$ and $4\mu m$ minimum feature size, were completed. Fig. 1.5 (a) shows a flowchart of the implementation system used by the CSIRO for their MPC effort. Prompted by CSIRO's efforts, the *Joint Microelectronics Research Center* (**JMRC**) and *Amalgamated Wireless*



DESIGN

Design Community:
Designers at
.Universities & Other Educational Institutions
.CSIRO Divisions
.Industry
.Government Labs

Project coordinators:
Project coordinators (one per State) act as an interface from a State's MPC Community to the VLSI Program.

COMMUNICATION

CSIRONET

IMPLEMENTATION SYSTEM

CSIRO VLSI Program, Adelaide
Meeting of constraints, Coordination, Logistics, Checking, Merging of designs, interface with various vendors in Australia and U.S.

CIF to MEBES: format conversion for electron beam mask making:	SynMos, Mountain View
Mask Making:	Micromask, Santa Clara
Wafer Fabrication:	AMI, Idaho & Comdial, Sunnyvale
Packaging:	Philips, Adelaide & Promex, Palo Alto

Packaged Chips Returned to Designers

Figure 1.5 : The Flowchart of the CSIRO AUSMPC prototyping process. This process was successfully completed 8 times before finishing in 1984. The above flowchart is copied from the CSIRO AUSMPC 5/82 Designer's Manual.

Australasia (**AWA**) in Sydney set up their own MPC facility and have also been successful in multiple MPC runs. Fig. 1.5 is very similar to the flowchart for the implementation system used by JMRC. The main differences arise in the Communication block where ACSNET[73] is used in preference to CSIRONET, and in the Implementation System block where AWA was responsible for Mask Making, Wafer Fabrication and Packaging. Between 1982 and mid 1985, JMRC had successfully completed 10 MPC runs using an nMOS $5\mu m$ minimum feature size process and in mid 1985 commenced efforts towards running a CMOS MPC. The University of Adelaide Dept. of Electrical and Electronic Engineering is also in the process of commencing CMOS MPC efforts. Both JMRC and Adelaide University CMOS MPC efforts would use AWA fabrication facilities initially.

Since February 1982 when the Mead and Conway VLSI design methodology was first introduced to Australian circuit designers, there have been numerous nMOS MPC efforts using $5\mu m$ and $4\mu m$ processes and the beginnings of CMOS MPC efforts resulting in hundreds of circuits being prototyped in silicon by remote designers with little or no experience in IC design, but with some experience in Computer Science or Electrical Engineering or preferably both. The number of organisations offering MPC runs is beginning to grow as is the number of organisations offering CAD tools for the design of these circuits. IC prototyping of circuits in Australia using MPC fabrication techniques is now a proven reality.

1.4 Summary

The intent of this thesis is to illustrate in detail that Australian Industry, Research and Design Centers and University Departments have the capability to allow designers to prototype their systems in silicon at an affordable price with a fast turn around as long as they have the appropriate Computer Aided Design facilities, both software and hardware. This aim will be realised by analysing the effectiveness of a range of different CAD tools as applied to the fabrication of IC's using the MPC process. This thesis provides examples of the MPC process as applied to the nMOS technology, and applies the same techniques to indicate the viability of the MPC process as applied to the CMOS technology.

It will be shown that as new technologies become available, using well designed tools, minor modification only is necessary to prepare VLSI CAD tools for use. The protoypes detailed in this thesis have been fabricated by different facilities both in Australia and the US, indicating the versatility of this method of prototyping electronic circuits.

Different design methodologies will be used to show different approaches to the complex VLSI design problem, and a comparison of their performances will be made. Chapter II details the variety of tools used in mask level IC design and compares the operation of many currently used tools. Chapter III introduces higher level tools, based on a symbolic design methodology and compares them to the mask level tools. Chapter IV will provide a detailed account of two nMOS design experiences as a verification of the mask level tools and existing MPC implementation systems, while Chapter V will provide a similar account of a CMOS design experience with the higher level tools, and draw comparisons between the High Level and Low Level Tools.

Full custom as opposed to semi-custom tools will be considered. Semi-custom design methodologies, while prompting research into many useful CAD tools (eg. for automatic placement and routing), and providing economical solutions to certain design problems not yet handled by reasearch CAD tools and design methodologies, will eventually lose favour due to their inherent disadvantages with regard to functional device density and performance which has been shown to be an order of magnitude worse than full custom designs[14.1].

As full custom CAD tools become more refined, as CAD software and hardware becomes less expensive, as standard IC fabrication processes become more commonly available in the US and Australia, and as communication networks become more common and accessible, CAD tools for VLSI circuit design similar to those described in this thesis will be used by circuit designers as common practice in prototyping their designs. The methods of prototyping circuits as detailed in this thesis are not new but need to be described as an essential part of the design methodologies. Documentation is also provided on the various Australian MPC facilities and some design experiences. Most of the tools that are examined have been used by many designers for many MPC runs. Some tools are totally new and will soon be tested by fabricating and testing designs on more MPC runs. The tools in particular provide an insight to component parts of the ultimate tool of the future, the Silicon Compiler.

CHAPTER II

LOW LEVEL CAD TOOLS FOR VLSI CIRCUIT DESIGN AND ANALYSIS

Because VLSI design is complex, designers think of their designs not in their full complexity, but rather as collections of parts which may be further decomposed into other parts. This hierarchical viewpoint both expresses the designer's understanding of his design and economises his thinking about it. All unnecessary information is suppressed, enabling the gate, module, subsystem or system of interest to be dealt with.

It is common to talk of these hierarchical divisions by discussing *Leaf Cells* as opposed to *Composition Cells*. A leaf cell contains mask level geometry and usually describes a fundamental block in a large system which may need significant replication or which may be placed in a cell library for use in several designs. Composition cells are groups of leaf cells or other composition cells, or both, which are combined to form larger system blocks. The information available within the composition cells commonly provides cell I/O connection positions and the layer on which the connection should be made, possibly some electrical characteristics of the cell and in some cases, peripheral mask level information[28].

The Broker/Foundry accepts the intermediate form description of the design layout, merges the design with others on an MPC, and then converts the intermediate form of the merged MPC into a form that will drive the chosen patterning mechanism. Currently, design files are converted to *Pattern Generator* (**PG**) files for use by a maskmaking firm for driving an optical or Electron Beam pattern generator, which is the first step in maskmaking. By a series of photolithographic steps, a mask house will produce a set of masks which a commercial fabrication house will then use to pattern silicon wafers. The remaining function of the Broker/Foundry is to have the wafers diced and packaged and yield tested before returning the chips to the designer.

A set of CAD tools should be able to represent the design in much the same way as does the designer. Thus the basis of a set of design tools should be a data-base representation of designs that is flexible, extensible and hierarchical. This chapter examines some tools and their different approaches to these design principles. For complete analysis of the tools, the ability to handle non-manhattan geometry will be used as one of the criteria for measuring each tool's versatility.

2.1 Mask level Design of IC's using MPC's

At the lowest level of programming, machine code, or bit information is loaded into a machine. At the lowest level of IC design, mask layouts are described by the designer. The mask level layout description of an IC design will conform to a set of relaxed design rules as described by Mead and Conway[8] (see Appendix A). To adequately describe mask level layouts of IC designs in such a way as to interface effectively with an MPC run, a number of stages must be methodically passed through (see Fig. 1.5).

At the mask level of IC design, the designer must first transform the circuit and topological level designs into a geometrical layout of the system. The design must be verified, so design checks discovering layout errors will be required, these checks will usually be carried out in an iterative manner until the layout is design rule error free. The result will be a design file that accurately describes the layout in a particular representative intermediate form which can be read by a Silicon Broker/Foundry. The intermediate form used by tools discussed in this thesis is the *Caltech Intermediate Form* (CIF[7]). Another intermediate form at a different design level will be introduced in Chapter III, however, the basic format that is able to be read by existing foundries is at the lowest design level, that of mask layout.

It is important to note that the design rules used with CIF are based on a *length* *unit*, λ. This unit can be scaled up or down as required to match the minimum value of a fabrication facility's process. That is, the λ concept should produce process independent designs. The transit time for transistors, resistance per square and capacitance per unit area will all vary from one fabrication line to another, yet the λ based designs should function correctly if the designs are logically and electrically correct. The following work describes and makes some comparisons of a complete set of CAD tools that have been successfully and extensively used for the design and verification of IC's, at University of Adelaide.

2.2 Embedded Layout Languages

Mask level layout tools use high level languages run on general purpose computing systems to provide Mask Layout languages. These Layout Languages generally exploit the features of languages such as PASCAL or LISP to enable faster more efficient digitization of designs into machine readable form (CIF). Mead and Conway[8] provide an example of such a language. Other examples are **DPL**[23], a LISP based Layout Language, **BELLE**[24] and **CELLE**[25], PASCAL Embedded Layout Languages (see Appendix B). Fig. 2.1 shows an example of a Layout Language and its resultant layout. Such Layout Languages inherit the power of the full programming language in which they are embedded. High level programs can be written that call Layout Language functions and vice-versa. These languages take a symbolic description of mask level layout elements such as boxes, or wires at different mask levels, and process these low level descriptions to produce an intermediate form (in this case CIF) description of the mask layout.

The only CAD peripheral hardware requirements absolutely necessary for design using such languages are a text editing terminal, a plotter, and some means of transporting information from the computing system running the tools to the broker/foundry organising fabrication of the design. This could be done using for example, an electronic mail network, (eg. Australian networks, **ACSNET**†[73], or **CSIRONET**[72]), magnetic

† ACSNET is a trademark of Sydney University

tape, or even removable disks. The software requirements are a compiler for the high level language used to embed the Layout Language, a text editor (the more versatile the better), and a plotting routine that can take the CIF output from the Layout Language and reproduce a hard copy for investigation.

Mask Layout Languages are programs used to digitize mask level descriptions of IC designs and therein lies a problem. A circuit designer normally considers systems in terms of, at worst, the fundamental electronic circuit components, transistors, capacitors, resistors and wires. The IC designer using a Layout Language however must manage the complexity of his circuit design as well as the non-trivial task of converting that circuit design to a detailed mask layout on paper, prior to digitization. This must necessarily be done on an absolute grid, preferably $\lambda = 1$ grid spacing. Such digitization techniques are quite tedious and slow relative to higher level tools for IC design, although Mask level Layout Languages provide a more efficient digitization medium than digitizing using CIF, which is possible but not advisable.

The advantages of Mask Layout Languages derive from the power of the parent languages in which they are embedded. Once a significant number of designs have been digitized and verified, a database consisting of a library of these cells can be used in conjuction with the Layout Language for composition of systems.

It is important at this stage to indicate that most of the tools discussed in this thesis create or accept only Manhattan geometry at the mask level. This means that all mask geometry created or processed is parallel to the x or y axis, there are no diagonal lines used. This design restriction significantly reduces the complexity of the IC designs as well as the design of the tools used for their creation and analysis.

2.3 Interactive Mask Level Geometry Editors

Geometry editors provide the ability to interactively manipulate pictorial representations of objects on a monochrome or colour video terminal. Sections of pictures can be inserted, deleted, moved, replaced, or operated on in much the same way as text

```
PROCEDURE dynshift;


CONST
      Diff_width        =  2;
      Diff_spacing      =  3;
      Poly_width        =  2;
      Metal_width       =  3;
      Pol_Dif_spacing   =  1;
      Pol_Dif_overlap   =  2;
      Impl_over_channel=  2;
      Impl_to_channel   =  2;
      Cut_size          =  2;
      Cut_spacing       =  2;
      Cut_to_edge       =  1;
      Pwr_width         =  4;

VAR
      cell_start: coordtype;
      cell_width,cell_length:  integer;
begin
   setsymno(9040);
   define('dyn_shift');
   cell_width:=33;
   cell_length:=48;
      Layer(Metal);
          box(0,0,cell_width,pwr_width);
          box(0,22,cell_width,22+pwr_width);
          box(0,44,cell_width,44+pwr_width);
    Layer(diffusion);
          box(14,14,20,20);
          box(14,6,17,15);
          box(16,6,30,8);
          box(14,28,20,34);
          box(14,33,17,42);
          box(16,40,30,42);
          wire(2,19,0);  y(6);
          wire(2,0,35);  x(6);  y(10);  x(14);
          box(15,22,19,26);

      Layer(poly);
           box(22,4,28,11);
           box(17,9,20,13);
           box(19,9,23,11);
           box(17,35,20,39);
           box(22,37,28,44);
           box(19,37,23,39);
           wire(2,13,17);  x(26);  y(37);
           wire(2,12,12);  y(-11);
           wire(2,3,33);  y(59);
           wire(2,11,31);  x(21);

      Layer(Implant);
           box(20,4,30,10);
           box(20,38,30,44);

      md(-1,34);
      mp(28,34);
      md(31,42);
      md(17,27);
      md(17,21);
      md(31,6);
      dp(17,11,0);
      dp(8,32,0);
      dp(17,37,0);

   enddef;
   end;
   begin
      dynshift;
      setnoend;
   end;
```

Figure 2.1 (a): A copy of the BELLE Layout Language used to describe the shift register cell designed for use in project C4, AUSMPC May 1982.

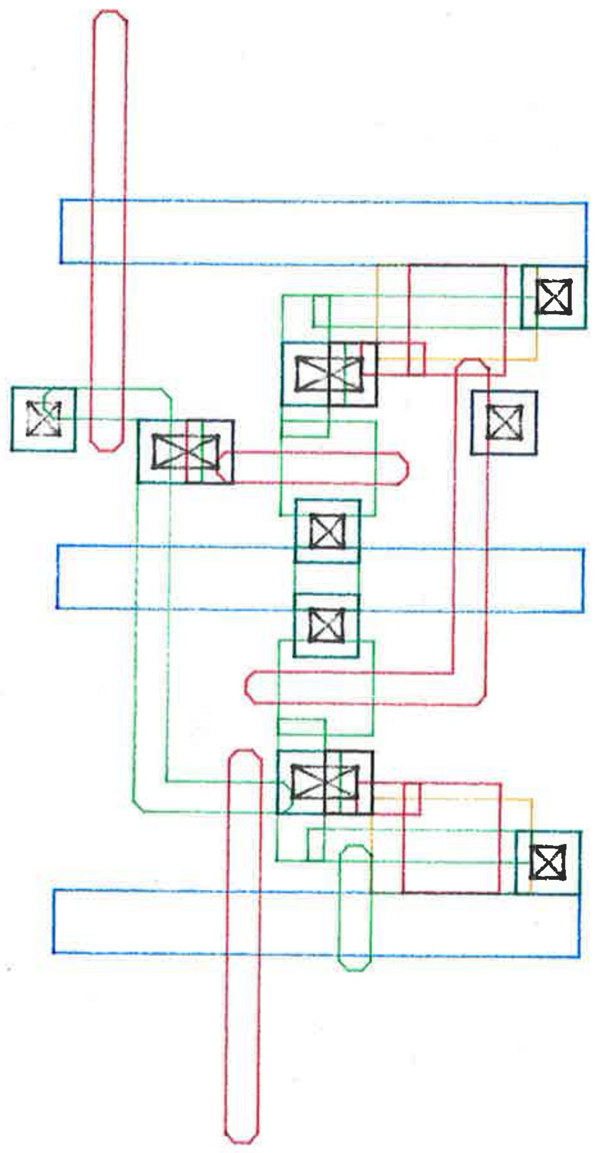


Figure 2.1 (b): A computer generated plot of the shift register cell described in Fig. 2.1 (a).

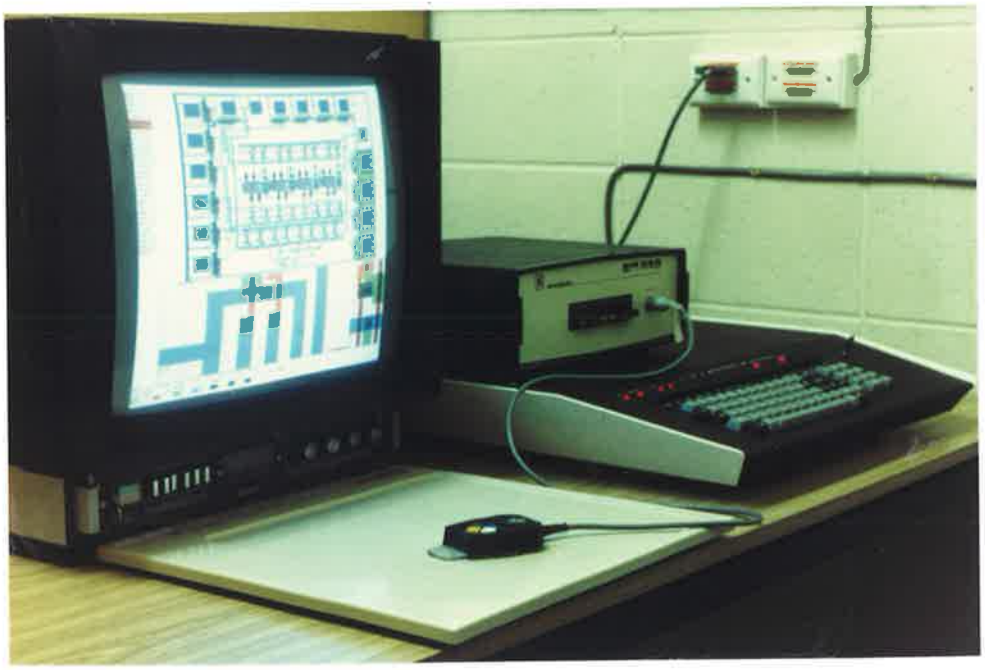
can be altered with a sophisticated text editor. Tools such as **ICARUS**[8], **KIC**[27], **CAESAR**[29], **PLAN**[26], **ET**[28] and **MAGIC**[29.1] are all examples of interactive, mask level, geometry editors.

The advantage in using a geometry editor as opposed to a Mask Layout Language is that the editor can contain all of the Layout Language functions and more. As well, the editor is able to provide visual evidence of the design developments and modifications interactively. The advantage this brings is that no detailed layout need be done on paper beforehand. Some ***STICK*** diagrams may be required for design guidelines, but even these may be unnecessary for an experienced user of a geometry editor.

The hardware required to use these tools is the same as that for the Layout Languages, plus either a monochrome or colour interactive graphics terminal, or both, with some method of manipulating cross-hairs on the terminal. This may be done either via the terminal keyboard, or a bitpad, or an optical or mechanical mouse[20][30]–[34]. Some typical ***Workstations*** are shown in Fig. 2.2 (a), (b) and (c) running various geometry editors. (Workstation is the common noun used to describe the hardware used by a designer to run a complete set of VLSI design tools).

The software required for this type of tool is again similar to the Layout Language software in its philosophy. The differences between the two tools are that as well as performing the layout task that the Layout Language succeeds in achieving, the Geometry Editor also allows interactive editing of layout. The editing task is much more time consuming using a Layout Language and can only be completed using an additional software tool, a Text Editor.

Commonly the structure of a geometry editor provides for a subdivision of the program into a number of procedures corresponding to the number of functions provided by the editor menus. A versatile editor will drive more than one type of graphics device and so usually contains a number of procedures corresponding to the number of device drivers used by the editor. Generally, a Geometry Editor is a much more complex



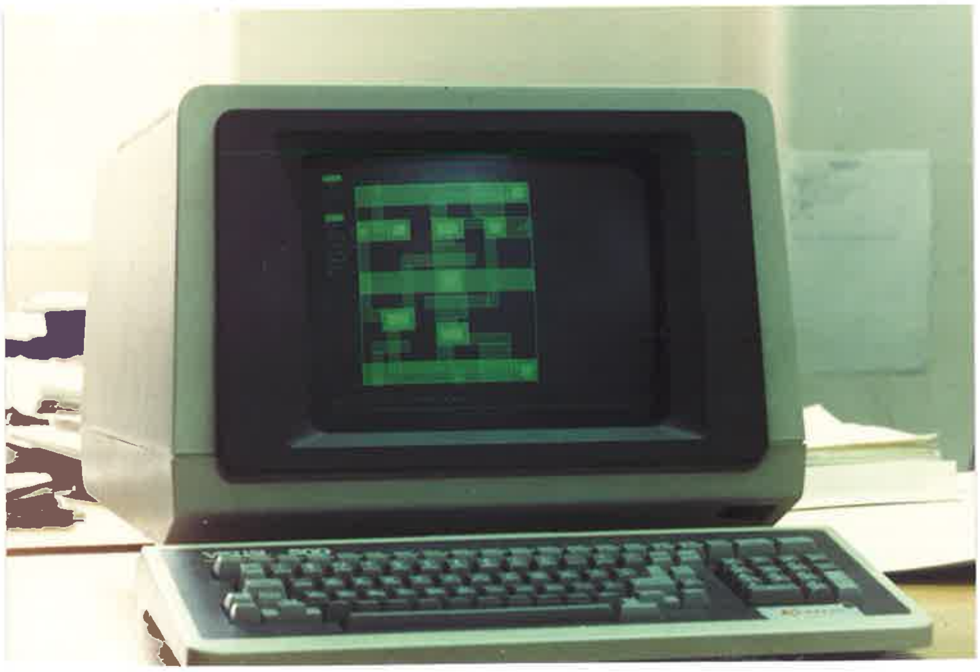


Figure 2.2: (a) Above, an early model of a colour graphics workstation. An AED512 with a colour monitor and a bit pad showing some KIC output of project C4. (b) Below, a VISUAL 500 monochrome workstation[30], showing some PLAN output of project C4. Note that the cross hairs in this case are controlled from the terminal keyboard.

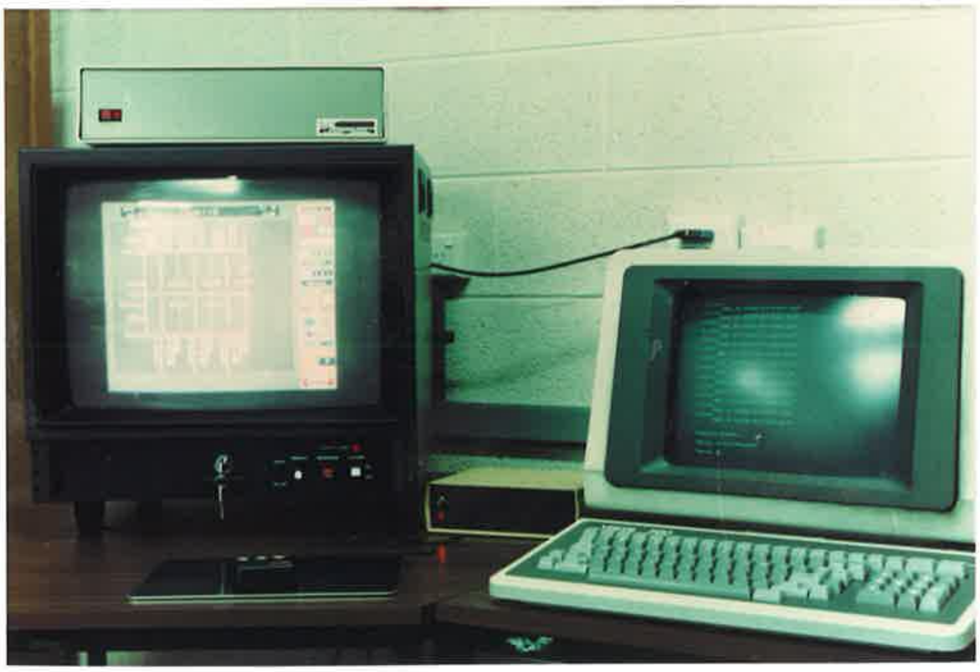


Figure 2.2 (c): Another colour graphics workstation, a VECTRIX[37] with optical mouse for cross hair control. Note also the monochrome terminal alongside used for displaying additional menus and allowing the use of a text editor, or alternatively some other CAD Tool while viewing the circuit on the colour monitor. Also compare the bulky bitpad in (a) with the much more compact optical mouse shown here.

program than a Layout Language as it must perform the same geometry instantiations as a Layout Language, it must display them interactively on a graphics peripheral, and it must also incorporate an editing capability which a Layout Language does not have.

Just as Layout Languages produce CIF output, so do Geometry Editors, but usually not directly. The machine readable output format varies from editor to editor. Generally, it is similar to CIF, but not the same. Additional software must accompany the editor to translate from the editor output format to the desired intermediate form, in this case CIF. It is generally possible to translate from the intermediate form to the editor output format as well. For some tools, this second function tends not to be as useful, because the hierarchical features which are embedded in some of the editors are not translatable to cif. For example, PLAN does not have a routine to transfer from

CIF to the PLAN output format because of the hierarchical restrictions imposed by PLAN on a designer.

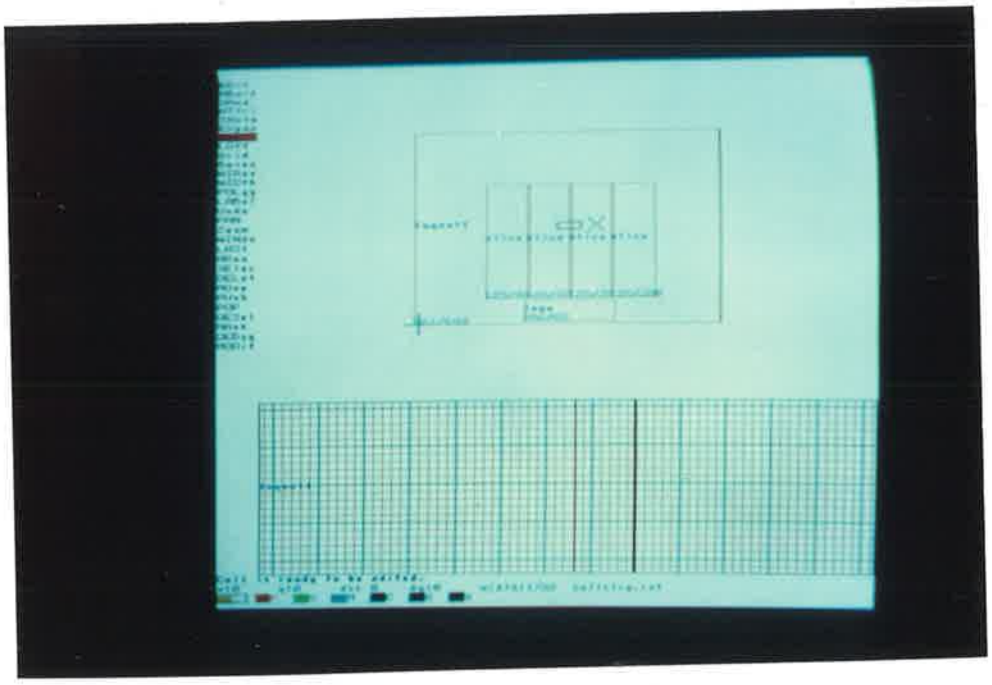
2.3.1 Hierarchical Design Philosophies of Geometry Editors

Hierarchical information is necessary to simplify the complex IC design task. The form this hierarchical information may take varies from editor to editor, but a simple comparison of KIC, PLAN, and ET, will provide an overview of different approaches to hierarchical IC design using geometry editors.

This comparison is based around innovations in hierarchical design strategies for geometry editors rather than a comparison of the effectiveness of one tool for VLSI design, with respect to another. The three editors named in this section each have very similar geometry manipulation functions and are at different stages of development. KIC was developed at the University of California, Berkeley[27], originally started as a Master's thesis by Keller in 1981. PLAN[26] has been developed by a firm called *Integrated Silicon Design* (**ISD**), and originated from a project, RIDE[25.1] carried out by Dickinson. ET has also been developed at the University of Adelaide as the result of research work by Woloszczuk[28][28.1].

KIC provides no hierarchical design restrictions. The electrical relationship between the cells is not machine recognisable. The onus is on the designer to recognise and account for all connections between cells at the time of composition. The only hierarchical division of cells is a visual bounding box around the perimeter of a cell at the time of composition. Even this hierarchical division may be freely turned on or off by the designer using a menu selection called **EXPAND** which then displays all of the enclosed leaf and/or composition cell information within this top level cell (see Fig. 2.3)

PLAN, however, provides connectivity information as well as the bounding box around the cell. When a cell is generated, pins must be placed on the inputs and outputs of the cell before connection can be made between cells when they are composed. These pins only appear within a bounding box, generally on the edge of the box, and serve



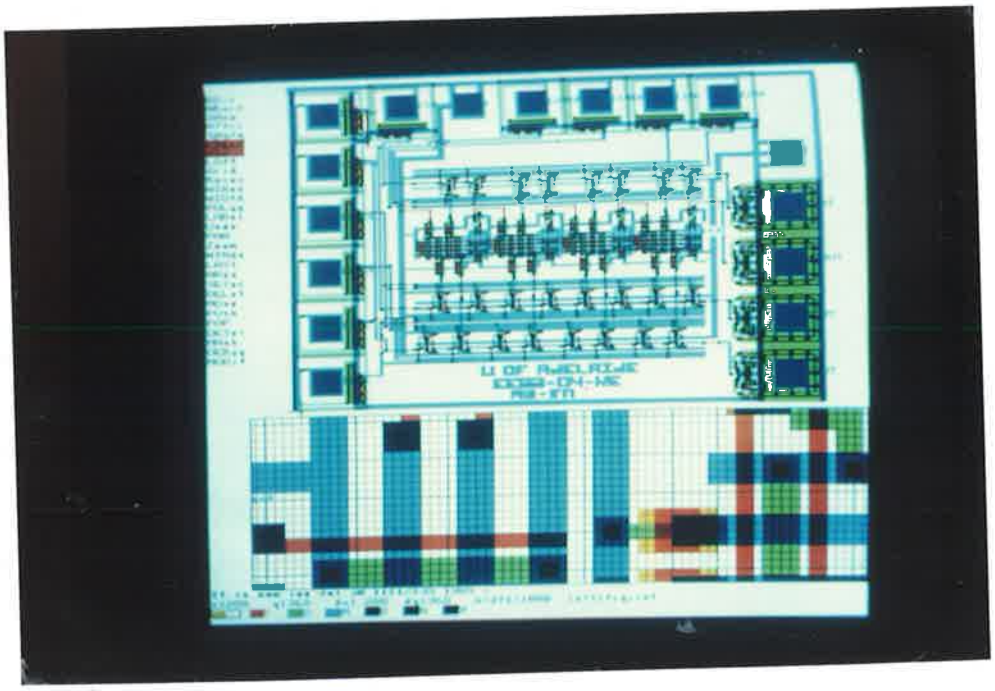


Figure 2.3: The top picture shows a non EXPANded view of project C4 using KIC, while the picture below shows an EXPANded view of the same design. Some interesting KIC features can be seen in these pictures. Firstly the Magnifying Glass, the grided rectangle at the bottom of the pictures providing a magnified view of the contents of the small black rectangle shown in the top picture on the left of the cross hairs. Also the MENU of KIC commands selectable via the bit pad can be seen on the left hand side of the pictures. At the bottom of the pictures can be seen the LAYER TABLE viewport indicating the various mask levels used and their associated colours, while immediately above this viewport is the PROMPT viewport enabling textual input by the designer. The hierarchy in KIC is only implied via the use of non EXPANded bounding boxes as in the top picture.

to define the cell electrical input and output ports and act as terminal nodes for wires (see Fig. 2.4). It is worthwhile noting that in PLAN only hierarchical information is displayed at different levels of composition. That is, the designer can only view composition cells as bounding boxes with I/O ports, there is no equivalent to the KIC **EXPAND** menu command. This restricts the designer to a strictly hierarchical design strategy but can cause problems if connection must be made to pins internal to the cell, (a poor design technique), or if layout geometry is placed close to the edges of the bounding boxes of composed cells, as design rules could be violated. In these circumstances, it is necessary to know the internal geometry of the cell or at least the geometry near the edges that could possibly be involved in design rule violation. PLAN does not allow a designer to see internal geometry for composed cells at the time of composition.

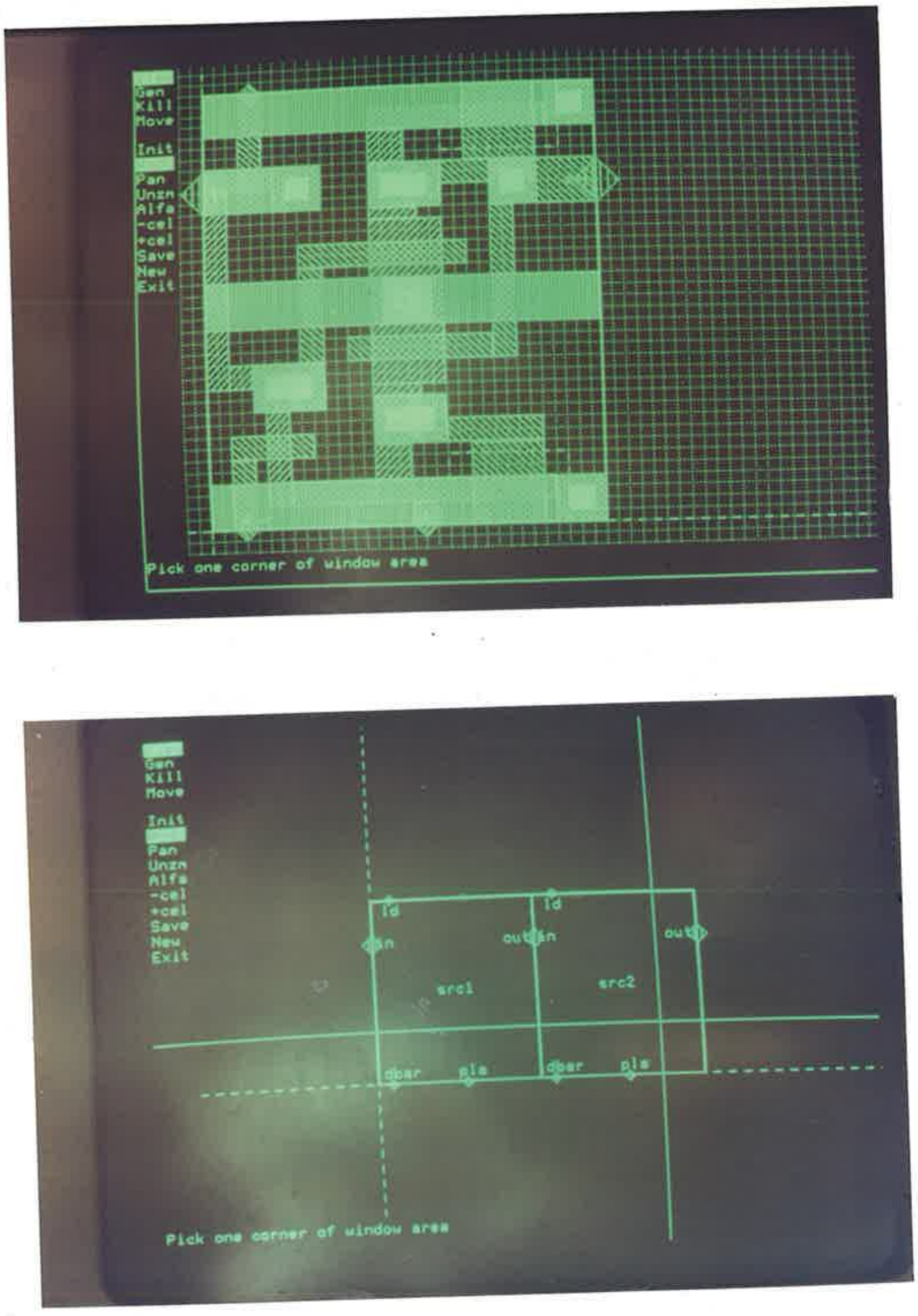
ET considers the hierarchical design problem exposed in PLAN and provides a solution to it. By restricting the designer to the same hierarchical design strategy (although a step backwards has been made by omitting the electrical connectivity information), and adding the feature of displaying mask level geometry to design rule penetration depth at the edges of the bounding boxes of composition cells, the design problems presented by PLAN are avoided (see Fig. 2.5).

2.3.2 Further Analysis of Geometry Editors

The different hierarchical attributes of the various geometry editors are not the only differences between them[26]. KIC is an editor written in the programming language C[35] to run under the **UNIX**† operating system and has been modified[36] to make it portable for use on a machine running the**VMS**‡ operating system. PLAN was written in PASCAL, to run under VMS. ET was written in **VAX-11 C**‡ to run under VMS, with the intention of making it portable to machines running UNIX.

The geometry manipulation functions provided by the different editors are

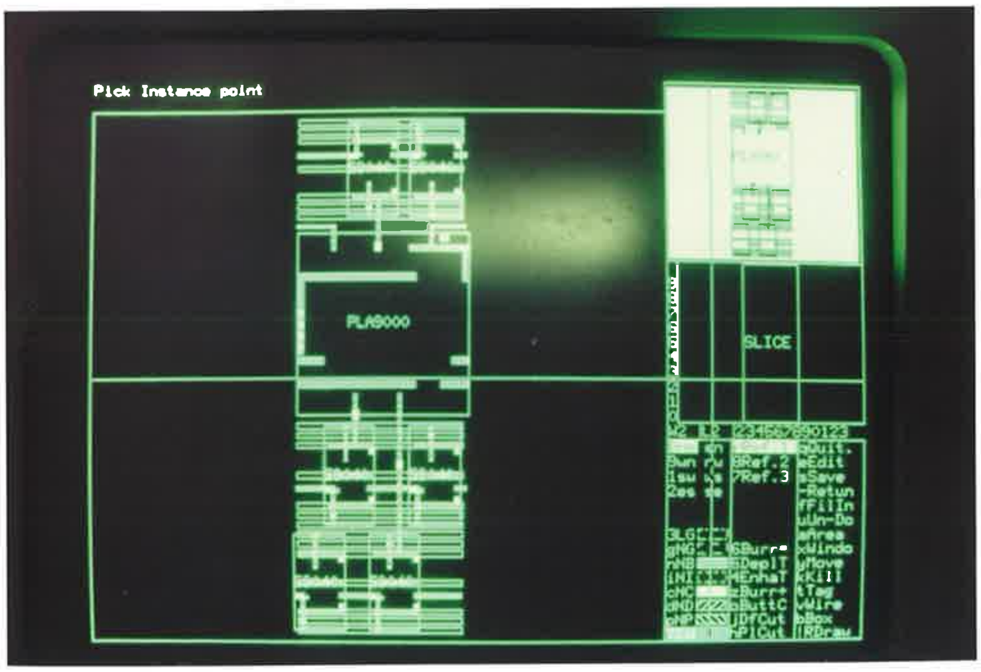
† UNIX is a Trademark of AT&T.
‡ VMS and VAX-11 are trademarks of Digital Equipment Corp.



Pick one corner of window area

Pick one corner of window area

Figure 2.4: A PLAN view of the shift register cell in Fig. 2.1. This cell is shown at leaf cell level above and at composition level below. Note the hierarchical pin information that must be specified on the bounding box to enable connection between leaf cells at the time of composition, shown on the picture below. Note that at a particular level of composition PLAN allows only the hierarchical information to be seen, unlike KIC which allows a full EXPANsion of the composition level down to the leaf cell level.



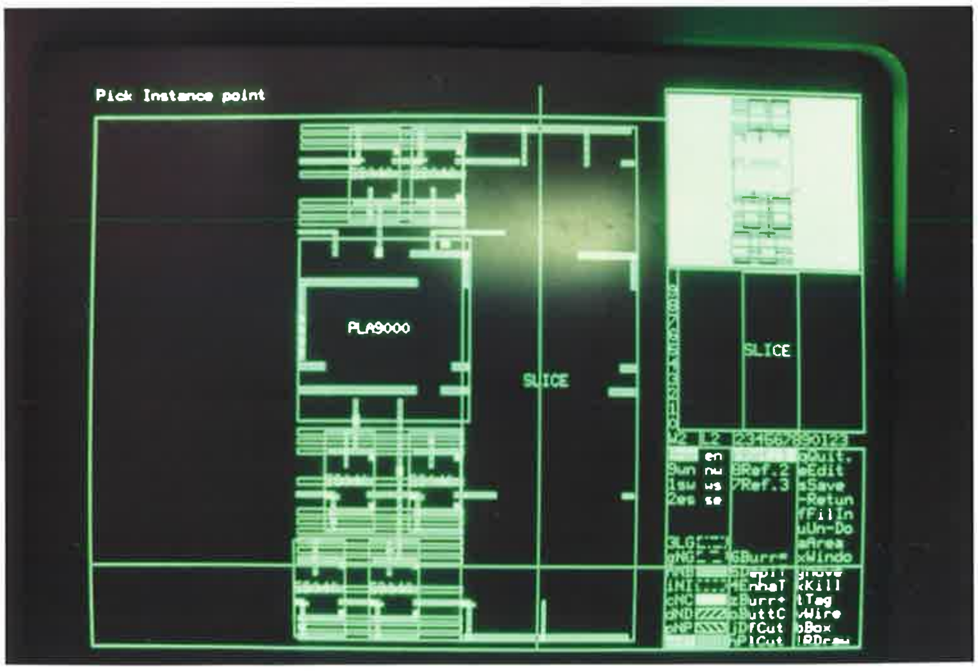


Figure 2.5: Above, an ET view of a bit slice of AUSMPC 5/82, Project C4, displaying routing over sub-cells used in the composition of the cell. Below, two slices are composed alongside each other. One shows leaf cell information to Design Rule Penetration Depth, while the other is the same as that shown above. Notice the three viewports displaying design information, as opposed to two used by KIC. The largest is the working viewport, while the highlighted viewport displays the original cell that was called. The third viewport displays the cell to be used at composition time in its correct composition orientation.

similar, with KIC currently being the most versatile in this respect. This is not surprising as it is one of the oldest of the tools considered, was developed in conjunction with other research tools such as CAESAR and preceded others such as MAGIC[29.1]. For more comments on the Geometry Editing tool MAGIC, see the concluding section, section 2.8, of this chapter.

The main disadvantages of geometry editors relate to the problems of designing in mask level geometry. The problems in converting from electrical design to mask layouts requiring the adherence to geometrical design rules by the designer adds a step in normal design complexity. The hierarchical advances in such editors can only minimize the complexity involved in this conversion process, it does not rid the designer of the conversion problem.

Comparison of Geometry Editors KIC and ET		
Subject	*KIC*	*ET*
Technology Independence	Excellent	Not Good
Terminal Independence	Excellent	Not Good
Hierarchical Information	Fair	Very Good
Hierarchical Restrictions	None	Good
Language	C	C
Operating Systems	VMS or UNIX	VMS or UNIX
Non Manhattan Geometries	Yes	No

Table 2.1: A comparison of factors indicating the versatility of two Geometry Editors. Note that KIC is seen here as the more versatile of the two tools. However, it has been in existance and developed over much longer period. ET will be developed by ISD as PLAN2.

The advantages of these geometry editors are based on the ability of such tools to carry out the conversion from electrical design to mask level layout much more efficiently and quickly, in terms of design time, than Embedded Layout Languages. Geometry Editors demand much more computing resource than Layout Languages however. Table 2.1 above details some of the qualities of two of the editors discussed in this section on a comparitive basis.

2.4 Design Rule Checkers

Using low level layout tools as described previously in this chapter, the chance of an error occuring in the layout process causing a design rule violation is high[37]. If the circuit is to work, careful checking is mandatory[7]. Error checking programs embody the rules for a particular process and examine CIF files or PG tapes for violations, reporting each instance in terms of coordinates or patterns along with the nature of the violation. For MPC processes, the error checking tool must examine the CIF as a remote designer will not be concerned with PG tapes.

Design Rules (see Appendix B) typically assign minimum distances to dimensions of features, spacing between features in the same layer, and spacing between features in different layers. ***Design Rule Checking*** (**DRC**) programs ideally take the CIF description of the layout and provide some output indicating errors when and where they occur. There are a number of different approaches to design rule checking and an analysis and comparison of a number of different design rule checking programmes, Baker's **DRC**[37], Noble's **ADRIC**[39], Hartley's **ROWAN**[42] and ISD's **CHECK**[42], will highlight these different approaches.

Baker's DRC uses a Raster Scan method, in which a small window is scanned across the design in raster fashion, and the contents in the window are checked for design rule violations. A 4λ by 4λ window is used as it was initially presumed that this was a large enough window to provide sufficient information to determine whether design rules had been violated. Baker does acknowledge however that this DRC produces false errors and a larger window, say 10λ by 10λ is needed to detect false errors arising from geometries that are close together, are separated, but are electrically connected. An example of such a situation can be seen in Fig. 2.6. Even the larger window modification will not always delete all such errors.

Apart from false errors, this DRC system has a number of other disadvantages. It is not very fast because it steps across every square of the design, even for large empty spaces. It has no hierarchical structure, the same error is reported every time it is encountered. Finally, the design rules for a particular manufacturing technology are intricately embodied in the program making this tool very inflexible with regard to technology change. It would be a major task to modify this program to accept a new set of design rules. Baker's DRC does accept non-manhattan geometries, but it approximates them in such a way that it makes certain width and spacing checks impossible to perform[39]. Baker's DRC was originally written in C under the UNIX operating system and transported to run under VMS.

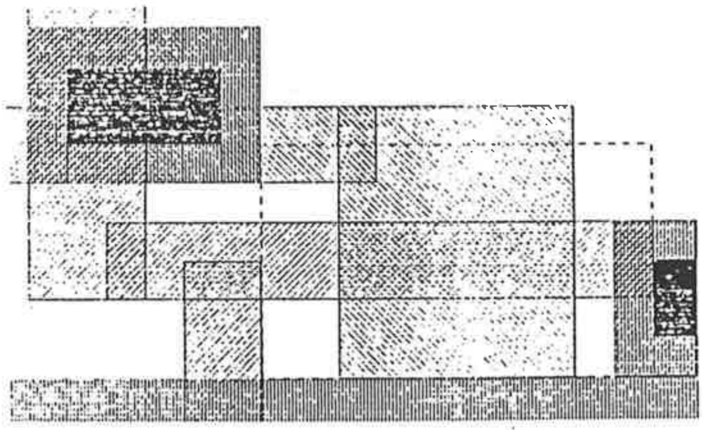


Figure 2.6: The butting contact alongside the depletion mode transistor above, while connected to the transistor, can be seen as erroneous by Baker's DRC if the transistor to contact join falls outside the raster window.

2.4.1 Hierarchical Design Rule Checking

Hierarchical design rule checkers exploit the hierarchy of a design by first checking leaf cells, then checking symbols which instance only leaf cells and CIF primitives, and subsequently repeating this process until the highest level of composition is reached.

ADRIC is an example of such a DRC. Apart from a different method to Baker's DRC, ADRIC also incorporates Hierarchical design rule checking based on the work done in Whitney's HDRC Algorithm[40], although ADRIC, unlike HDRC was designed to accept CIF.

The hierarchical philosophy used by ADRIC applies restraints on the designer at the layout stage. During each run, ADRIC will only check what has not been checked before unless a previously checked and validated symbol has been altered. ADRIC forces the designer to check each level of the hierarchy as it is completed. This is because it can only check those symbol definitions which exclusively instance previously validated symbols. If the designer runs ADRIC regularly, this causes no problems, otherwise the designer may find that several runs of ADRIC are necessary to reach the current level of composition. This restraint could be annoying unless it was to be implemented automatically at the time of composition.

The Polygon method of design rule checking used by ADRIC applies contraction and expansion to the features of the layout. These features are stored as polygons and if two of these polygons overlap when they are expanded by a certain amount, then they are too close together and a design rule is violated. This type of DRC operates by referring to a table indicating which features are to be expanded or contracted and by how much for each design rule for the technology. ADRIC is a much less technology dependent tool than Baker's DRC and together with its hierarchical analysis algorithm, implements some good concepts. However, it can be rather slow and consumes large amounts of storage space. Although ADRIC was never completed, it provided useful background research for ROWAN. ADRIC was written in PASCAL under the VMS operating system.

ROWAN is based on the DRC, LYRA[41], which is written in LISP. The checking philosophy used by this tool is corner based. The rules are only checked at the corners of features on the layout because if a design rule is broken, then there must be a corner at which the violation either begins or ends, or both. This DRC is also hierarchical although the hierarchical philosophy of ROWAN is different to that of ADRIC. ROWAN will take a design at any level of composition and check for errors without having to iteratively run the checker for all unvalidated cells, at each level of the hierarchy. ROWAN takes advantage of the fact that CIF allows structured designs in which certain basic circuits can be defined as symbols, and then be incorporated in larger circuits by just calling the symbol. If the design is done like this in a repetitive fashion, a tree-like structure results.

ROWAN exploits this structure to speed up the design rule checking and reduce the number of error reports made for the same error. ROWAN does this by using a bounding box concept similar to that used in PLAN for individual cells. This cell is checked once, when it is defined, and skipped when it is called later as long as no CIF primitives intrude within the bounding box. If CIF primitives do intrude within the bounding box, the cell will be checked.

If it is possible to skip a symbol when it is called, a thin boundary layer, similar to that appearing in ET, must still be checked for errors involving objects surrounding the symbol call. So if a symbol has been checked upon definition, then most of the data structure of that symbol may then be discarded, leaving just the boundary layer, which is then called as if it were the complete symbol. Certain layout restrictions are placed on the designer. To effectively use the hierarchical features of ROWAN, a designer must adhere to rules about the cell bounding box, see Fig. 2.7. If these rules are not obeyed, ROWAN cannot detect hierarchy and will check all cells which are intruded upon. The hierarchical features of ROWAN provide great speed increases, especially on sparse designs, as long as the designer conforms to the hierarchical layout restrictions.

2.4.2 Technology Independent Design Rule Checking

Technology Independent Design Rule checking calls for increased input to the DRC Tool. ROWAN includes a separate PASCAL programme called a design rule compiler[42]. This programme takes a text file describing the design rules for a particular fabrication technology and produces a complete set of tables suitable for use by ROWAN. This file is called a *Technology File*. The language used for expressing the rules in this file is an embedded language and is actually a restricted list of PASCAL procedures. This feature of ROWAN has already been exploited to provide accurate checks of designs

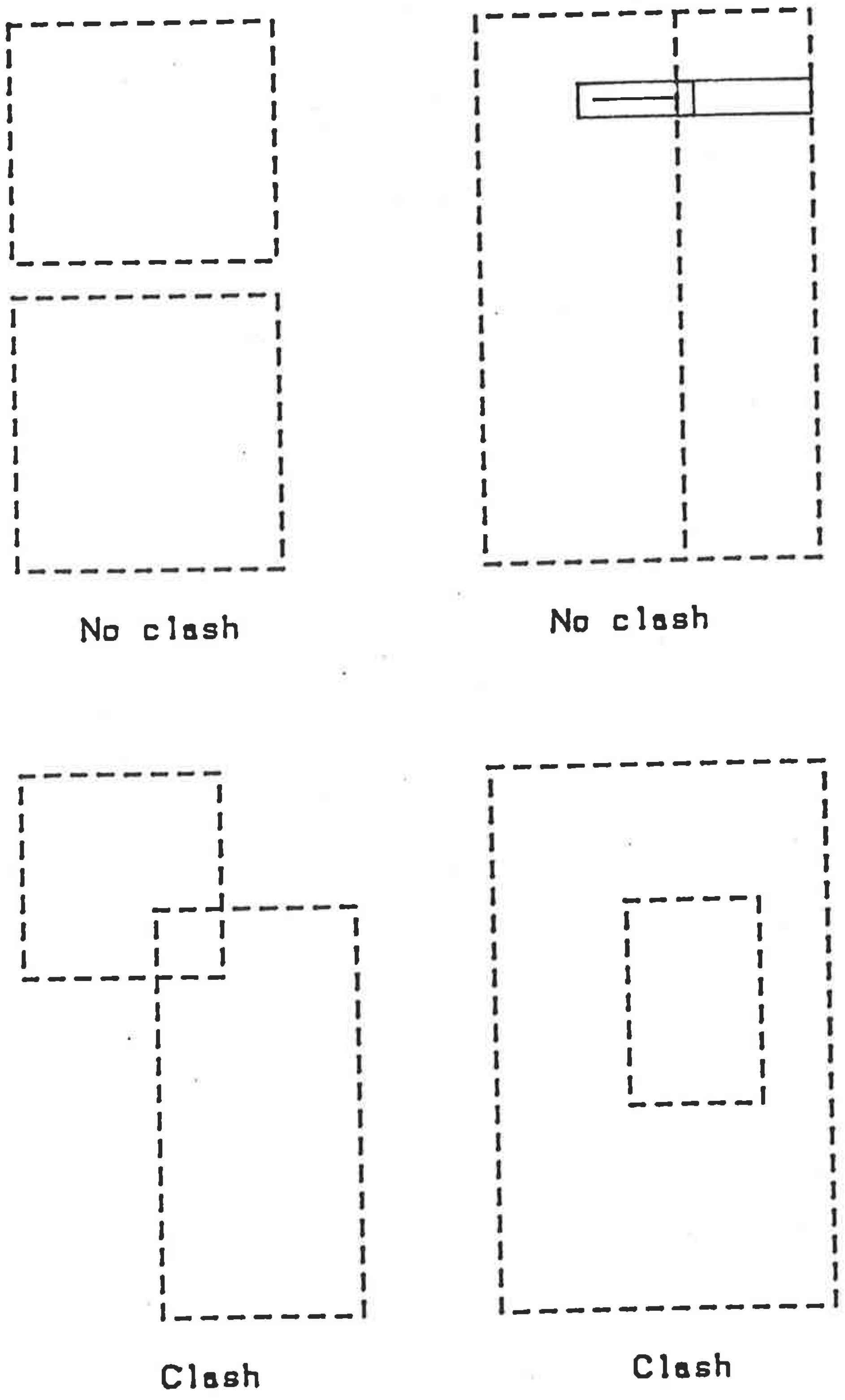


Figure 2.7: Examples of ROWAN symbol interactions. The dashed lines represent the cell bounding boxes. If the cells clash, the hierarchical nature of the design is lost. Note that wires joining to a cell may have their half-width protrusions entering the cell bounding boxes without clashing.

under a number of different technologies at the University of Adelaide. These technologies are, the Mead and Conway nMOS technology, two ***Jet Propulsion Laboratory*** (**JPL**) CMOS technologies, and the Eshraghian and Weste[48] CMOS technology. These Technology Files contain an average of about 65 statements.

The obvious advantage of a Technology File means a new Design Rule Checker can simply be created within a few hours, for any technology using geometrical design layout rules. This feature alone makes ROWAN a very versatile tool, but considering its hierarchical checking algorithm, its efficient storage and speed of operation, it is considered to be an extremely flexible DRC. ROWAN is another tool that does not accept non-manhattan geometries.

Another DRC, CHECK[42.1], is an extension of work done in ROWAN which makes it an even more versatile DRC. Like ROWAN, its checking is corner based, uses a technology file, but can either be run as an hierarchical checker like ROWAN, or as a non-hierarchical tile based DRC.

In its tile based mode of operation, a design is split into a number of square windows, ususally 512λ square but with some flexibility to cater for small designs, which completely cover the design area. Each of the tiles making up the complete design are checked individually. As a result, any hierarchy involved in the design is lost at the tile partitioning stage. This method of analysis is used in CHECK as even modest sized IC designs require the use of more Virtual Memory than can currently be reasonably allocated to users on a multi-user computer. As a result, if CHECK were to be operated without its tiling feature, its algorithm would place severe restrictions on the size of circuits designed on multi-user systems.

Most designers find it difficult to confine themselves to the hierarchical design restrictions placed upon them by the DRC's ROWAN and CHECK for most efficient use of their hierarchical checking features. Non-hierarchical designs require very significant amounts of storage for checking using the hierarchical routines. The tile based approach was implemented to avoid this problem.

An unusual feature of CHECK provides for the optimization of the input CIF code. This feature takes hierarchical CIF and flattens it, that is, it replaces CIF symbol calls with instantiated CIF. All overlapping geometries are also altered to form abutting geometries. This feature is intended to make plotting and Circuit Extraction (to be discussed in detail in the next section) more efficient.

2.4.3 Design Rule Checking Input and Output

As tools become more sophisticated they require and produce more information in an effort to make the use of such tools more ***User Friendly***, a term often employed to indicate the ease of use of software or hardware for novice users.

The input to the DRC programmes of all of the DRC's discussed in this section is CIF, as is required for MPC prototyping. However, the tools ROWAN and CHECK also require the use of Technology Files. The Technology File input to these tools is done by the use of embedded languages similar to those used for design layout as discussed, except that these embedded languages are used for the purpose of creating a new design rule checker. The input is via a restricted set of PASCAL procedures. As an example, the procedure set for ROWAN is summarised as follows:- ***lambda, layers, layer, grow, rules, width, spacing, separation, extension, overhang, define, context, constraint and endrule***. A sample technology input file may be seen in Appendix C. ADRIC also used an extra file which was used as an I/O file. It's purpose was to keep track of previously validated symbols to enforce ADRIC's unique method of hierarchical design rule checking.

The output from the various DRC's varies quite significantly depending on the tool. Baker's DRC provides a single output file, a ***.DRE Design Rule Error*** file. This file is often very large due to the fact that Baker's DRC reports the same error every time it is encountered. The output tends also to be cryptic and not particularly easy to follow. ADRIC produced two output files, the ***.VSS, Validated Symbol Skeletons*** file and the *.DRE* file. ADRIC need not be considered in any great depth owing to its incomplete state, yet its unique DRC algorithm necessitating the two file output is of interest. ROWAN produces a single output file, also called a *.DRE* file. The file reports errors as they occur, once only and also contains some user accounting information. CHECK produces very similar output to ROWAN.

2.4.4 DRC Conclusions

Many different types of Design Rule Checking programs are available for use under a range of operating systems. The latest make use of design hierarchy as well as technology file input to make the tools efficient and technology independent. It is interesting to note that hierarchical checking does not necessarily provide great advantages unless the designer uses an hierarchical design methodology. This is in most cases not a trivial task. The technology file notion does however provide great versatility by allowing the generation of an essential tool in a matter of hours instead of months.

Comparison of Design Rule Checkers, ROWAN and CHECK		
Subject	*ROWAN*	*CHECK*
Technology Independence	Excellent	Excellent
False Errors	Minimal	Minimal
Hierarchical	Yes	No
Algorithm	Corner Based	Corner Based
Language	PASCAL	PASCAL or C
Operating Systems	VMS	VMS or UNIX
Non Manhattan Geometries	No	No

Table 2.2: A comparison of statistics indicating the versatility of two DRC's. The speed of checking will vary for designs incorporating different degrees of hierarchy. Note that CHECK may be modified to provide for either non-hierarchical or hierarchical checking. If this happens, it would be by far the more versatile of the two tools.

Design Rule Checkers as individual CAD tools will probably become redundant as Circuit Extraction and DRC tools often use very similar algorithms for the solution of different problems and therefore lend themselves to efficient combination. However for the purposes of this thesis, table 2.2 is presented as an analysis of two versatile design rule checkers operating on the basis of two different philosophies.

2.5 Circuit Extractors

After checks have been made to verify the mask layout, further checks must be made to verify the electrical operation of the design. This verification is done using Simulators and Electrical Rule Checkers, CAD tools which will be discussed in later sections of this chapter. However these CAD tools do not accept CIF as input. As a result, an intermediate tool is required that can produce the appropriate input information for these tools, from CIF.

A *Circuit Extractor* is a program designed to take mask layout information (eg. CIF) as input and produce output consisting of a network of the electrical devices represented by the layout. The output of a circuit extractor is called a *Wirelist*, (although the amount of output can vary for different circuit extractors), and can be fed into other CAD tools. Different Circuit Extractors can vary in a number of ways. They can differ in

1. Their use of hierarchy present in the design.
2. The constraints that they impose on the designer.
3. The amount of detail present in their output.
4. The Algorithms they use to locate devices and find Connectivity.

This section examines a number of different extractors and discusses the differences between them. The extractors discussed are **CIFPLOT**[43], **MEXTRA**[44], **HEXT**[45], **ACE**[46] and **NET**[47]. A brief introduction will also be given to tools combining both Circuit Extraction and Design Rule Checking.

CIFPLOT and MEXTRA were both written by Fitzpatrick. They provide output acceptable to a variety of CAD tools in the Berkeley Design Suite[43]. They are both are both ***flat*** (non-hierarchical) extractors that accept CIF. CIFPLOT can handle non-manhattan geometries while MEXTRA cannot. As a result, MEXTRA is an order of magnitude faster than CIFPLOT and has an approximately linear growth rate. MEXTRA output is essentially the same as CIFPLOT output and so the two tools will be considered together. They produce output suitable for use with a Static Electrical Rules Checker (MOSERC), a Switch Level (logic) Simulator (MOSSIM), and a Circuit Simulator, (SPICE). CIFPLOT also has the ability of being able to produce plots of the CIF with nodenames and numbers superimposed for visual inspection. Unfortunately the extraction processes in these two simulators only handle simple transistors (see Fig. 2.8). The resultant extracted circuits will not always be accurate. Both programs are written in C and have the technology rules tightly embedded in their code, which makes them technology dependent. MEXTRA uses a corner based algorithm for circuit extraction and was originally developed for nMOS extraction.

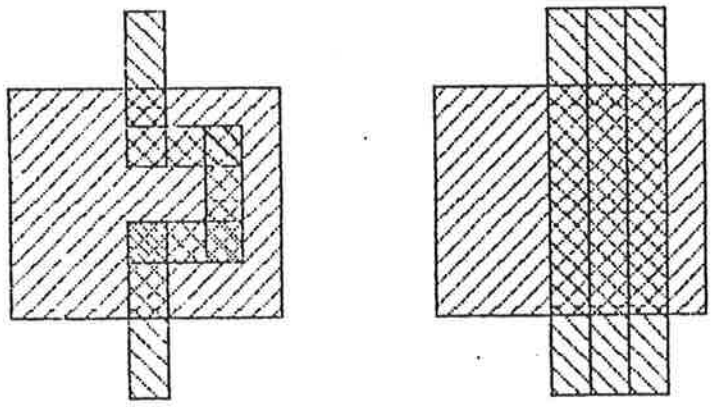


Figure 2.8: A transistor as it actually appears on the left, and as it is recognised by MEXTRA on the right. MEXTRA resolves all transistors to be simple rectangles with x and y dimensions equalling the total x and total y dimensions respectively.

HEXT is an hierarchical circuit extractor. It has also been developed for nMOS designs. It is unusual in its approach because it uses a two part process, a front end and a back end. The front end is responsible for eliminating overlapping symbol instances and isolating hierarchical cells, which reduces circuit extraction time. The back end of HEXT is responsible for analysing the geometry contained in each of the distinct windows found by the front end and ultimately combining the adjacent windows to form a complete circuit for the whole chip. The back end is in fact a modified version of ACE. The Algorithm used in ACE restricts the designer to nMOS manhattan geometries. The output of HEXT is an hierarchical wirelist, but because most CAD tools require a flat wirelist, this is not currently a particularly useful feature. HEXT is written in C to run under the UNIX operating system. The advantage of using HEXT is that it forms hierarchical windows from a design without forcing a restricted hierarchical design methodology on the designer as does the hierarchical DRC, ROWAN. However, because of its generality, it loses efficiency in circuit extraction for some designs.

ACE is a flat extractor in that it works on a fully instantiated description of the chip. Again restrictions on the designer are such that only nMOS designs using manhattan geometry can be extracted. The algorithm used is edge based and has advantages over the raster scan method in that empty space and large device structures are extracted easily. Its disadvantages are that it does not handle nMOS butting contacts and its method of determining transistor parameters is crude[46].

The extractor NET introduces new features to the circuit extraction process which makes this tool very versatile. NET adopts features of ACE and solves the faults, mentioned previously, that are characteristic of CIFPLOT and MEXTRA, as well as providing a technology independence by the use of the technology file concept. NET's versatility is enhanced by being written in both C and PASCAL for use on UNIX or VMS respectively. NET requires manhattan CIF input while its output can be used with all of the CAD simulators and Electrical Rule Checkers already mentioned as well as the tools produced by ISD. These tools are PROBE, a simulator, and ELEC, an Electrical Rules checker. Its output can also be used to display node names on a mask level plot of the design. Currently NET is functional for both nMOS and CMOS technologies.

To conclude this discussion on extractors it is interesting to note that some of the distinguishing features of circuit extractors and Design Rule Checkers are similar, for example the algorithms used for their separate functions and the fact that they both require CIF input. New tools are emerging which exploit these similarities to combine both functions, Design Rule Checking and Circuit Extraction, into one CAD Tool. Shand[49] describes such tools using hierarchical analysis methods. This thesis however looks at each function separately on a comparitive basis enabling the reader to more fully understand the complete checking process.

Comparison of Circuit Extractors MEXTRA and NET		
Subject	*MEXTRA*	*NET*
Technology Independence	None	Excellent
Accuracy	Not Good	Very Good
Hierarchical	No	No
Algorithm	Corner Based	Edge Based
Language	C	PASCAL or C
Operating Systems	UNIX	VMS or UNIX
Non Manhattan Geometries	No	No
Input	CIF	CIF
Static Rules Check Output	Yes	Yes
Logic Simulator Output	Yes	Yes
Circuit Simulator Output	Yes	Yes

Table 2.3: A comparison of statistics indicating the versatility of two Circuit Extractors. NET is a much more versatile tool as it is more accurate, it provides output for all of the tools that MEXTRA does and more, and it is supported under two different Operating Systems.

2.6 Electrical Rule Checkers

Once circuit extraction has been completed, the abstracted electrical circuit is presented as output usually in a number of different forms, and is available for use by other CAD tools. One type of tool that uses extracted output is a Static Electrical Rules Checker. Two such tools are **MOSERC**[50], and **ELEC**[51]. They will be used to illustrate electrical rule checks made on both nMOS and CMOS circuits.

MOSERC checks only nMOS circuits. It was written by F. Baskett at Stanford University. It is written in the language C to run under the UNIX operating system.

MOSERC looks for unusually configured circuit elements. For example, it looks for such things as nodes that cannot be pulled up, nodes that cannot be pulled down and bad ratio values for inverters.

ELEC is similar in its operation to MOSERC. It was written in PASCAL to run under VMS by R. Kinnear at the University of Adelaide and a version written in C for UNIX also exists. Both ELEC and MOSSERC provide error file outputs that can be used by the designer for rectifying bad mask layout. Table 2.4 shows a comparison of these two tools.

Comparison of Static Electrical Rules Checkers MOSERC and ELEC		
Subject	*MOSERC*	*ELEC*
Technology Independence	None	Excellent
Accuracy	Good	Good
Hierarchical	No	No
Language	C	PASCAL or C
Operating Systems	UNIX	VMS or UNIX
Non Manhattan Geometries	No	No
Input Tools	MEXTRA and NET	NET

Table 2.4: A comparison of statistics indicating the versatility of two Static Electrical Rules Checkers.

2.7 Simulators

Simulation tools are used for checking electrical functionality as well as for optimizing certain performance parameters of designs, particularly speed and power dissipation. Just as large scale designs are produced hierarchically, so simulators exist for different levels of electrical functionality checking.

2.7.1 Circuit Simulators

An example of the lowest level of simulation tool is **SPICE**[52]. This is a well known tool that is written in FORTRAN for use with input files written in the SPICE format. It is in fact a *Circuit Simulator*. Circuit Simulators model electrical circuits of transistors, capacitors, resistors, diodes etc. to determine static and transient behaviour of node voltages and branch currents within networks. The results can be presented in a number of different ways but are usually presented as plots of voltage or current versus time. The input to this simulator can be hand generated or produced as output from a circuit extractor.

```
*
*       Device Parameters for a CMOS Inverter
*
.SUBCKT INV1 1 2 3
*       DENOTING INPUT, OUTPUT AND SUPPLY RAIL VDD
M1 2 1 3 3 PTYPE L=4U W=8U AD=80P AS=80P PD=20U PS=20U NRD=1.25 NRS=1.25
M2 2 1 0 0 NTYPE L=4U W=8U AD=80P AS=80P PD=20U PS=20U NRD=1.25 NRS=1.25
.ENDS INV1
*
*       Process Parameters for the P and N type transistors of the Inverter
*
.MODEL PTYPE PMOS(LEVEL=3 VTO=-0.9 GAMMA=1.0 RSH=100
+ CGSO=4E-10 CGDO=4E-10 CGBO=2.8E-9
+ PB=0.7 NSUB=3E16 XJ=6E-7 LD=5E-7 UO=200 UCRIT=8E4
+ UEXP=0.15 UTRA=0.3 TOX=5E-8 PHI=0.65 TPG=-1
+ CJ=2E-4 CJSW=4.5E-10 VMAX=5E4 ETA=2 DELTA=0.1
+ KAPPA=0.5 THETA=1 MJ=0.5 MJSW=0.33 PHI=0.65
+ JS=1E-4 NSUB=5E15 UCRIT=8E4
*
*
.MODEL NTYPE NMOS(LEVEL=3 VTO=+0.9 GAMMA=1.5 RSH=20
+ CGSO=5.2E-10 CGDO=5.2E-10 CGBO=2.8E-9
+ PB=0.7 NSUB=1E15 XJ=8E-7 LD=6.4E-7 UO=450 UCRIT=8E4
+ UEXP=0.15 UTRA=0.3 TOX=5E-8 PHI=0.73 TPG=+1
+ CJ=3.3E-4 CJSW=9E-10 VMAX=5E4 ETA=2 DELTA=0.1
+ KAPPA=0.5 THETA=1 MJ=0.5 MJSW=0.33 PHI=0.65
+ JS=1E-4 NSUB=2.5E16
*
```

Figure 2.9: SPICE Device Parameters and Process Parameters for a CMOS circuit showing both the nMOS and the PMOS devices. These parameters are used by the LEVEL 3 SPICE. Not all of the Process Parameters are used.

The input consists of circuit elements and their connections as well as *Device* and *Process Parameters* for elements that require modelling. For example, PMOS and nMOS transistors each have separate sets of Device Parameters for CMOS technologies, as do enhancement and depletion mode transistors for nMOS (see Fig. 2.9). SPICE is a complex circuit simulator which has three levels of model complexity for MOS devices. The different levels are accessed by using one of the model parameters. SPICE has 8 device parameters which describe the MOS device geometry while it has up to 37 model parameters which describe the MOSFET model. The three different models used are the ***Schichman-hodges model*** at level 1, the ***MOS2 model*** at level 2, and a semi-empirical model at level 3, where level 1 represents the simplest model and level 3 the most complex.

With the circuit parameters appropriately set, the simulator is then activated by modelling circuit behaviour in a series of incremental time steps with the increments commonly in the order of nano-seconds or less. Initial conditions for voltages may be set and function generators may be used with the simulator also. The result can be quite an accurate simuation of the circuit operation, depending on the model complexity and parameter accuracy. For SPICE, device parameters are difficult to obtain and as a result, simulations could possibly be erroneous in orders of magnitude. Chapter 4 shows results obtained for an nMOS circuit fabricated using the MPC process compared to its simulated results.

Extremely complex models such as those provided in SPICE need only be used for particularly crucial circuit components, as the trade-off for using a very accurate circuit simulator is the long time taken to run such simulations. Circuit Simulators, even for the less complex models are commonly only used for small circuits (no more than a few hundred elements).

2.7.2 Timing and Logic Simulators

Another form of simulator is the ***Timing Simulator***. This type of simulator is not unlike a circuit simulator but is characterised by two features. The first feature is that it exploits the fact that models can be described by lookup tables with subsequent interpolation and geometrical scaling. The second feature is the fact that only active nodes need be altered at the time steps used. This type of simulator could rightly be considered a circuit simulator using a less accurate model for the devices. The advantage in reducing model complexity is reduced simulation time. The disadvantage is of course the less accurate result, however this need not be of great concern as long as the particular circuit being simulated is not crucial to the system function. The analogue output produced by the timing simulator while not totally accurate, still gives the designer a good first approximation to his circuit functionality. An example of a simulator that could be called a timing simulator is **PROBE**[54] if the previous definitions are used. This is a PASCAL program written by M. Pope at the University of Adelaide to run under VMS. The difference between Timing and Circuit Simulators is probably most significant in the nomenclature only.

When systems begin to grow larger than a few hundred components, circuit simulation can become prohibitive. Consequently, the VLSI circuits currently being designed and those of the future cannot be effectively simulated with such simulators, other methods giving great simulation speed increases are required. As the designer departs from the lowest level of hierarchy, a new level of simulation tool must be used to verify the circuit operation. The *Logic Simulator* provides the simulation speed increase that the designer seeks. It does so at the expense of discarding analogue type analysis and replacing it with discrete digital level analysis. An example of a logic simulator is **MOSSIM**[53] written at MIT by Terman. It is written in C to run under the UNIX operating system and has also been converted to run under the VMS operating system.

Comparison of Logic Simulators MOSSIM and PROBE		
Subject	*MOSSIM*	*PROBE*
Technology Independence	None	Excellent
Hierarchical	No	No
Language	C	PASCAL
Operating Systems	UNIX	VMS
Input Tools	MEXTRA and NET	NET

Table 2.5: A comparison of features indicating the versatility of two Logic Level Simulators.

The input to MOSSIM is obtained from the output of a circuit extractor while the output is a file detailing nodes with the characters ***0***, ***1***, or ***X*** indicating low, high or undefined or unknown voltages respectively, at a particular node. Logic simulators by themselves are not good enough to simulate circuit operation and should only be used in conjunction with a reliable circuit or timing simulator.

In conclusion, simulators at higher levels exist and are often referred to as ***Register Transfer Level*** or ***System Simulators***. However to discuss these tools under the heading of this chapter would be inconsistant and so this discussion is left for the next chapter.

2.8 MAGIC

The tool MAGIC, has been mentioned in passing as it is a recently developed tool which is as yet unavailable at the University of Adelaide. This tool does warrant further investigation and a paper by Ousterhout et al[29.1] provides a detailed summary of its features. Discussion of this tool has been left until the concluding section of this chapter as it incorporates many of the tools discussed after the section on Geometry Editors, that could not effectively be explained in that section.

The features of MAGIC include increased flexibility and power by allowing for simple circuit modification using a ***corner stitching*** data structure[29.1]. With most Geometry Editors, modification of a layout is almost as difficult as re-entering the whole layout[29.1]. The MAGIC data structure allows for the efficient implementation of automatic interactive Design Rule Checking that is done continuously and incrementally during editing sessions. It allows layouts to be compacted and stretched in a manner that allows the design rules to be obeyed and the circuit to maintain its structure. It provides routing tools that can work under and around existing wires in channels and it provides for an hierarchical Circuit Extraction directly from the MAGIC internal database.

The MAGIC design style resembles those of symbolic systems such as MULGA and VIVID, see Chapter III, the difference being that it provides precise geometry sizes and positions. As a Geometry Editing tool, it is the most sophisticated mentioned in this thesis, as it incorporates a system of tools working on its mask level output description.

2.9 Conclusions

The previous discussion offers a quick summary and comparison of a basic low level CAD Tool Set for VLSI design. Tools from the different categories presented allow a designer who has access to such tools to efficiently proceed with remote design of IC's using either an nMOS or CMOS process. The tool set presented has evolved over a period of approximately four years with much effort being devoted not only to the development of many new, more efficient tools, but also to the troublesome effort of porting software written to run under the UNIX operating system, to VMS machines and vice versa.

Summary of Preferred Low Level CAD Tools for VLSI Design from University of Adelaide Design Environment		
Tool Category	*nMOS Tool*	*CMOS Tool*
Layout Language	BELLE1	BELLE1
Geometry Editor	KIC	KIC
Design Rule Checker	CHECK	CHECK
Circuit Extractor	NET	NET
Circuit Simulator	SPICE	SPICE
Electrical Rules Checker	ELEC	ELEC
Logic Simulator	PROBE	PROBE

Table 2.6: Preferred Low Level Tools used in the University of Adelaide VLSI Design Environment as of late 1985.

The low level tool set discussed fulfills all of the requirements needed in the MPC remote design process. The diagram in Fig. 2.10 shows how each of the tools used at the University of Adelaide fit into the VLSI Design Environment and this diagram used in conjunction with Fig. 1.5, clearly describes the steps a designer needs to iterate through to prototype a circuit in silicon.

Conclusions regarding which of the tools discussed in this chapter should be used for low level design, are presented subjectively as introductory material to a complete set of Low Level CAD tools. The preferred tools to be used are listed in Table 2.6.

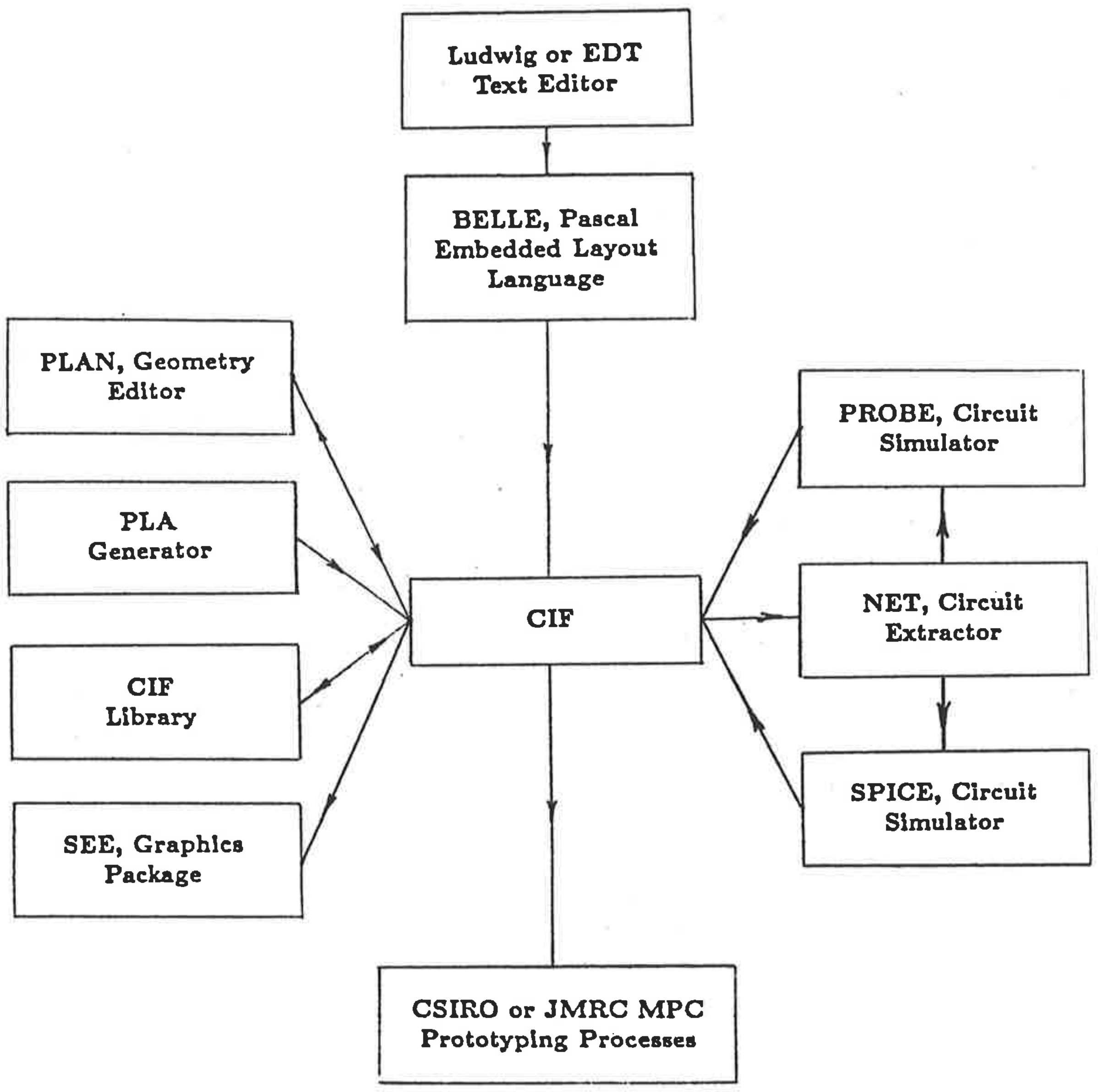


Figure 2.10: The University of Adelaide Low Level CAD Tool Environment for VLSI Design. (Note that the LUDWIG[85] text editor is the commonly used text editor at Adelaide University on both VMS and UNIX DEC machines).

CHAPTER III

Higher Level CAD Tools

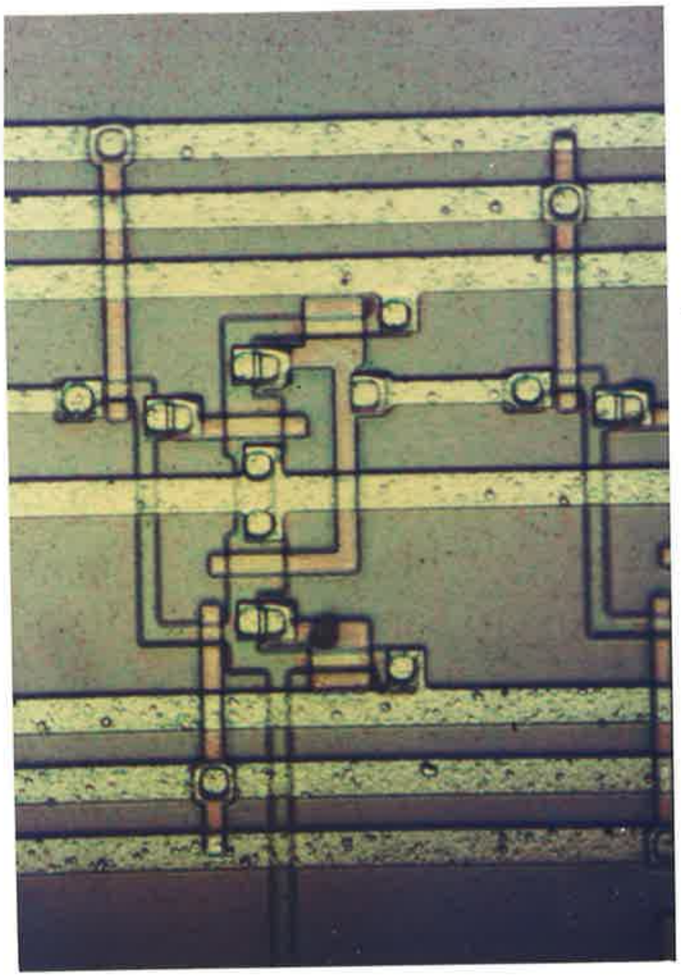
Contrary to accepted lore, geometry goes into masks, not silicon. Transistors and wires go into silicon, and these, not squares and rectangles, are what must be represented to capture the logical intent of a circuit. Today, however, mask geometry serves as the universal base-level representation. Mask geometry stands in relation to silicon as does absolute binary code to a computer - neither has any utility as a design representation. Either can be re-created from a higher-level representation if needed.

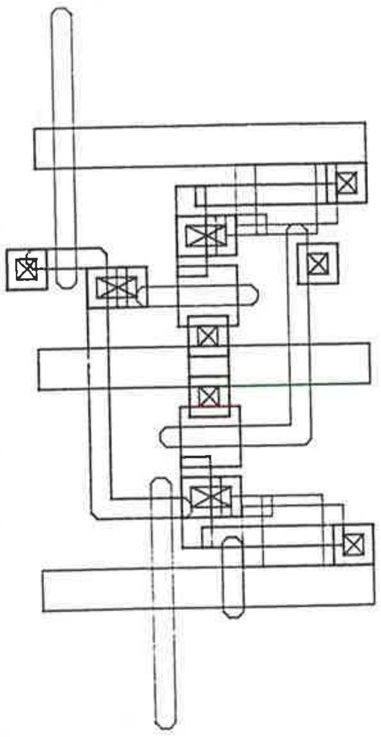
-Mead and Lewicki†

A new philosophy will be discussed in detail and a complete set of CAD Tools using the philosophy, described. Other Tools, on even higher levels will also be discussed, although they will be considered as individual Tools rather than components of an Integrated Tool Set.

The concepts of IC design using MPC processes as outlined in the first two chapters of this thesis remain unaltered when considered in conjunction with the CAD Tools discussed in this chapter. These new Tools are of interest because they apply a different philosophy to IC Design in an effort to make the task less complex and more efficient. The new philosophy is based on the concern that ***Circuit Designers*** should only need to know about ***Circuit Elements*** such as transistors, wires or contacts. Rather than add the task of understanding mask level geometries and their interrelationships as is required by the Low Level Tools already discussed, this new philosophy removes the need for mask level knowledge, from the designer and embeds this knowledge in the tool set.

† Carver A. Mead & George Lewicki, *Silicon Compilers and Foundries will Usher in User-Designed VLSI*, Electronics, August 11, 1982, pp 107-111.





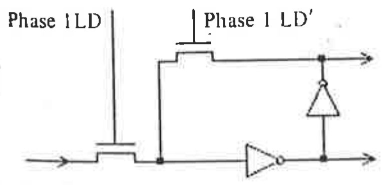


Figure 3.1: A photomicrograph of the shift register cell in Project C4 on the left with a computer generated plot on the right. Both pictures show the different Mask Level Layers used in the design yet neither bares any obvious physical resemblance to the circuit schematic of the design, also shown above, indicating the enormity of the change from circuit to mask layout.

3.1 Introduction to Symbolic Level Design on a Virtual Grid, VIVID

The language BELLE (see Appendix 5) posseses mainly geometric primitives, as does any Embedded Layout Language operating at the Mask Level, (eg. boxes, polygons, etc.). These geometries describe the different mask patterns used in the design of IC's as can be seen by the comparison of a photomicrograph of a cell with a plot of the same cell, see Fig. 3.1. If IC design is carried out at the mask level, the designer is confronted with two major tasks in his design procedure. The first task is to correctly produce a traditional electronic circuit design at the transistor level. The second task is to translate this circuit design into a mask level description of the same circuit which means the designer is then faced with the problem of mastering the manipulation of seven or more mask layers in relation to a non-trivial set of Design Rules, to complete this task. This has necessarily to be done on a fixed spatial grid providing much room for error in mask layout, particularly for first time designers.

A new concept suggesting the use of symbolic circuit primitives rather than mask level geometric ones, has been considered and a complete set of CAD Tools has been developed to enable this design philosophy to be realised. Using elements such as transistors, wires, contacts, and pins as primitives, designs can much more easily be completed by a circuit designer, as he is dealing with fundamental primitives that he already understands.

To realise the aim of this new philosophy, it was necessary to introduce a new tool to the set already discussed in Chapter II. This tool is described as a Circuit Compactor and is discussed at length in section 3.1.4. To enable the designer to remain relatively oblivious to mask level information, this tool had to be designed so that it could take a primitive circuit description and manipulate it by moving the circuit primitives to provide a design rule error free and reasonably compact mask level layout of the design. As well as this totally new tool, the new philosophy also uses a ***Virtual Grid*** as opposed to a fixed grid.

The Virtual Grid concept was first used in the ICDL language that formed the basis of the MULGA system[59][60]. The grid lines in such a system are used by the designer to specify relative positioning of the circuit elements as opposed to their absolute placement as in a fixed grid system. The Virtual Grid concept enables insertion or deletion of horizontal or vertical grid lines in an effort to increase the power of Circuit Editing Tools, and to enable the operation of the Compaction Tool, which moves these virtual grid lines relative to one another to achieve the smallest design rule legal circuit. It is the Compactor which automatically determines the final positioning of the mask geometries, not the designer. Virtual Grid symbolic layout is most useful in Leaf Cell design. Using the design language discussed in section **3.1.1**, the designer may construct larger cells by using an abutting method and/or a technique called ***block-interconnect***.

The introduction of these two new concepts, compaction and virtual grid layout, to those tools discussed in Chapter II, primarily highlights the differences between the two Tool Sets, and forms the basis for all of the differences between the two Tool Sets. The Integrated Tool Set to be discussed in this chapter is the **VIVID**[56] (***Vertically Integrated VLSI Design***) system. It is a system originating from the ***Microelectronics Center of North Carolina*** (**MCNC**), and has been developed primarily for the design of MOS IC's. It is written in C to run under the UNIX operating system. Work has also been done by researchers at the University of Adelaide and in turn by ISD to produce a VMS version of the VIVID system.

3.1.1 A Better Circuit Description Language (ABCD)

In Chapter II, early Design Tool discussion considered CIF and the Embedded Layout Language BELLE. Both languages used mainly geometrical primitives to produce the mask level representation of the layout. In this chapter, the first Tool to be discussed will be the design language used by the VIVID system.

Languages describing IC designs may exist at various levels of abstraction. The various examples range from ***Hardware Description Languages*** (**HDL**) describing high level structural and behavioural attributes of designs, as in MacPitts, (see Fig 1.4 (b)), down to purely physical description languages that define the interrelation of wafer fabrication mask levels needed to realise a design, as in CIF. Recent research has centered around using Artificial Intelligence (AI) to produce expert systems that transcend more levels of abstraction than has been possible in the past. An example is the **CADRE**[57] system.

Languages that specify behavioural description and upon compilation yield a physical mask description may be classified as Silicon Compilers, in comparison to the computer programming language equivalent. The language used as the heart of the VIVID system, ABCD[58], has been developed from the lowest level of IC design languages and as such could be described as a Silicon Assembler. As in most assembly languages, the language statements represent a small number of the lowest level primitives. In contrast, compiled languages use statements that can be compiled into a very large number of the lowest level primitives. Ultimately, the lowest level primitives are the mask geometries.

The advantages of such a language are many. A one to one correspondence between a line of text in the language and a single circuit element is maintained making interactive circuit editing feasible. Again drawing the analogy to assembly languages, a symbolic transistor representation could be considered equivalent to an assembly level macro. Symbolic references in the textual description are translated into internally maintained linkages thereby explicitly maintaining circuit connectivity, a feature which has been recognised as imperative for all future design systems[61].

ABCD forms the basis of the VIVID system. It uses primitives such as transistors, wires, pins and contacts to describe designs. It uses a virtual grid layout and is based on the ICDL language used in the MULGA symbolic design system developed at Bell Laboratories, and partially on the language used in the ICSYS system[64]. (See Fig. 3.2 for examples of ABCD in text and graphic form). The primitives used are intended to define circuit primitives rather than geometric ones as in the low level design systems.

It is a text-based language as opposed to a machine readable format such as CIF. This is a useful feature as easily understood formats are more rapidly accepted by all who work with them leading to greater understanding by all members of design teams as well as tool designers. Such a format can be and is used as input to a range of other tools to perform all of the checking functions discussed in Chapter II. The number of checking functions that need to be performed is of course reduced by the introduction of the Compaction Tool (ie. Design Rule Checking is performed within the Compaction Tool).

```
#
# $Header$
#
# Created by ICE 1.1 on Tue Apr  8 08:22:33 1986
#
begin inv bbox=(0,0,6,10)
    out_e:      pin       metal (6,5)
    in_w:       pin       poly (0,5)
    vdd_nw:     pin       metal (0,10)
    vdd_ne:     pin       metal (6,10)
    vss_se:     pin       metal (6,0)
    vss_sw:     pin       metal (0,0)
    c2:         contact   vdd (3,10) or=n
    c1:         contact   vss (3,0) or=n
                wire      poly (2,5) (0,5)
                wire      poly (3,2) (2,2) (2,8) (3,8)
    cd1S:       contact   autocontact d1.s or=n
                wire      metal (cd1S) (3,0)
                wire      metal (3,5) (6,5)
    cd1D:       contact   autocontact d1.d or=n
    cd2S:       contact   autocontact d2.s or=n
                wire      metal (cd2S) (cd1D)
    cd2D:       contact   autocontact d2.d or=n
                wire      metal (cd2D) (3,10)
    d2:         device    p_type (3,8) or=n
    d1:         device    n_type (3,2) or=n
                wire      metal (0,10) (6,10)
                wire      metal (0,0) (6,0)
end inv
```

Figure 3.2 (a): An ABCD description of the CMOS Inverter shown on the following page. Notice the difference in primitives used in this language compared to those used in BELLE in Fig. 2.1.

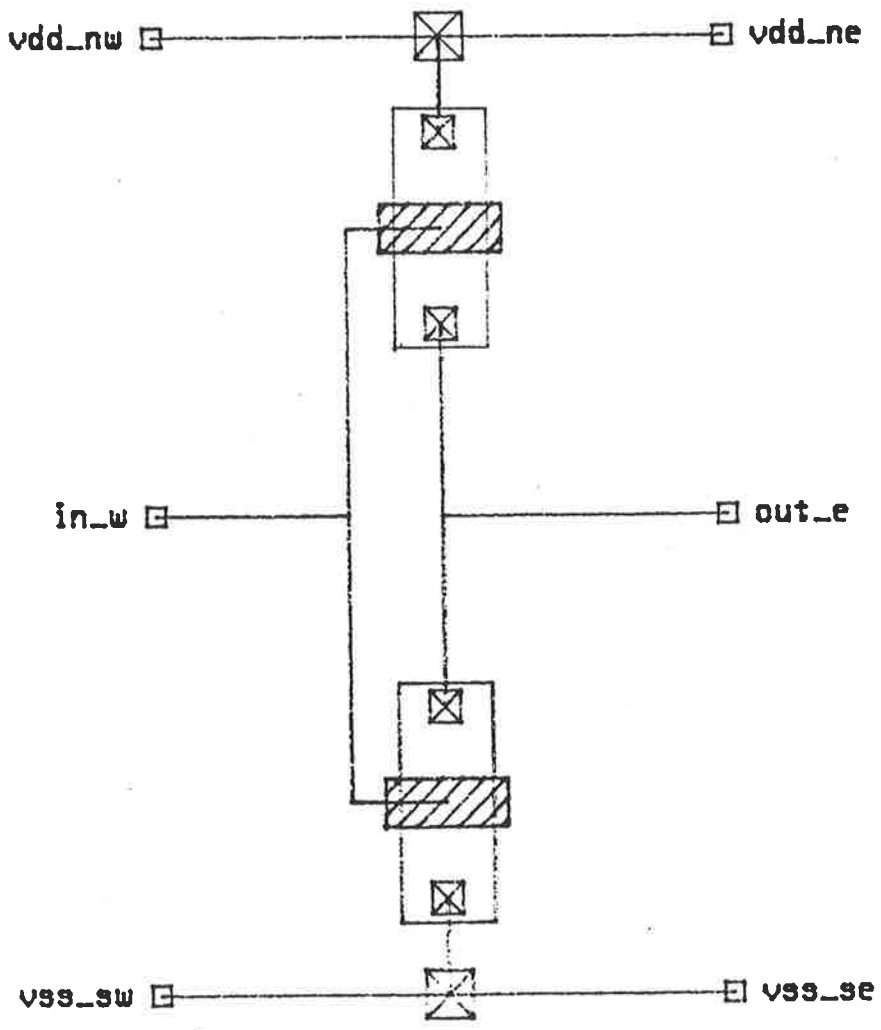


Figure 3.2 (b): The Computer generated plot of the inverter described on the previous page. The elements are all shown on a virtual grid as opposed to an absolute grid layout used for the mask level description.

ABCD in the VIVID system, performs most of the functions of CIF in the Low Level Tool Set. That is, for the Higher Level Tool Set, ABCD has been implemented as the standard interface between a large set of IC design tools. Whereas in the Low Level Tool Set, CIF is used as the standard interface between a large set of design tools. ABCD and CIF appear similar in their abilities to scale up or down to meet the needs of a particular technology, however there are vast differences. In CIF the value of λ can be redefined by altering the scaling field in the **DS** statement. This does not allow for relative size scaling between circuit elements or mask features. The relative sizes will remain constant using CIF. Using ABCD, a virtual grid circuit description is translated into a mask description, and at this time dimensions of its components are specified independently of each other. Since these different elements usually change scale at different rates as technology changes, ABCD is much more flexible than a layout language. Designs may be specified in the ABCD language using either a text editor to generate the code directly or by using a Graphics Editing Tool (see Fig. 2.2 (c)) to visually edit circuit layouts and produce ABCD as output.

Apart from its current flexibility ABCD is an attractive language to use as a standard interface because of the proposed development of the complete VIVID System[58]. Plans exist to develop a structured front end to ABCD to allow procedural design and to extend this approach from MOS to other technologies such as bipolar. In conclusion, the ABCD language forms an excellent base interface language that will allow development from the Assembler level to the Compiler level.

3.1.2 Higher Level Graphic Circuit Editing (ICE)

Designs represented in the ABCD language may be created by way of a text editor if necessary, just as CIF descriptions could also be generated by the use of a text editor. The most efficient method of ABCD circuit description however is by way of the VIVID *Interactive Circuit Editor* **ICE**.

The principles of operation of ICE are the same as those discussed for Geometry

Editors in Chapter II. The difference between ICE and the Low Level Geometry Editors is the fact that the low level tools produce the layout language CIF on a fixed grid as their ultimate output while ICE produces ABCD , the higher level circuit description language using the virtual grid. The importance of the virtual grid becomes most clearly evident when used in conjunction with a circuit editor as it allows a designer to work at the Topological level rather than a geometric level. The virtual grid removes all necessity for design rule considerations from the designer and leaves him to deal with circuit placement only, a function that the designer would normally have been trained for.

The editor ICE is a menu driven program which has six primary features that make it much more versatile than any of the Geometry Editors that have previously been discussed. They are (i) Symbolic Layout, (ii) Virtual Grid, (iii) Element Tracking, (iv) Automatic Wire Routing, (v) Easy to use interactive menus, and (vi) Technology Independence[62].

The Symbolic Layout and the Virtual Grid concepts have already been discussed at length. Use of symbolic circuit elements also facilitates the tracking of an element relative to another. If the designer moves a particular element during the design process, ICE moves any related elements automatically. Wire connections are maintained regardless of the addition or deletion of space during the design process[62].

Automatic Wire Routing is also a feature provided by ICE. This connects symbolic elements within cells and between cells. The points to be connected must be designated and then the router constructs the wiring. Wire routing eliminates complex manual placement of wire networks and contacts[62].

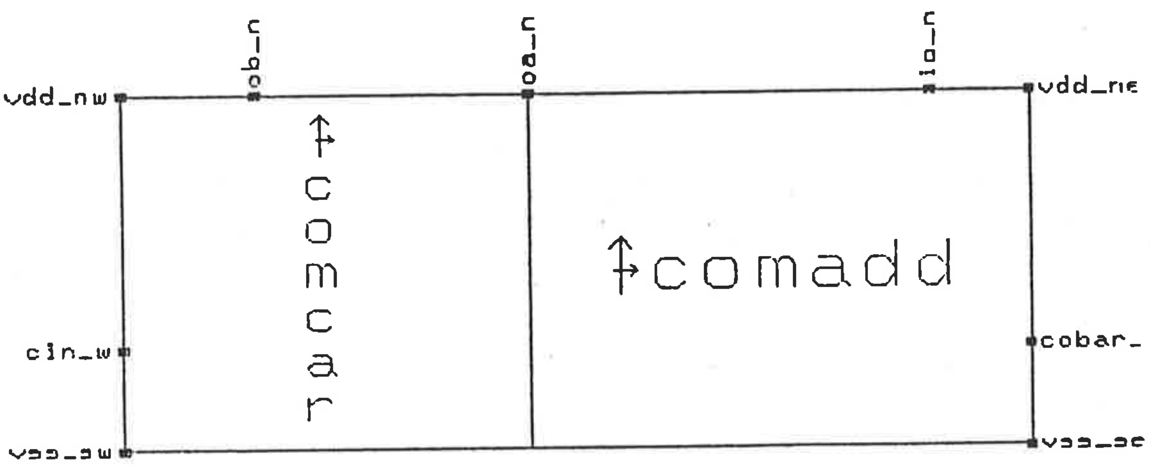
All of the circuit design tasks are provided in predefined menus on alphanumeric and graphics screens. Keyboard input is kept to a minimum allowing the designer to focus attention on the graphics screen as the primary work area. Menu choices are represented in colours and symbols that are easy to learn and recognize. They are organised hierarchically with each menu containing selections that are relevant only to the particular task being performed[62].

Technology independence is achieved through the VIVID ***Master Technology File*** (**MTF**) system which can be modified to suit specific design requirements[62]. Any of the technologies (CMOS, nMOS and SOI) can be chosen to meet the designers' fabrication requirements. Also, various ICE attributes can be altered, such as the appearance of primitives, colours, virtual shapes, and menu organisation[62].

ICE allows the designer to adhere to a rigid hierarchical design structure. At the highest level would be the ***Master Floorplan***, the top level composition cell containing all of the lower level hierarchy. The hierarchical structure of any composition cell in a design can be displayed by using the appropriate menu function. Fig. 3.3 shows ICE output for a component part of a CMOS Adder cell described in detail in Chapter V. The system used to indicate composition cells in ICE is similar to that in KIC, where composition cells are identified by their bounding box and the name of the cell inside the bounding box.

Most of the problems associated with composing cells at the mask level are non-existent in the VIVID system as design rules and hence geometry relationship is no longer an issue. As long as only pin information exists on a bounding box for an ABCD design, there is no possibility of any design violations at the time of composition. This enables a truly hierarchical approach to design and combined with the connectivity information implied through the use of pins, forming the only legal connection between composed cells, all of the advantageous features of Interactive Circuit Editors as mentioned in Chapter II can be found to exist in ICE.

As a component of the VIVID system, ICE is written in C to run under the BSD (UCB) 4.2 UNIX operating system and has been transported to the VMS operating system by researchers at Adelaide University and ISD. The hardware required to run the system is a colour graphics terminal, a monochrome text terminal and a bit pad or



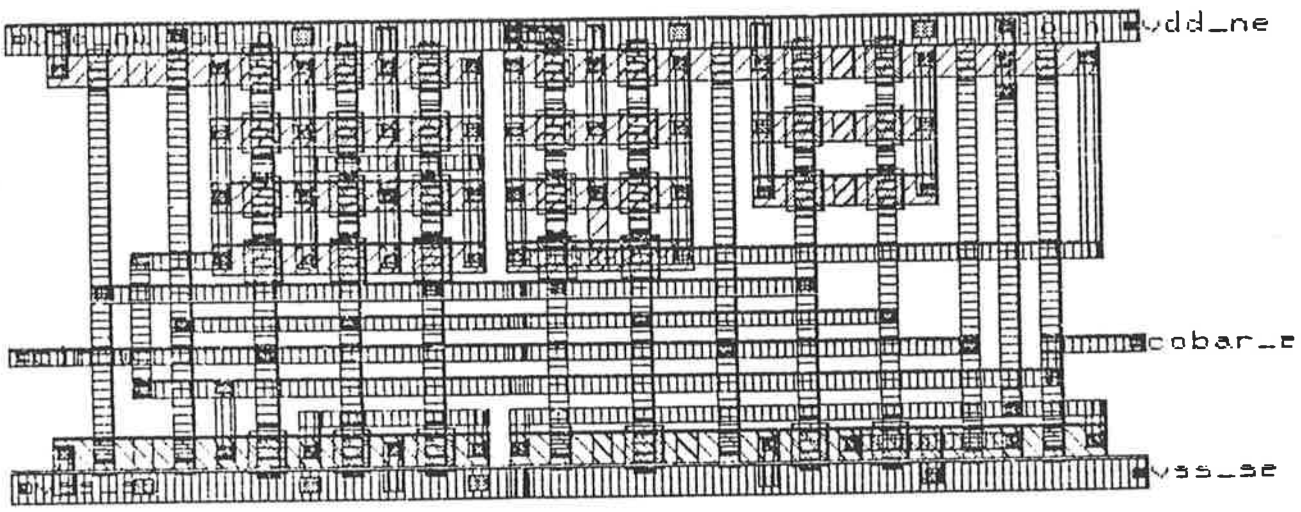


Figure 3.3: ICE output showing composition cells with bounding box and port labels only (above), and with symbolic circuit level detail shown (below).

optical mouse for cross hair control. Fig. 2.2 (a) and (c) show workstation configurations capable of running ICE.

In conclusion, ICE is not only a versatile editor because of the VLSI design philosophy used as its basis, but also provides the designer with a relatively terminal independent tool, as is shown in Table 3.1.

VIVID System Compatible Terminals		
Graphic Displays	*Hard Copy Output Devices*	*Input Devices*
Ramtek 9400	HP 7580 8-Pen Plotter	Summagraphics BitPad
Lexidata 3400	HP 7220 8-Pen Plotter	USI OptoMouse 4000
Lexidata 3700	QMS 1200 Laser Printer	GTCO Data Tablet
AED 512	Versatec V80 Plotter/Printer	
Vectrix VX 384	Printronix 300 Line Printer	
Metheus Omega 530		

Table 3.1: A table produced by MCNC indicating the versatility of the VIVID system measured by its terminal independence as of early 1986. The VIVID system currently runs under 4.2 bsd UNIX on DEC VAX 11/780's and 11/750's and is being developed to run under VMS on the same machines.

3.1.3 Circuit Extraction and Simulation

The higher level tool set allows circuit extraction and simulation prior to mask layout. Using the ABCD language as input, the circuit extractor **ABSTRACT** is used to provide input to one of two simulators, SPICE, the circuit simulator discussed in Chapter II, or **FACTS** a timing/logic level simulator. FACTS is most often used as it gives a much faster response than SPICE and it also allows a more versatile operating interface for the user.

The symbolic level circuit extractor ABSTRACT references the MTF system and calculates the electrical parameters associated with each circuit element. The result of course will not be completely accurate as the actual mask geometries have not been defined at this stage of the design phase. The need for fast design verification however makes it desirable to be able to perform an approximate circuit extraction and subsequent simulation at this stage. The estimates are a good approximation for all elements except wires whose lengths are directly dependent on the final layout size. It is expressed in the VIVID documentation that reasonable estimates of wire length can be made as it is assumed that the spacing between the virtual grid lines will average out over the design[63]. This average grid spacing parameter is coded in the MTF system and can be tuned by the designer according to the technology being used and the performance of the compactor.

As well as circuit extraction, error checking is performed by ABSTRACT and both textual and graphical output is available. Examples of errors detected are overlapping or improperly abutted cells, unconnected or short circuited components, and improperly named signals.

Circuit simulation is performed by FACTS. This simulator has been designed for MOS simulations and can be used with circuits containing as many as several thousand devices. FACTS precalculates tables of simulation values before beginning a simulation and allows the use of a simple first order transistor current model, or the use of a more detailed current model taking into account the effects of saturation, linear and cut-off regions, channel length modulation, drain and source threshold dependence, and capacitance modelling. FACTS also monitors all node voltages and when the changes are small, increases the time step to avoid redundant or insignificant calculations[63].

FACTS output can be varied depending upon the user hardware whereas standard SPICE output is a character plot to a line printer or a numeric list to a file. Often SPICE output can be manipulated to provide more readable output, however, FACTS has been built into the VIVID system in such a way as to use the display routines to enable readable plots on a variety of devices including colour graphics terminals, monochrome graphics terminals, monochrome text terminals and line printers.

The advantage of using FACTS over SPICE is that it quickly provides informative feedback on circuit operation for a designer using the VIVID system. As stated in Chapter II for logic and timing simulators, FACTS is not intended to provide accurate network analysis down to analogue detail, rather it allows swift verification of circuits of up to a few thousand transistors during the design phase.

Included in FACTS are interactive features which enable a simulation process to be halted at any time to allow alteration of circuit parameters such as loads on nodes, or of the display of nodes.

These two tools ABSTRACT and FACTS, allow swift verification of the virtual grid layouts of designs produced using ICE or ABCD. This is an essential feature of the higher level tool set as it allows the designer to stay away from the low level CIF description of the design even in the layout verification stage of the design process. The aim is to provide a highly interactive design environment to increase the efficiency of the design phase. This would not be the case if the designer had to continually move between the symbolic and mask levels.

3.1.4 Compaction to a Standard Mask Description

Different methods for compaction have been suggested[48] by various authors, however only the compactor provided in the VIVID system will be analysed in this thesis.

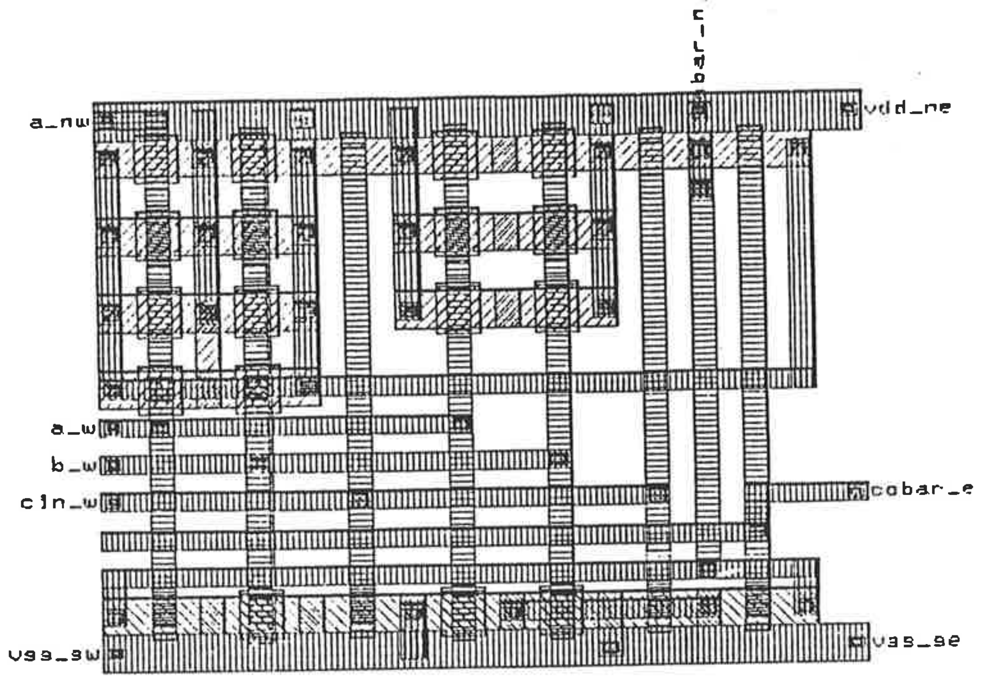
Once a design has been completed at the ABCD level, the task remains to convert the circuit level description of the design into a mask layout description enabling the generation of masks ready for fabrication. This task is handled by the VIVID *Hierarchical Compactor* **HCOMPACT**. The compactor takes ABCD descriptions and generates a **LLAMA** mask description of the circuit. The compactor operates in two distinct steps, leaf cell compaction and hierarchical compaction[63].

An ABCD leaf cell will contain only circuit primitives. The compactor translates the symbols into a mask representation and then spaces them according to the design rules. HCOMPACT solves some of the problems of compaction experienced in the past due to the interdependency of these two tasks. Traditionally the placement of wires and contacts on transistors has caused many problems. HCOMPACT solves this problem by adding flexible wires to rigid structures such as transistors[63]. These flexible wires can be extended to allow decoupling of the rigid constructs from the rest of the layout to allow the compactor to treat the mask spacing problem in a uniform manner[63].

Mask spacing is determined during two passes, vertical and horizontal, across the cell. The compactor then determines the minimum allowable spacing between grid lines by reference to the MTF. After the spacing for a grid line has been established, all of the objects on the line are positioned together[63].

The hierarchical compaction procedure compresses the hierarchy into a set of cells that completely *tiles* the circuit[63]. The compactor analyses all of the positional occurrences of each of the leaf cells, and finally generates a compaction solution for each distinct leaf cell that will satisfy all instances of that leaf cell. The compactor then performs the actual leaf cell compaction and obtains a mask cell that can be placed in any of the original environments without causing design rule errors. Once each leaf cell has been compacted, the mask layouts of the cells are assembled according to the original layout.

During the assembly phase of the compaction, the primary problem is that of cell ***pitch-matching***. Pitch-matching is a term used to imply alignment of ports on cells that must be connected at the composition stage of the design process. The pitch matching on the virtual grid layout of a design may be offset when the physical layout is established due to the leaf cell compaction process. The compactor therefore retains information on the original virtual grid layout, enabling it to stretch cells by the appropriate amount to achieve a match. Compression of cells will not occur as the leaf cells will have been compressed to minimum size in the first place[63]. The result of the compaction process is a mask description of the circuit in LLAMA. Fig. 3.4 shows a leaf cell before compaction in its symbolic form and after compaction in its mask form.



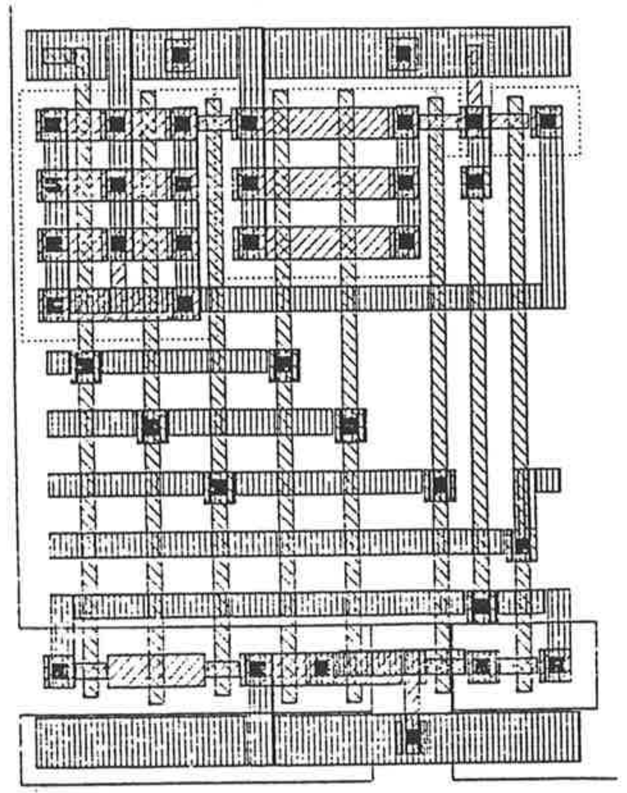


Figure 3.4: The VIVID system allows design in symbolic form, free from the concerns of Design Rules and once it is complete, VIVID compacts the symbolic design to a mask level layout. The plot above shows the symbolic representation of a cell, while the plot below shows the compacted mask layout.

Both HCOMPACT and the MULGA system compactor[65] use a symbolic design system on a virtual grid and both use the same two basic steps of mask rectangle generation and grid space assignment. The primary difference between the two is the order in which these two steps are done. Mulga does grid space assignment first while HCOMPACT does the mask rectangle generation first[65].

3.1.5 **Mask Level Layout Description**

The generation of a mask description of the design marks the end of the symbolic work in the VIVID system and the beginning of the physical manipulation of mask geometries. It is intended to provide an efficient Geometry Editor with the VIVID system but as yet this does not exist and it is necessary to use current tools that are available. In this case KIC (see section 2.3 of this thesis) is used to provide the geometry editing functions required.

The VIVID system may then be currently regarded as a front end processor to tool sets such as those described in Chapter II of this thesis. Consequently, ***A Translator of Layout Languages*** **ATOLL** is provided within the VIVID system to translate from the LLAMA mask description language to other languages used as standard interfaces between particular low level tool sets. Examples of standard formats that ATOLL already caters for are CIF, CAESAR, and Calma GDS II Stream Format. The architecture of ATOLL has been structured to facilitate the addition of other mask description languages[63].

3.1.6 **Master Technology File System**

One of the concepts discussed in the low level tool set was that of technology independence and in line with this concept, the tool ROWAN, based on LYRA, was discussed as the first of a series of tools to use a technology file as the basis for this technology independence. Technology independence is a favoured feature of any tool or tool system as it enables swift generation of a tool or tool set to handle any new technology based on the manipulation of mask layers that conform to a set of design rules.

The VIVID ***Master Technology File*** **MTF** System is the part of the VIVID system which provides the interface between the designer and the design system. The VIVID system configures itself to a particular technology by referring to the MTF system.

This system contains all of VIVID knowledge of circuit primitives and controls the representation of primitives in the ABCD language by defining the names of device types and process layers. It controls the appearance of the circuit by defining the symbol shapes, colours and stipple patterns and it controls the formation of mask layout forms of the circuit primitives by providing symbol to mask translation rules. Finally, the MTF system contains information about the primitives electrical properties.

It is not a trivial task to create a new Master Technology File, but altering a current one to reflect technology changes is not a difficult task. Chapter V gives a design example using a new Master Technology File as created for the design rules for a CMOS process described by Weste and Eshraghian[48]. Part of this new MTF is shown in Fig. 3.5.

3.1.7 VIVID Summary

The VIVID system provides a set of tools for symbolic virtual grid layout of custom VLSI circuits. These tools enable high level circuit design, which allows the designer to concentrate on circuit topology rather than on mask level layout and manipulation. The system is currently available to run under the UNIX operating system but will soon be available, due to research primarily done at Adelaide University, and by development work by ISD, to run under the VMS operating system.

Tools allowing mask level manipulation are required at the back end of the system for final mask level simulations and any resulting mask level alterations, or for such tasks as the wiring of pads to the final layout. They are not yet supplied as a part of the VIVID system. However it is intended that they ultimately will be.

```
compaction{
        layer_spacing{
                alum,    alum,    15 * MICRON / 2;
                poly,    poly,    10 * MICRON / 2;
                thinox,  thinox,  15 * MICRON / 2;
                pthinox,pthinox,15 * MICRON / 2;
                cut,     cut,     10 * MICRON / 2;
                poly,    thinox,   5 * MICRON / 2;
                poly,    pthinox,  5 * MICRON / 2;
                poly,    cut,     10 * MICRON / 2;
                p_plus,  p_plus,  10 * MICRON / 2;
                thinox,  p_plus,  10 * MICRON / 2;
                p_well,  p_plus,  25 * MICRON / 2;
        }
        antifeature_width{
                cut 10 * MICRON / 2;
                alum 15 * MICRON / 2;
                poly 10 * MICRON / 2;
                p_plus 10 * MICRON / 2;
                p_well 15 * MICRON / 2;
                thinox 15 * MICRON / 2;
                pthinox 15 * MICRON / 2;
        }
}
```

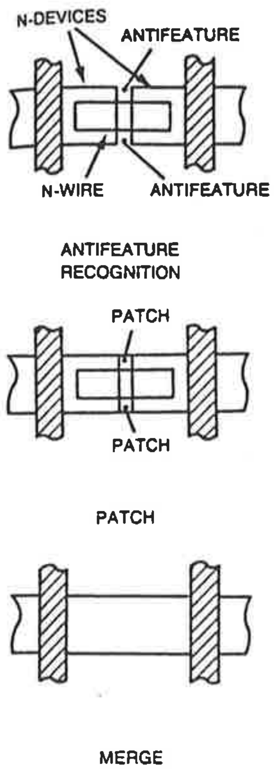


Figure 3.5: A component of the Master Technology File used in the VIVID system for the description of the spacing rules used for the Weste and Eshraghian[48] CMOS process. A new concept introduced by the higher level tools is the ANTIFEATURE width. As mask geometries are automatically generated by the compactor, small antifeatures can result as shown at top right. While it does not affect circuit performance, it is important because small photoresist features are likely to peel off during processing, causing problems elsewhere on the mask[48]. As a result, small patches are added to fill the antifeature as shown at centre and bottom right.

The combination of the best of the tools from Chapter II, as described in Table 2.5, with the VIVID system provides a complete set of tools enabling the most efficient design of full custom IC's on a very large scale using state of the art techniques. The diagram in Fig. 3.6, taken from the VIVID System Overview Manual, describes the complete VIVID system. When the proposed developments are completed, the resultant system will provide a complete stand alone software package for IC design. This package will then provide a sound base on which to build potential Silicon Compilers of the future.

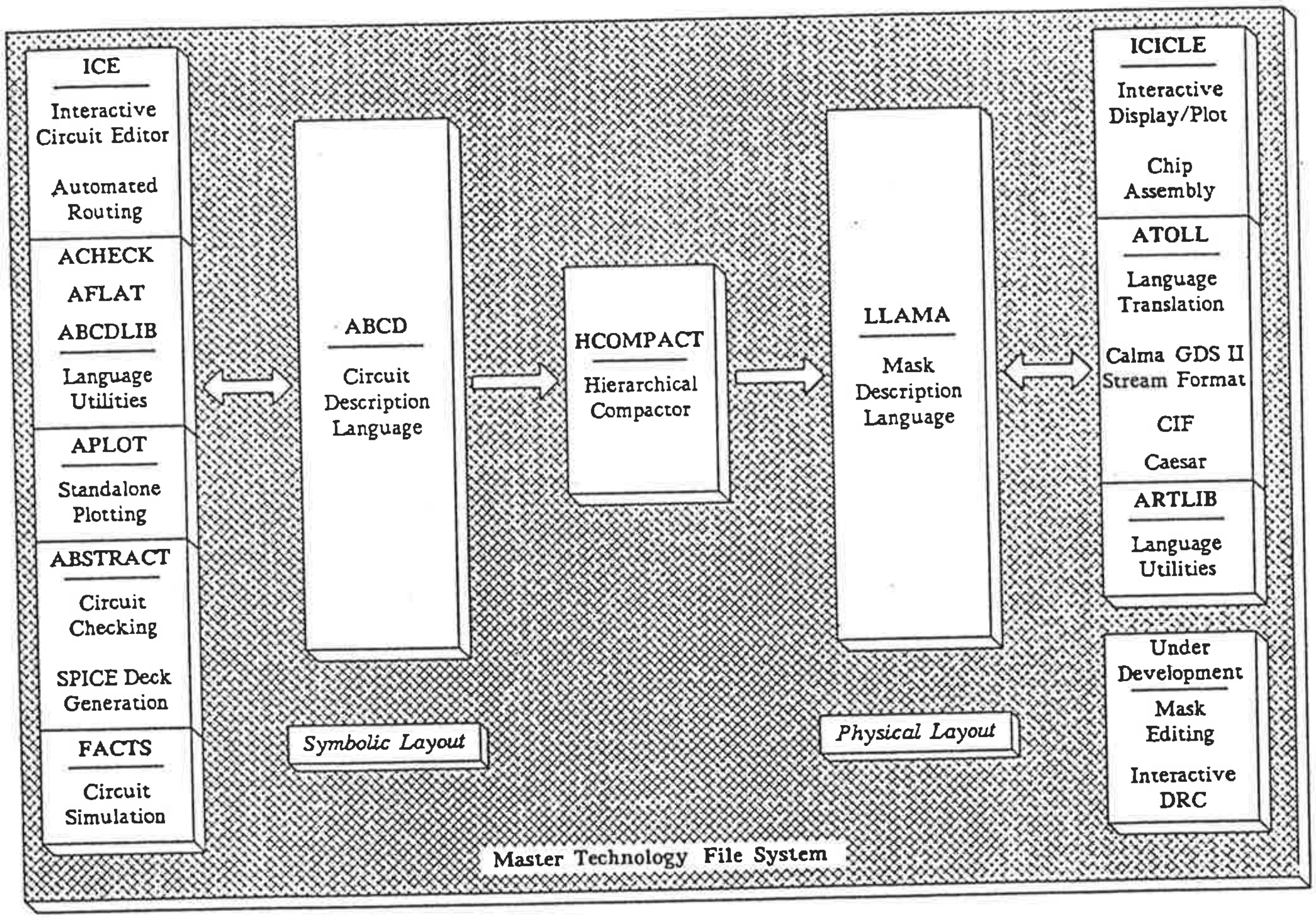


Figure 3.6: The VIVID System design environment indicating the separate mask level and symbolic level components. Chip assembly currently requires the use of low level tools such as those described in Chapter II.

Integrated Circuit Design Levels			
Design Level	*Components*	*Information Units*	*Time Units*
Processor	CPU's, IOP's,	Blocks of Words	10^{-3}-10^{3} s
	memories, IO devices		
Register	Registers, combinational	Words	10^{-9}-10^{-6} s
	circuits, simple		
	sequential circuits		
Gate	Logic gates, flip-flops	Bits	10^{-10}-10^{-8} s
Mask Layout	Mask Geometry	Analogue Signals	$< 10^{-10}$ s

Table 3.2: Integrated Circuit Design Levels. These levels provide the boundaries that enable differentiation between the levels of VLSI Design Tool Sets. Chapter I described a tool set at the lowest, Mask Layout, level while this Chapter has introduced the VIVID System, a Gate level tool set.

3.2 Design Levels

The design of a complex system such as any VLSI system, can be considered at different levels of complexity depending on the components recognised as the primitives. In digital computer design there are commonly three major levels considered. These are the *Processor Level*, the *Register* or *Register Transfer Level* and the *Gate Level*. (With IC design a fourth level, that of mask layout has been identified and described). Table 3.2 lists the types of primitives or components recognised at each level. The boundaries between the levels are never absolutely defined and so it is possible to encounter system descriptions using primitives from more than one level.

The gate level is the subject of classical switching theory and is the concern of the logic designer. Its level of complexity can be considered equivalent to that involved in the design of SSI logic. The register transfer level is approximately the level seen by an assembly language programmer and has a complexity comparable to MSI circuit design. The processor level is that seen by a computer architect and can be equated to the system design problem faced by an LSI or VLSI designer.

Other methods of distinguishing the various levels are considered in Table 3.2. Such methods may consider the time taken to process the information used at the various levels, starting commonly at nanoseconds at the gate level and moving on to seconds or minutes at the processor level. The amount of information processed at the various levels is also a useful measure, with bit information being processed at the gate level and blocks of words at the processor level.

These design levels are classified as high or low depending upon the complexity of the primitive used by the particular level being considered. This Chapter has considered a tool set, VIVID, which can be used at the lowest level discussed here. Following sections will consider more advanced, higher level tools, not yet composed into a system, but none the less higher level than those tools introduced so far.

The principles behind the development of such tools revolve around the fact that complex systems tend to have hierarchical organisation[67]. The development of complex systems is made easier by and arguably requires the existence of stable intermediate steps. Hierarchical organization also has important implications for VLSI designers. It is perhaps most natural to proceed from higher to lower levels of design as this corresponds to a progression to successively greater levels of detail. This ***Top Down*** or ***Structured*** design approach has been promoted most strongly first by the proponents of structured programming[67] and then adopted by the proponents of VLSI design[8].

Design problems at the various levels are quite different. At the gate level, substantial ***Switching Theory*** exists as a basis for the solution of problems encountered. Register and Processor level design however still remains largely an art and is highly dependent on the designer's skill and experience.

As a result, most of the developed tools currently in existence for IC design are based at the gate level or at the mask layout level. The following section of this Chapter discusses a higher level tool for the design of IC's.

3.3 Programmable Logic Arrays, PLA's

The *Programmable Logic Array* (**PLA**) is described in this chapter as a higher level tool because of the method used to input information to the tool. While it cannot be considered the most efficient way to implement a logical function in terms of area or speed, it can still be considered a full custom design tool because it requires the manipulation of all mask levels to provide the desired result. The PLA is used to provide fast layout solutions for irregular *Combinational Circuit* problems.

Combinational Functions can be defined by a truth table that specifies for every input combination the corresponding output value. Combinational Circuits are the physical realisation of Combinational Functions and are constructed from primitives called gates which are themselves the physical realisation of simple Combinational Functions.

The algebra which models combinational circuits is a type of *Boolean Algebra* which originated with the work of George Boole (1815-1864), a contemporary of Babbage[67].

The first task is to describe a PLA. Fig. 3.7 shows the general form of a PLA. Using a 2 phase clocking scheme information can be shifted directly, or in complement form, into a structure containing an AND plane and an OR plane. The AND plane generates combinations of these inputs (*literals*) to form a number of boolean *product* terms each containing a number of these literals. These product terms are defined as *minterms*, as long as they contain all of the literals. The outputs from the AND plane are then fed into the OR plane where the boolean *sum* of the products is completed. This information is then stored in output registers and shifted out on the second phase of the clocking. An example of a canonical sum-of-products form of a function is

$$f = \overline{x}_1.\overline{x}_2.\overline{x}_3 + x_1.\overline{x}_2.x_3 + x_1.x_2.\overline{x}_3 + x_1.x_2.x_3$$

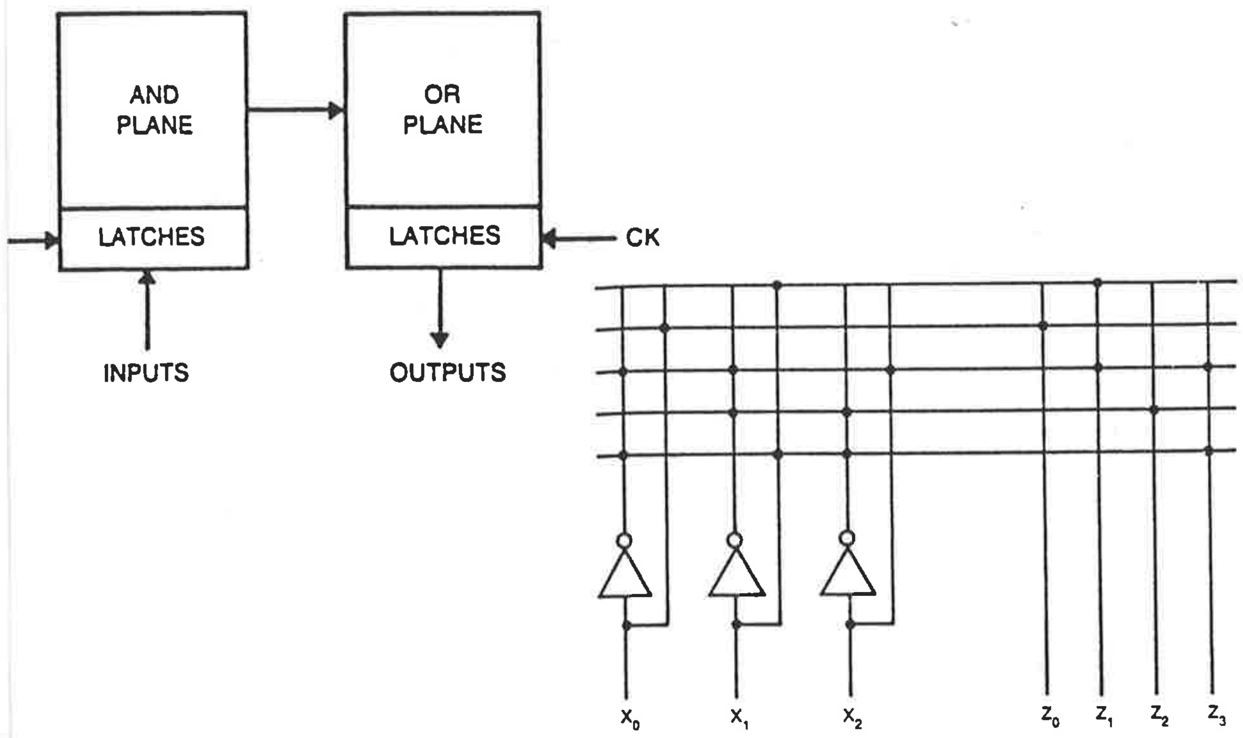


Figure 3.7: On the left is seen the general form of a PLA indicating the AND-OR planes, clocking, and inputs and outputs. The diagram on the right shows a possible circuit schematic for the equations referred to in the text below.

A minimal sum-of-products expression for the previous equation would be

$$f = x_1.x_2 + x_1.x_3 + \overline{x}_1.\overline{x}_2.\overline{x}_3$$

This equation corresponds to a two level AND-OR circuit with the fewest gates and the fewest gate inputs. These equations use three inputs and one output. A PLA can be designed to provide solutions for an arbitrary number of inputs or outputs. A simple example of a PLA scheme is shown in Fig. 3.7 for the equations

$$Z_0 = X_0$$

$$Z_1 = X_1 + \overline{X}_0.\overline{X}_1.X_2$$

$$Z_2 = \overline{X}_1.\overline{X}_2$$

$$Z_3 = \overline{X}_0.\overline{X}_1.\overline{X}_2 + \overline{X}_0.X_1.\overline{X}_2$$

PLA's can implement very irregular combinational functions but still maintain a regular structure as the irregularities are mapped only in the positioning of pull-up or pull-down transistors in the PLA. The size of the PLA and its shape is a function of four parameters. (1) The number of inputs, (2) The number of outputs, (3) The number of product terms and (4) the minimum feature size of the particular technology being used.

A PLA generator removes the designer from the physical circuit by allowing interactive input at the boolean equation level. This effectively allows the designer to skip further steps in his normal design translation procedure. The gate level digital system designer works in terms of boolean equations which describe system operation. The equations are minimised and subsequently translated into gate level descriptions of the circuit. If the designer is working at the symbolic (VIVID) level, these gates are then converted to device level descriptions and the circuit can be implemented using ICE or ABCD. If the designer is working at the lowest level, the device level descriptions must in turn be converted to mask level descriptions and implemented with a layout language or geometry editor. At each translation stage errors are likely to be made and as the lower levels are reached, the complexity increases greatly, increasing the likelihood of error.

A *PLA Generating* (**PLAGEN**) Tool automatically generates mask layout in an interactive computing session. PLAGEN, the tool discussed in this thesis is written in PASCAL to run under VMS. Fig. 3.8 details an interactive session in which the PLA layout shown in the same diagram was generated for the following boolean equation:-

$$O = \overline{M}.P + P.S.I + P.\overline{S}.\overline{I}$$

The input to PLAGEN comes not only interactively from the designer, but also from a file. This file contains a ***Karnaugh Map*** style of input as well as information regarding the number of inputs, outputs, and the number of product terms involved in

```
$@plagenn pla
Enter number at which symbols start:9000
Subsymbols to be generated?y
Do you want to program this PLA?y
Do you want all inputs on the same side?n
Which inputs do you want at the top?1 3
Do you want the inputs of this PLA to be clocked?n
Do you want the outputs of this PLA to be clocked?n
Do you want outputs on the bottom side?n
Do you want input and output labels omitted?y

PLA completed.
$
```

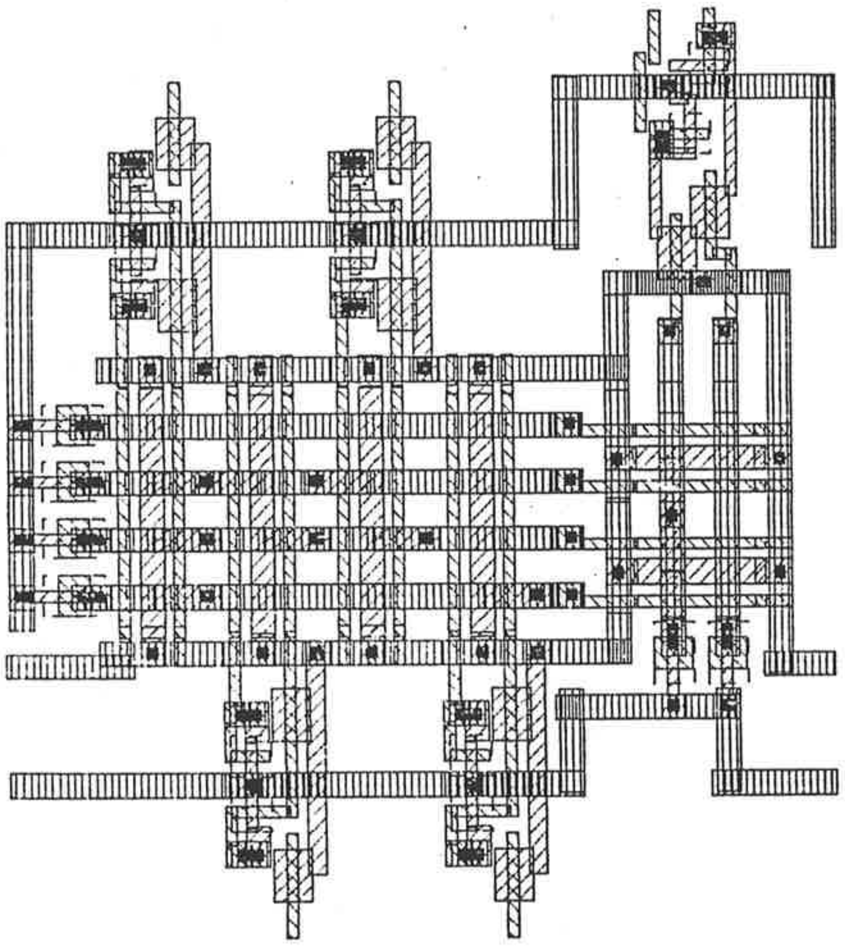


Figure 3.8: The text above shows the interactive session between the designer and the tool PLAGEN used to create the comparator PLA for Project C4, AUSMPC May 1982. The layout for the PLA is shown at the bottom of this diagram.

the boolean equation. A Karnaugh Map provides a method for minimising the number of primitives (gates) used in a particular combinational circuit design and so provides an attractive format for the input of the Equation as it should prompt a designer to minimise his equation prior to implementation. Fig. 3.9 details the input file for the PLAGEN session described in Fig. 3.8.

$$O = M.P + P.S.I + P.\overline{S}.\overline{I}$$

```
4 3 1
1xx1 1
111x 1
100x 1
```

	SM			
PI	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	1
10	1	1	1	0

Figure 3.9: The contents of the Input File to the tool PLAGEN are placed in the form of a karnaugh map representation. Here the input file on the left is compared to a karnaugh map representation of the boolean equation, on the right.

The first line of the PLAGEN input file has three numbers in it. The first of the three indicates the number of different inputs to the PLA, or the boolean equation. The second of the three inputs indicates the number of product terms involved in the boolean equation. The last of the three numbers indicates the number of outputs required on the PLA. The number of lines below this first line corresponds to the number of product terms, while the number of entries on the line itself corresponds to the number of inputs (literals) used in each of the terms as well as the outputs. A space separates the inputs from the outputs. Once this file has been created, then PLAGEN can be executed.

Another example of a PLAGEN input file, this time for a boolean equation which is a non minimised form of the previous equation is used to indicate the possibiltiy of redundant circuitry. The circuit resulting from the boolean equation below is shown in Fig. 3.10.

$$O = \overline{M}.P + \overline{M}.P.S.I + \overline{M}.P.\overline{S}.\overline{I}$$

The PLA offers great reduction in design time and a much more friendly designer

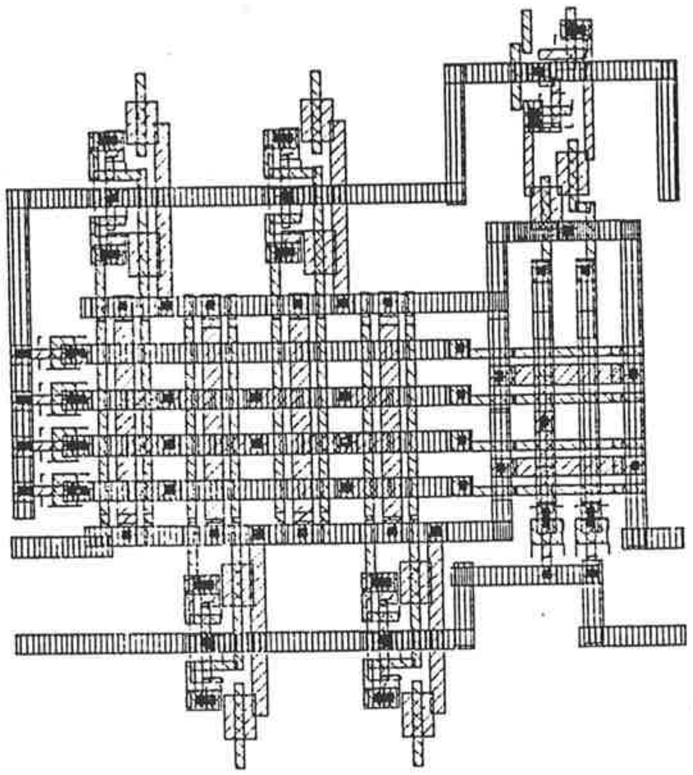


Figure 3.10: The PLA layout above provides the same circuit functionality as that shown in Fig. 3.8, but the one shown in Fig. 3.8 has been minimised. The effect this minimisation process has in this case does not alter PLA size, however for larger equations such minimisation could effectively provide significant space savings for an IC design.

interface than most other design tools. The designer interface to this tool, using design techniques based on switching theory, distinguishes it from other gate level layout tools (VIVID) and places it on a higher level. Of all the tools discussed in this thesis, the PLA can easily be considered the highest level design tool, and examples of circuits designed using this tool will be given in Chapter IV.

Unfortunately, as yet, no Gate Level tool exists that totally removes the need for the designer to be involved in the lowest level of design. The VIVID system still requires manipulation of mask geometries by the designer for final connection of pads to the design IO ports. The designer must also manipulate mask geometries to connect to PLA generated layouts. As soon as any Mask level manipulation is required, most of the low level tool set is required to verify the additions to the design.

3.4 Display Tools

When considering the complexity of IC design and all of the functions desirable in a tool set that can handle VLSI systems, it is easy to overlook the most basic function that is a requirement of all tool sets. Some efficient method must be used to display mask and/or symbolic level information and the results of verification tests done on the designs on a variety of different peripheral devices, ranging from simple text terminals at the lowest level to high resolution colour graphics devices at the top level of available terminals. Both interactive displays and hard copy information must be provided for.

In well organised tool sets, the standard interface language can be used as an input source to a tool which will allow display of the design information on a wide variety of peripherals. An example of such a tool is SEE[66], the ISD generated tool designed to display nMOS and CMOS CIF geometries on a number of different display devices, including a variety of plotters, as well as a wide variety of VDU's, both colour and monochrome. SEE is written in PASCAL to run under the VMS operating system. As an example of this type of tool, Fig. 3.11 lists an interactive SEE session and shows the resultant diagram produced .

As well as displaying the complete design, SEE allows some manipulation of the displayed information. For example, on an interactive VDU, the designer may selectively zoom in or out on sections of the design. On all devices, windows of certain areas of the design may be selected for display and the designer may also select any combination of the mask levels to be displayed. Other features such as the display of node labels also exist.

The higher level tool set should also provide an integrated approach to the display of not only mask level information but also the symbolic level information and simulator output as well. The VIVID system uses high performance graphics subroutines, collectively described as the Z-Graphics package to provide output for its graphical tools including, FACTS, ICICLE, (a program designed as an interactive tool for viewing

```
$ see bitplt.cif
Enter a value of Lambda you want to use (Default is 250.00) ?
No errors detected in cif file - D1:[JNOONAN.SUB1]BITPLT.CIF;
Technology is  nMOS ,  Lambda = 250
The bounding box of the complete design is :-
   Lower left  (x,y)    ( -3   ,   3 ) Lambdas
   Upper right (x,y)    ( 122  , 383 ) Lambdas
The display scale to fit the vertical height is : 1.97
Enter the display scale in pixels/lambda (Default is 1.00) ?
Do you want to select a window (Default is No) ?
The window you have selected is :-
   Lower left  (x,y)    ( -3   ,   3 ) Lambdas
   Upper right (x,y)    ( 122  , 383 ) Lambdas
Enter layers to be displayed (Default is all layers) ?
Do you want to display nodelabels (Default is No) ?
```

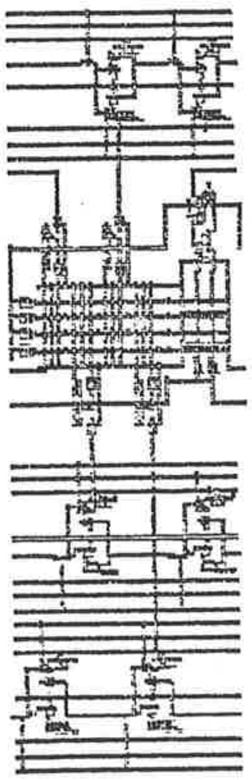


Figure 3.11: The text shown at the top of the page details an interactive session between a designer and the tool SEE. The result of the session can be seen below the text. SEE provides either hard copy or interactive displays of CIF layouts.

mask layouts using LLAMA mask descriptions), APLOT, (a stand alone ABCD design plotting program used for producing a quick graphical view or hard copy when interactive editing is not required), SIMPLOT (a program which plots output from FACTS or SPICE), and MTF (technology independence and information such as colours, stipple patterns and shapes for the other graphical tools, are all available for user manipulation using MTF). The Z-Graphics subroutine library is general purpose and device independent, allowing the future addition of new devices as and when necessary.

3.5 **A System Level Simulator (TicToc)**

Research by Dickinson[55] at Adelaide University has led to the development of a tool designed to provide assistance for the VLSI designer in the pre-layout phases of architectural specification and verification, and the development of microcode and test vectors.

If a designer can represent and simulate a circuit at a very high level of representation (abstraction), he can more quickly proceed through the top down design process. This form of verification is necessary as VLSI design is the process of converting a high level functional specification into a set of low level geometric mask patterns. Expert designers accomplish this complex translation problem by partitioning the design process into a number of levels of representation as shown in Fig. 3.12[57].

Partitioning of a design is a difficult task for a VLSI designer as there are none of the traditional bounds such as packaging constraints or physical size limitations enforcing partitions. The partitioning process for a VLSI designer must rather be considered in the light of parameters such as (1) Minimisation of interconnect, and (2) Maximisation of concurrency. System simulation with a tool such as TicToc allows the optimisation of these parameters by enabling various partitioning strategies to be tested at a high level.

TicToc enables the designer to specify and verify the architecture on a communication basis. The TicToc specification describes the hierarchy and connectivity of the system without being overly concerned with the internal workings of modules. Modules either exist as connections of sub-modules or as leaf modules.

TicToc was written in MODULA-2 and the TicToc specification language is actually an extension of this language. This is appropriate as the language is suited to the description of systems that are partitioned and communicate only over defined interfaces[55]. For example, the leaf modules of a TicToc specification are written in MODULA-2 and need not bear any relation to a physical realisation except in the function that relates module input and output[55].

3.6 Conclusions

Relating Digital Design concepts as discussed in sections 3.2 and 3.3, to IC design is necessary as it provides the framework around which efficient design methodologies may be developed at higher levels. Chapter II described the lowest level of design tool required for IC design, the lowest level considered in Table 3.2. This chapter has introduced the VIVID system, which can be classified as a gate level tool set for IC design, the second level considered in Table 3.2. The VIVID system is still being developed to provide all of the functions that this level of design requires.

Various attempts at even higher level design tools have been made. The Silicon Compiler concept has already been mentioned. Early examples of such tools are ***Bristle Blocks***[68], ***FIRST***[69], and ***MacPitts***[14.3][70]. Each of these tools is characterised by the notion of a fixed floor plan in which a designer has a limited amount of freedom to explore the design space[48]. They are very specialised tools and no contemporary silicon compiler has yet demonstrated any ability to deal with the complexity involved in full custom VLSI circuits[48].

The VIVID system has developed to the stage where it can be used to design CMOS and nMOS circuits at a symbolic level in conjunction with tools from the low level tool set. It provides a stable intermediate assembly level representation for VLSI design and allows development of higher level tools from this stable intermediate level.

An example of research on this subject can be seen in the CADRE system[57] where a collection of expert systems communicating through a central manager, (using the ZetaLisp environment on LISP machines), have been used to model the human physical design process. A virtual grid and symbolic system is used at a lower level of the CADRE system as can be seen in Fig. 3.12 which shows the CADRE view of the levels of design abstraction for a VLSI design.

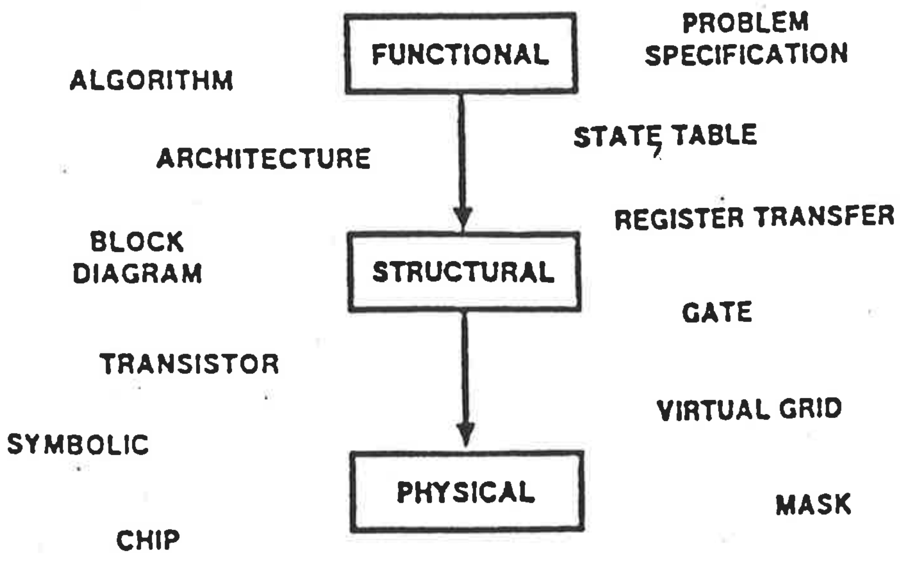


Figure 3.12: The CADRE view of Levels of VLSI Design Abstraction.

CHAPTER IV

nMOS DESIGN USING MASK LEVEL CAD TOOLS AND TWO MPC PROCESSES

4.1 Introduction

Australian MPC efforts commenced in 1982 as indicated in section 1.3 of Chapter I. These efforts enabled Australian circuit designers, who would normally be remote from IC fabrication facilities, to use for the first time a formally coordinated system for prototyping circuit designs on silicon. This Chapter presents an analysis of the CAD tools used for the designs and compares some testing information with some simulated results to verify the tool set functionality.

The first CSIRO AUSMPC, AUSMPC 5/82 (May 1982), provided free fabrication of circuits using an nMOS 5 micron process, (ie. $\lambda = 2.5\mu m$ or minimum feature size $= 5\mu m$). Free fabrication of prototypes was offered to the best 46 designs submitted to the CSIRO VLSI Program for its first fabrication run as a means of publicising the MPC effort and to attract engineering excellence to the work[17].

The first part of this Chapter describes a design by Noonan and Williams[16.1], which was sucessfully submitted to AUSMPC May 1982. The project was designated **Project C4 AUSMPC 5/82**, and was completed using some of the low level tools discussed in Chapter II (see Table 2.6) and one of the high level tools discussed in Chapter III. Project C4, was designed to be a ***Signature Analyser*** and was the designers' first attempt at designing any IC and specifically their first attempt at nMOS IC design following the Mead and Conway design methodology. This device is designed to function as a component part of Eshraghian's ***Passive Subharmonic Transponder***[82][83], **PST** project for use in a Vehicle Recognition System. Project C4 was designed as a 4-bit slice version of a system that could selectively compare bit information between two registers, using a third register as a mask register.

One of the Designers' aims was to get a working version of the device in silicon so as to reduce the amount of space taken up by hardware in the PST project, but primarily the aim was to use this effort as a familiarisation exercise to learn how to design in silicon following the design methodology of Mead and Conway and evaluate the CAD tools described in this thesis. The priorities were to attempt a modestly complex design and get it working. The designers were not trying to complete the most compact design, rather, to fulfill the project aims, as many tools as possible were used to try and provide a greater CAD Tool evaluation, hence the use of a PLA for the comparator part of this circuit. The layout of this design was completed using only the PASCAL Embedded Layout Language, BELLE and the PLA Generator PLAGEN.

The second part of this Chapter describes a design by Dunis and Webber[76] which was fabricated using the MPC process (again a 5 micron nMOS process) coordinated by the JMRC. The layout of this design was completed using the Geometry Editor KIC.

The tools used for this design complete an analysis of the Low Level Tools as described in Chapter II. The results of this work can be used to gauge the effectiveness of full custom design, using a simple CAD Tool set, with two different MPC processes.

4.2 The Signature Analyser

The Signature Analyser consists of four functional blocks. They are (1) the Input Shift Register; (2) the Comparator, (3) the Signature Shift Register, and (4) the Mask Shift Register. A block diagram of this device can be seen in Fig. 4.1.

The Signature to be analysed consists of a 64 bit code which is designed to contain a number of subsets of information regarding a vehicle carrying the code. For instance, the 64 bit code could contain information regarding the registration number, the engine number, the make and model of the vehicle, the class of vehicle, eg. car,

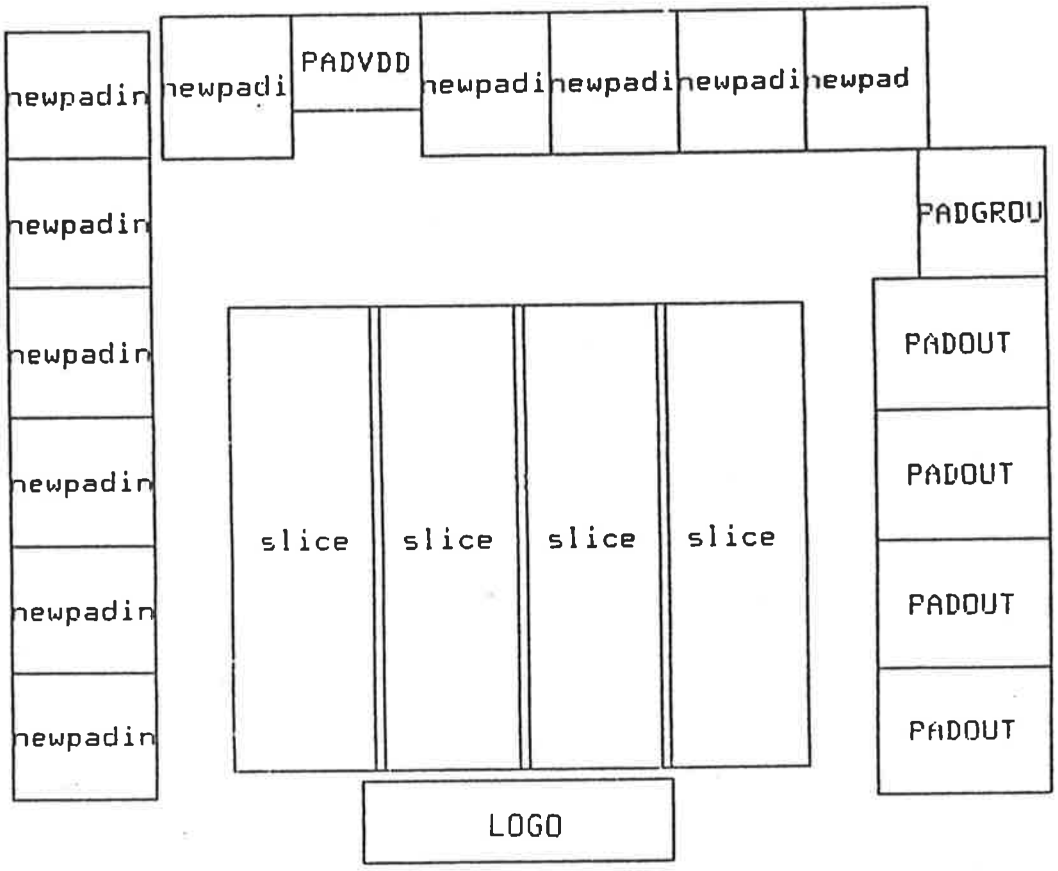


Figure 4.1: The floorplan of the Signature Analyser design submitted for fabrication as Project C4 on AUSMPC 5/82. This floorplan can effectively be used as a block diagram for the design.

utility, truck, bus, van, ...etc., the official classification of the vehicle, eg. police vehicle, ambulance, fire engine, private vehicle ...etc. This information can then be used for a variety of purposes. For instance the information could be used for detection of an individual vehicle or, for that matter, an individual class of vehicles for traffic control purposes. A possible use could be for traffic control to let an ambulance pass unhindered to an accident or a fire engine to a fire.

The basis of operation of a complete Signature Analyser is as follows. An input code originating from a vehicle consisting of 64 bits is serially shifted into the input register where the information is stored. At some time previous to this, a signature code would be serially shifted into the signature shift register where it is also stored. Another 64 bit code is also serially shifted into the mask shift register and stored prior to loading the input information. The purpose of the device is then to compare all of the bits in the input and signature shift registers that are not masked out by the mask shift register, and to indicate a positive comparison.

Even though the Signature Analyser chip has been designed with the strict intent of providing an IC implementation for a part of the Vehicle Recognition System, it is easily conceivable that this device could be used for a variety of other purposes with minor alterations. For instance, Signature Analysis is a well recognised technique used in testability procedures[81] and with the addition of feedback in the input shift register and some other design modifications as detailed in Liebelt's thesis[81], this design could provide useful background work for a test structure.

4.2.1 Problem Definition and Floorplanning

Using the Mead and Conway Design Methodology, the design proceeds in certain steps which are foreign to the designer used to designing in TTL using SSI or discrete components. The methodology takes into account not only circuit efficiency, (eg. the best way of completing the circuit design in terms of chip area, or speed), but also considers design time efficiency (ie. the time taken to complete and test the design) which has rapidly become the weak link in the design chain as far as large systems are concerned. A description of the aims of the Mead and Conway Methodology is that the designer needs to consider both circuit efficiency and design time efficiency by creating a *Structured Hierarchical Design* using a reduced geometric and electrical rule set.

The first task encountered in the methodology is to write a description of the problem and list the number of of inputs and outputs required. It is very important to give careful consideration to these problems at this stage because if for instance, an input is overlooked, it can be very difficult, if not impossible to rectify the problem at a later stage in the design process. This first design stage could be called ***Problem Definition.***

The problem definition stage as applied to the Signature Analyser project often takes into account wider aims than just circuit optimisation. In this case part of the problem was to test a variety of CAD Tools as well as to complete a working nMOS prototype of a four bit slice of the desired circuit.

From the designer's understanding of the general sizes of the leaf and composition cells to be used in the design, in this case shift registers and PLA's, an estimate of the size the actual functional blocks would take is then required to be laid down on paper. As well, consideration must be given to the routing of data and communication lines between blocks, and also between the pads and the blocks. This second task involved in the Design Methodology is called creating a floorplan.

It must be stressed that this is only an initial estimate of the final floorplan and the accuracy of this first estimate depends on how familiar the designer is with the leaf and composition cells and how well the signal paths are routed. Most often VLSI designers will find that, to refine the floorplan to the point where the actual layout blocks fall inside those set out on the floorplan, is an iterative process and the number of iterations will be related to the complexity of the circuit and the designers experience. The number of iterations in this process reduces as the experience of the designer increases. A mask layout level floorplan confines a designer to absolute geometrical relationships and sizes and must be considered very carefully.

Care must also be used in positioning pads around the the perimeter of the circuit. If this is done well, much routing of signals can be avoided. This design experience showed that the routing of signals around the regular internal blocks can take a significant percentage of the design time (see section 4.3.2). Fig. 4.1 shows the Floorplan created for the Signature Analyser.

4.2.2 Stick Diagrams in Mixed Notation

When designing on a fixed grid as is required with the low level tools described in Chapter II, an accurate picture of the cell at mask level is required on paper before layout can commence. This minimises the number of editing iterations required to accurately digitise the mask layout of the circuit design. The quantity of information required varies depending on whether a Layout Language or Geometry Editor is used by the designer in the layout process. If a Geometry Editor is used, Stick Diagrams are usually all that is required as hard copy preparation for the design. Using a Layout Language normally requires a detailed accurate layout of the complete circuit as well as the Stick Diagrams which aid in the complete layout.

Stick diagrams are a low level IC design aid that allow simple representation of designs at the basic Leaf Cell level rather than at the Composition Cell level of the hierarchy. The use of stick diagrams at Leaf Cell level could be equated to the use of a Floorplan at the highest level or lower Composition Cell levels of the hierarchy.

A stick diagram gives the designer a way of quickly representing his design so that he can distinguish the general form of his layout yet not take into account the basic spacing design rules. Mead and Conway used colours to represent the different layers in silicon. The representation is as follows:- ***RED=Polysilicon, BLUE=Metal, GREEN=Diffusion, BLACK=Cuts, YELLOW=Implant*** and ***BROWN=Overglassing***, and is the convention used in this Chapter when colours are used. With these coventions, stick diagrams representing all types of VLSI devices may be drawn simply and quickly. Fig. 4.2 shows the layout of the Shift Register Cell used in the Signature Analyser Project and its associated Stick and Logic Diagrams.

At the floorplan stage of the design it is necessary to think about signal routing. At the following stage it is of benefit to the designer to use mixed notation within the floorplan to represent the different cells and enable a more accurate estimate of the size required for the total design area as well as to formalise on paper the exact paths

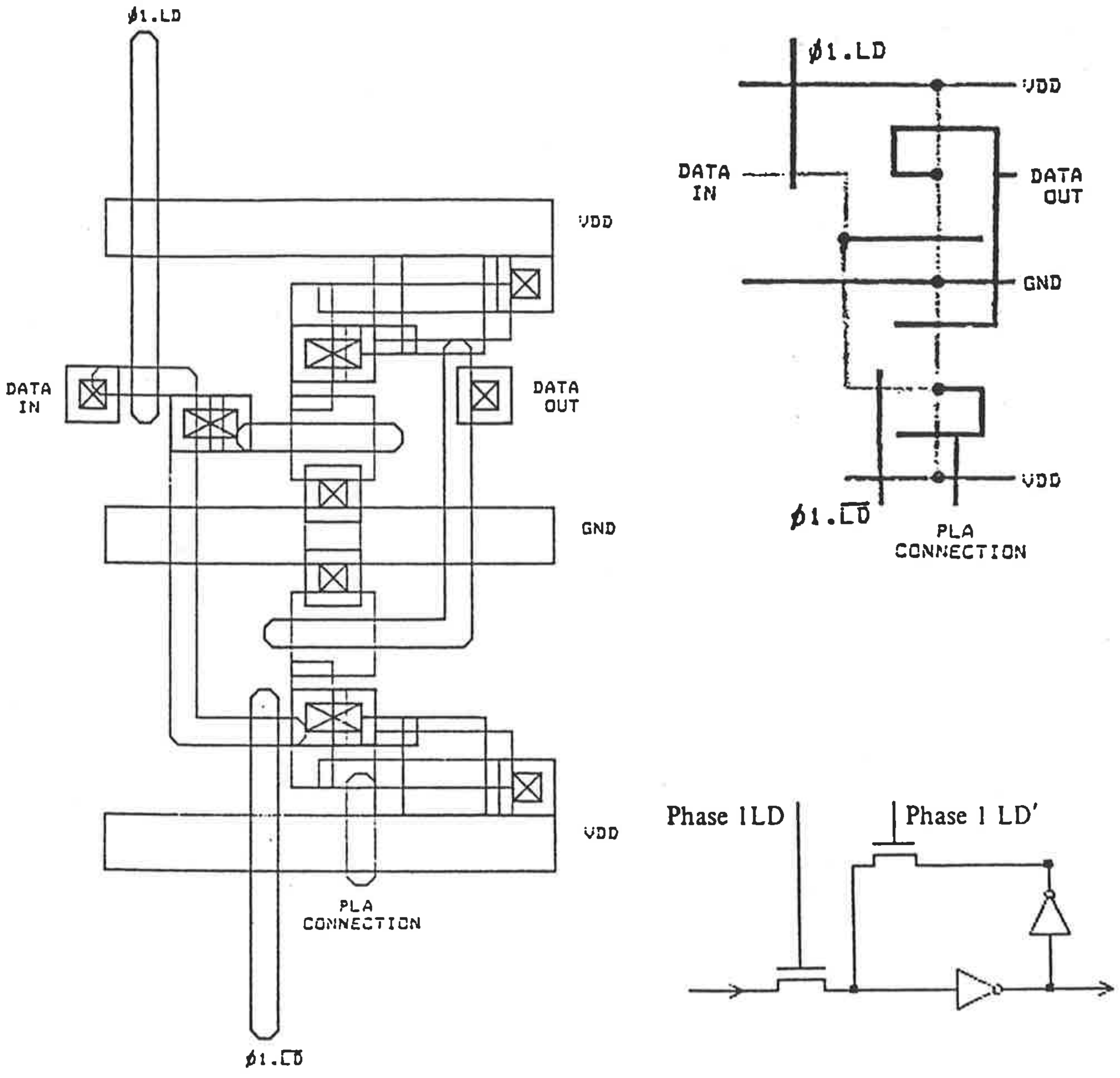


Figure 4.2: The layout of the Shift Register leaf cell used in the Signature Analyser design accompanied by its associated logic and stick diagrams. The node labels clearly identify the inputs, outputs and clocking lines on the layout.

for signal and data flow. The aim of mixed notation is to use composition cells and stick or leaf cell representation combined in such a way that they represent exact path architecture such that all that is required after this stage is the realisation of the actual layout using the full set of design rules.

A very important consideration is that of timing. Race conditions and glitches must be avoided. These problems occur if careful consideration is not given to the layers on which the signals are routed, or if signals are not generated from drivers ensuring appropriate rise and fall times of signals over long lines. For the Signature Analyser design, no timing restrictions were set, as the main design criterion was to achieve a fully functional circuit that would verify the design tools used and would allow future characterisation of the MPC process used for fabrication. Hence the normal concerns with race conditions are not crucial for this design.

4.3 Design, Layout and Checking

At this stage of the methodology the designer has almost exact estimates of the project size, knows where the signal and data paths lie between the functional blocks, and knows how the leaf cells will interconnect with each other. The designer is then able to commence work on actual mask layout of the individual leaf cells.

This work, when completed will define exactly the size of the project. For some MPC runs, (eg. all CSIRO MPC runs), a design must fall inside a final floorplan that has been specified early in the design process because once the final bid for space has been made on a multi project chip, the designer is limited by these bounds with little or no recourse for change if the design is to be accepted on that particular MPC run. The reason for this was that the competition for space on CSIRO MPC runs dictated the need for precise floorplans to enable multi-user participation and adherence to strict deadlines. There was seldom little, if any, room for floorplan expansion on such runs.

4.3.1 **Leaf Cell Layout**

Only two leaf cells are used in the Signature Analyser design, the Shift Register shown in Fig. 4.2 and the Comparator shown in Fig. 4.3. Belle has been refered to in Chapter II. The layout of the Signature Analyser Shift Register Leaf Cell, shown in Fig. 4.2, was completed using the BELLE file description shown in Fig. 2.1 (a).

An auxiliary program to BELLE called **GETSYMBOL** is intended for use in conjunction with the IMPORTSYMBOLS function within BELLE. It is used to generate a header file of the type required by IMPORTSYMBOLS. GETSYMBOL operates on a CIF file and produces a file containing the name of all symbols taken from a library of symbols used by the designer, their symbol numbers and their bounding box.

The Shift Register cell is a selectively loadable/resettable dynamic shift register with an 8:1 pull-up to pull-down ratio capable of driving pass transistors for cascaded stages, as would be required for the Signature Analyser. The Mead and Conway inverter[8] was modified to conform with standard design rules and to cater for space allocation. It can be seen that modifications could be made to the inverter to make it more area efficient still, however these modifications could not be implemented due to the deadline restrictions enforced by the CSIRO. Fig. 4.2 details the layout of the shift register with node labels and compares it to its stick diagram and its logic circuit diagram.

The basic cell size is 33λ by 48λ although the cell was extended to cater for standard connection to metal signal paths. This means that the bounding box for the cell is 33λ by 70λ but the cell does not take up the full bounding box in silicon real estate as it overlaps the signal feed paths.

The Comparator used in the Signature Analyser project was layed out using the tool PLAGEN. This tool has been refered to in Chapter III and is used to generate CIF code from a truth table description of a circuit. PLAGEN reads from an Input File and creates the PLA mask layout in a resultant CIF file. The CIF output of PLAGEN has to be concatenated with the CIF output describing the other design components to ultimately provide the final CIF code representing the complete design. Fig. 3.9 shows the input file to PLAGEN and the Boolean Equation used to generate the Truth Table for the input file. The resultant layout of the comparator described by the CIF output from PLAGEN is shown in Fig. 4.3 along with a schematic diagram of the comparator function that the PLA is supposed to perform.

It was realised that as PLA's get larger they also get slower and as this PLA is relatively small no problematic delays were anticipated or encountered. However it was to be expected that the speed of operation of the PLA would be the limiting factor in the overall speed of operation of the Signature Analyser.

The size of the PLA is 118λ by 120λ which is large considering the logic function it performs. However it is non-standard in that the positioning of the input buffers was modified to allow the PLA to be used in a bit slice application in order to produce a highly regular design. Originally the particular PLAGEN program used could provide inputs on one side of the structure only. The PASCAL program was modified to allow inputs on either side of the PLA. By repositioning the input buffers, system routing complexity is markedly reduced as signals can be fed between functional blocks in a regular manner and also the positioning of other functional blocks attains a higher degree of flexibility. A disadvantage in repositioning the input buffers is that the overall PLA cell will be larger. This disadvantage is however balanced by the advantageous reduction in interconnection complexity.

The specific modifcations made to the PLAGEN program were as follows;

(a) To allow any input buffer to be located at exactly the same x position but reflecting the input buffer about the x axis and placing the buffer at the correct position to line up with the AND plane.

(b) Minor modifications were made to the input buffers to allow ease of interconnections. This simply entailed extension of the POLY input wire.

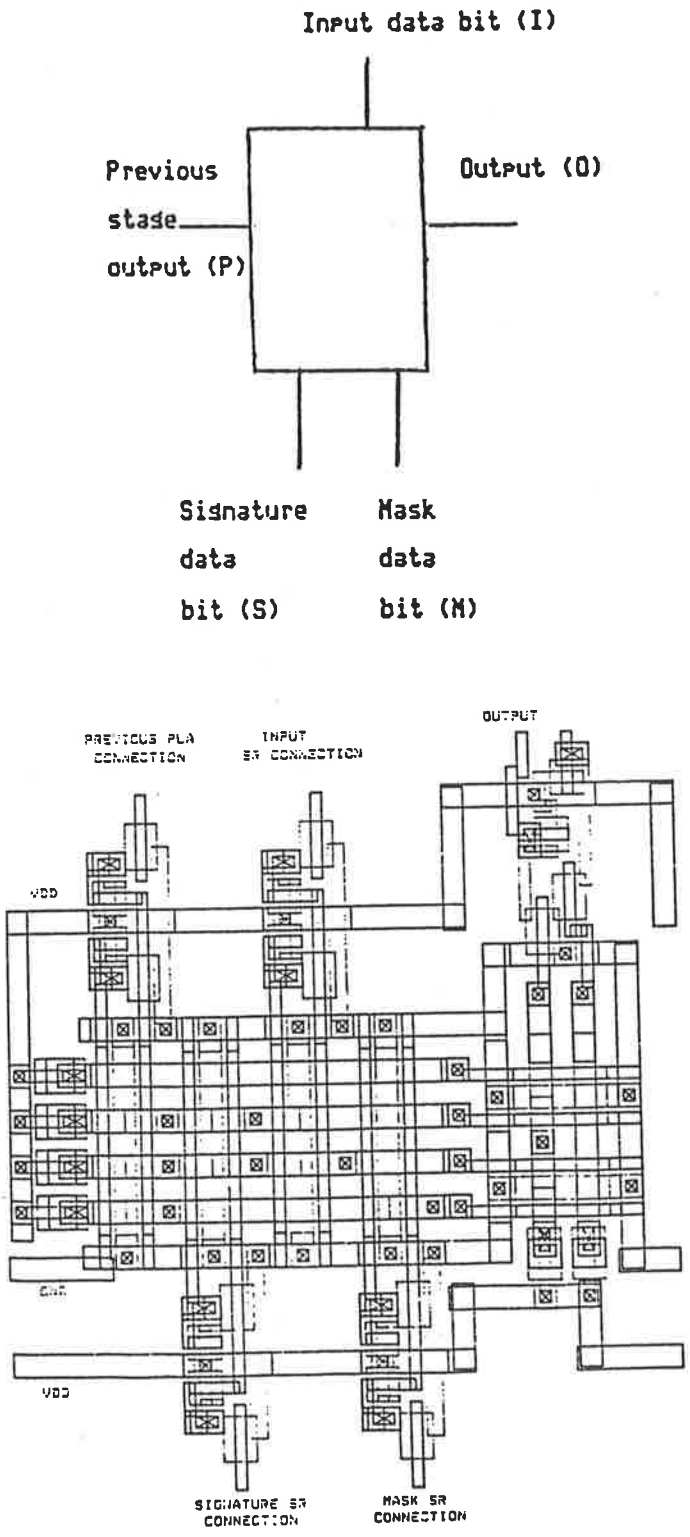


Figure 4.3: The layout and associated block diagram of the comparator used in the Signature Analyser Design. Note the extra silicon area required when using inputs on either side of the PLA structure.

(c) Minor modifications to PLAGround and the lengths of the AND and OR POLY and DIFFUSION wires made the paramaterisation of (a) much simpler.

(d) The rerouting of the GROUND and VDD wires was also necessary to cater for the abuttment of the PLA in this Bit Slice application.

Some standard cells were provided by the CSIRO to assist designers with their designs by providing standard leaf cells such as Input Pads and Output Pads in a library open to all designers so that these cells could be concatenated with the rest of the design software.

BELLE was used to layout the Shift Register, compose the leaf cells into the final design layout, and interconnect the leaf cells and the pads. PLAGEN was used to layout the Comparator. Appendix D details the BELLE program used, the PLAGEN input, and the resultant CIF required to completely describe the Signature Analyser design as shown in Fig. 1.2 (a). Because PLAGEN produces CIF as output, a means of concatenating the CIF in correct relation to the BELLE produced CIF, is required. This can be done using either a text editor or alternatively with the aid of the operating system command language.

The overall size of the project was 861λ in the x direction by 649λ in the y direction. λ for AUSMPC 5/82 was 2.5μm and so the dimensions of the complete Signature Analyser device was 2.1525mm by 1.6225mm.

4.3.2 Pad Allocation and Positioning

The pads for project C4 are specified very early in the design process as stated previously. The pads for most projects normally are evenly spaced on all sides of the circuit. For this device we have pads on three sides of the circuit and then the logo at the bottom. The total number of pads is 17 and each of the pad structures used was a standard cell provided by CSIRO. There are 11 input pads using **NEWPADIN**, 4 output pads using **PADOUT**, and then **PADVDD**, and **PADGND**. The pads are shown labeled in the plot in Fig. 4.4 which also indicates the floorplan.The description of these pads is as follows:-

PAD 1 = NEWPADIN. This pad has been labeled *PHI2SL* and is the clock line which acts as the shift through signal on phase 2 for the signature and mask shift registers.

PAD 2 = NEWPADIN. This pad has been labeled *PHI1SL* and is the phase 1 equivalent of PAD 1.

PAD 3 = NEWPADIN. This pad has been labeled *PHI2NSL* and is the clock line which acts as the refresh for the signature and mask shift register cells on phase 2 of the two phase clock for the signature and mask shift registers.

PAD 4 = NEWPADIN. This pad has been labeled *PHI1NSL* and is the phase 1 equivalent of PAD 3.

PAD 5 = NEWPADIN. This pad has been labeled *MSKIN* and is the data input to the mask shift register.

PAD 6 = NEWPADIN. This pad has been labeled *SIGIN* and is the data input to the signature shift register.

PAD 7 = NEWPADIN. This pad has been labeled *INPIN* and is the data input to the input shift register.

PAD 8 = PADVDD. This pad has been labeled *VDD* and is the supply voltage input.

PAD 9 = NEWPADIN. This pad has been labeled *PHI2NDL* and is the phase 2 clock signal which refreshes the input shift register cells.

PAD 10 = NEWPADIN. This pad has been labeled *PHI1NDL* and is the phase 1 equivalent of PAD 9.

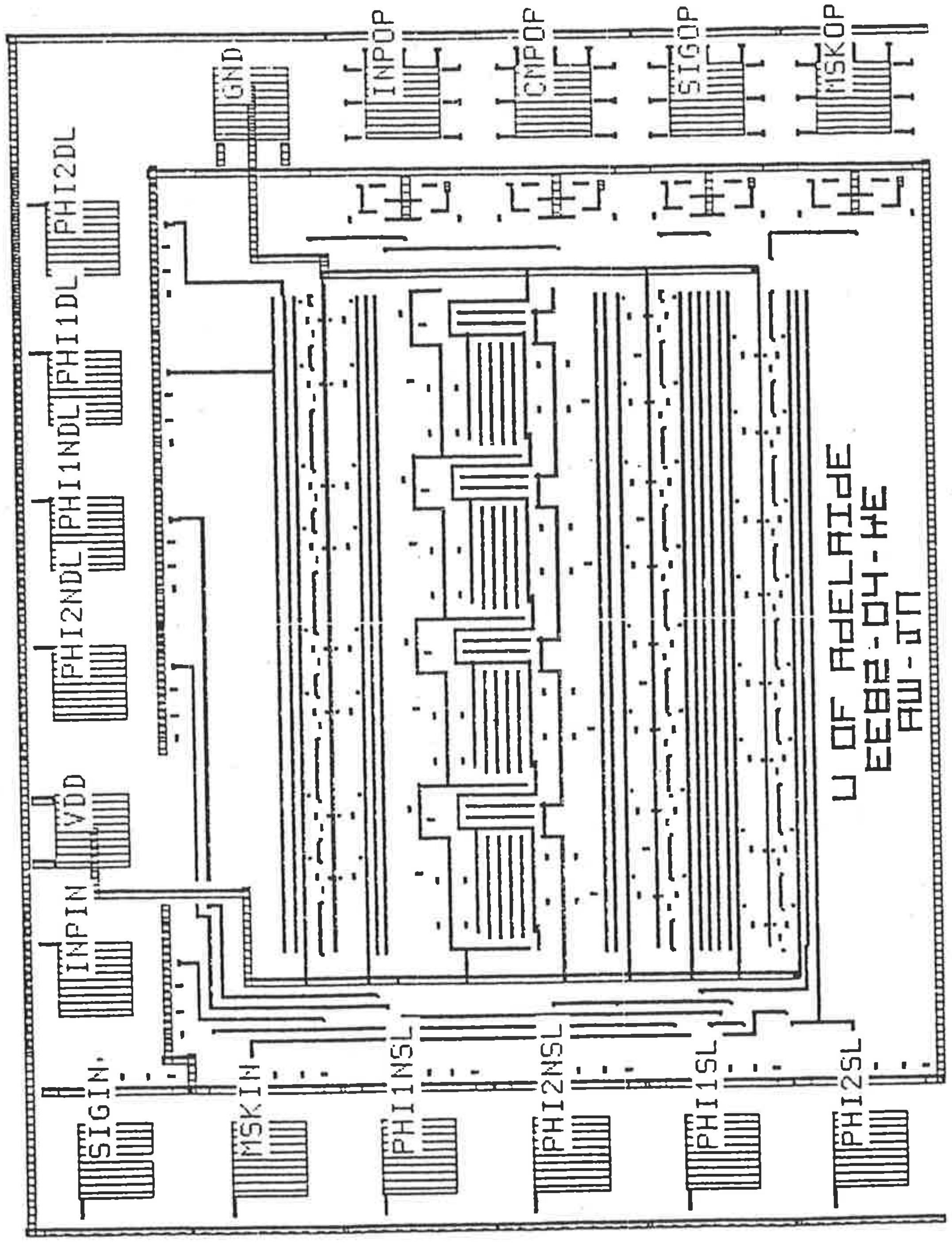


Figure 4.4: The plot of the Signature Analyser metal layer, Project C4 on AUSMPC 5/82 with the pads labeled. These labels coincide with those used in the text and the test information presented in following pages.

PAD 11 = NEWPADIN. This pad has been labeled *PHI1DL* and is the phase 1 equivalent of PAD 12.

PAD 12 = NEWPADIN. This pad has been labeled *PHI2DL* and is the phase 2 clock signal that shifts data through the input shift register.

PAD 13 = PADGND. This pad has been labeled *GND* and is circuit ground connection.

PAD 14 = PADOUT. This pad has been labeled *INPOP* and is the output of the input shift register.

PAD 15 = PADOUT. This pad has been labeled *CMPOP* and is the output from the comparator.

PAD 16 = PADOUT. This pad has been labeled *SIGOP* and is the output from the signature shift register.

PAD 17 = PADOUT. This pad has been labeled *MSKOP* and is the output from the mask shift register.

This pad labeling is used in the test result information presented in section 4.4.2 of this thesis. The labels appear alongside of the signals that are used as source signals for clocking and as input for the various shift register cells and alongside the signals received as responses from the shift register and comparator cells.

Pad Positioning is a very important design consideration as it is important to have the pads as close as possible to the circuit connection points so that the interconnection paths from pads are as short and as simple as possible. As mentioned previously most designers find that the interconnect problem can take up a significant percentage of the design time. The interconnect paths for this project were relatively simple and yet they took about 5% of the total design time.

4.3.3 Design Rule Checking

CHECK, the most efficient of the Design Rule Checking tools discussed in Chapter II, was used on the Signature Analyser CIF with different errors introduced to highlight the operation of a Design Rule Checking tool.

Design Rule Checking is necessary at different stages of a design. Rather than complete a full system with unchecked leaf cells, which would give multiple reports of the same errors should any exist in any of the cells, using a non-hierarchical DRC the designer would be wise to first check each leaf cell used in the design separately so as to validate that cell. The Signature Analyser has only two leaf cells, and one of these was generated using PLAGEN which produces CIF layouts according to the design rules. As a result the only leaf cell that needed checking was that produced using BELLE, the Shift Register, although the PLA structure may be checked to ensure its layout validity. The interactive DRC session and its results for the check completed on the Shift Register and the Comparator are shown in Fig. 4.5.

The composition of leaf cells to form the final layout is a source of many layout problems as is the routing of the inputs and outputs between the pads and the input and output leaf cells. As a result, each of these stages should also be individually checked to ensure final correct layout. Mistakes such as wrong spacing between interconnect wires or between interconnect wires and cells, are a common source of this design time extension. Often, the designer may find that the estimated space required for routing is not enough and using fixed floor plan design techniques, as with CSIRO MPC's, this can often force a designer to try and compact leaf cells in an effort to fit a design within the allowed floorplan space. This type of redesign work using mask level layout tools can take longer than the original leaf cell designs themselves.

As a final check, the complete Signature Analyser Design is checked, Fig. 4.5 shows the output produced from the Design Rule Checker along with the time taken for the Tool to complete the check. Note that the warnings given in the output file result

```
$ check [jnoonan.sub1]shiftreg.cif
No Cif Errors Found
bounding box is  -3,-11 to  33, 59
PROCESSING TILE   1 ,   1
tile is   -13,  -21 to    43,   69

0 CIF errors, 0 CIF warnings.
0 design rule errors.

  Accounting information:
  Buffered I/O count:             61      Peak working set size:   952
  Direct I/O count:               71      Peak page file size:    1770
  Page faults:                  2174      Mounted volumes:           0
  Charged CPU time:      0 00:00:06.80    Elapsed time:      0 00:00:35.23


$ check [jnoonan.work]comp.cif
No Cif Errors Found
bounding box is   0,-31 to 122,101
PROCESSING TILE   1 ,   1
tile is   -10,  -41 to   132,  111

0 CIF errors, 0 CIF warnings.
0 design rule errors.

  Accounting information:
  Buffered I/O count:             61      Peak working set size:  1000
  Direct I/O count:               61      Peak page file size:    1770
  Page faults:                  2954      Mounted volumes:           0
  Charged CPU time:      0 00:00:19.42    Elapsed time:      0 00:00:37.03


$ check [jnoonan.work]try.cif
No Cif Errors Found
bounding box is -13,  0 to 848,649
PROCESSING TILE   1 ,   1
tile is   -23,  -10 to   509,  522
PROCESSING TILE   2 ,   1
tile is   489,  -10 to  1021,  522
PROCESSING TILE   1 ,   2
tile is   -23,  502 to   509, 1034
PROCESSING TILE   2 ,   2
tile is   489,  502 to  1021, 1034

0 CIF errors, 31 CIF warnings.
0 design rule errors.

  Accounting information:
  Buffered I/O count:             63      Peak working set size:  1000
  Direct I/O count:               70      Peak page file size:    2075
  Page faults:                 18278      Mounted volumes:           0
  Charged CPU time:      0 00:04:30.73    Elapsed time:      0 00:05:57.63
```

Figure 4.5: Samples of the CHECK analysis of the shift register, comparator and the complete Signature Analyser with their results.

from the fact that some wires used in this design fall on half λ grid boundaries due to the use of the minimum allowable metal wire width for the nMOS technology, 3λ.

Note the result of the check done on the same circuit, but with an error introduced into the shift register as shown in Fig 4.6. The error introduced is that the metal wire width is only 2λ while it should be 3λ, (the minimum width of a metal wire as prescribed by the design rules in Appendix A) in the ground connection of the shift register.

Once the circuit has been passed through the Design Rule Checker, the designer can be guaranteed that his circuit layout obeys all of the layout rules specified and that any malfunction of the circuit will not be due to mask layout error.

4.3.4 Circuit Extraction

Before circuit simulation can be performed, the mask layout of the design must be converted to an electrical circuit representative form. This can be done manually or automatically. If it is done manually, the layout is visually interpreted by the designer and an electronic circuit representation produced in coded form suitable for input to various simulators and some other tools. There is no guarantee that the electrical circuit representation of the mask layout is correct, as errors in the interpretation process are easily made.

To guarantee that the electrical layout is the same as that represented by the mask layout, automatic generation of the electrical circuit form of the design by a Circuit Extraction Tool must be completed. Using the CIF description of the circuit, a Circuit Extractor usually produces various output files that can in turn be used as input to various simulation tools.

The most efficient of the Circuit Extractors discussed in Chapter II, NET, is used to illustrate the function of Circuit Extraction. As indicated in Chapter II, various

```
$ check [jnoonan.thesis.work]fig46.cif

No Cif Errors Found
bounding box is  -3,-11 to  33, 59
PROCESSING TILE   1 ,   1
tile is   -13,  -21 to    43,   69

0 CIF errors, One CIF warning.
22 design rule errors.

  JNOONAN      job terminated at 14-APR-1986 17:21:08.22

  Accounting information:
  Buffered I/O count:            62     Peak working set size:    972
  Direct I/O count:              48     Peak page file size:     1771
  Page faults:                 2349     Mounted volumes:            0
  Charged CPU time:      0 00:00:06.37  Elapsed time:      0 00:00:10.21

                      CIF ERRORS AND WARNINGS
                      -----------------------
Warning, line    94, symbol   999 : No geometry

                      DESIGN RULE ERRORS
                      ------------------

1.3: Metal width between 0,1 and 0,3
1.3: Metal width between 33,1 and 33,3
1.3: Metal width between 0,3 and 0,1
1.3: Metal width between 33,3 and 33,1
2.3: Metal spacing between 33,3 and 33,4
2.3: Metal spacing between 29,4 and 29,3
2.3: Metal spacing between 33,4 and 33,3
1.3: Metal width between 0,23 and 0,25
1.3: Metal width between 15,23 and 15,25
1.3: Metal width between 19,23 and 19,25
1.3: Metal width between 33,23 and 33,25
1.3: Metal width between 0,25 and 0,23
1.3: Metal width between 15,25 and 15,23
1.3: Metal width between 19,25 and 19,23
1.3: Metal width between 33,25 and 33,23
2.3: Metal spacing between 29,44 and 29,45
2.3: Metal spacing between 33,44 and 33,45
1.3: Metal width between 0,45 and 0,47
1.3: Metal width between 33,45 and 33,47
2.3: Metal spacing between 33,45 and 33,44
1.3: Metal width between 0,47 and 0,45
1.3: Metal width between 33,47 and 33,45

0 CIF errors, One CIF warning.
22 design rule errors.
```

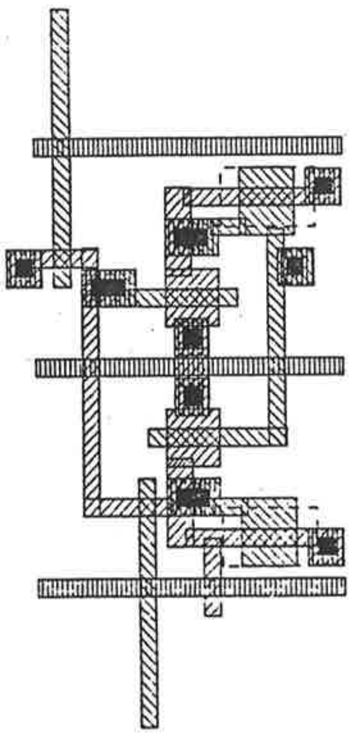


Figure 4.6: The CHECK analysis of the Shift Register design and its results with an introduced error, the error being 2λ wide metal wires instead of the required 3λ.

levels of simulation exist with the lowest levels of simulation providing the most accurate and detailed description of the electrical operaton of the design. The circuit extractor provides output for the tools SPICE, MOSSIM, MOSERC and PROBE (see Chapter II). The output required for these various tools is different and so the tool NET can be run with a number of different command line options which will allow output variation according to whether the designer is working with nMOS or CMOS circuits and which tool the designer wishes to use for simulation or electrical rule checking.

Apart from the technology independence and various types of output that are available using NET, the tool has other useful features. A command line option exists which will allow the designer to generate CIF nodelabel statements for all of the node numbers allocated by NET. This allows the designer to display the node numbers together with the full design with the aid of a design tool such as SEE (described in Chapter III), so that the user can instantly see the location of the node numbers. Another useful feature of NET is the command line option which allows syntax checking of the CIF input.

The various output formats for the simulation tools will be discussed in the following section. The results of the two features of NET discussed immediately previously as applied to the Signature Analyser can be seen in Fig. 4.7. The various output formats for the different tools for which NET provides input can be seen in the following diagrams using the Shift Register from the Signature Analyser as the example CIF input.

Fig. 4.8 shows the output file produced by NET that is used by the simulator SPICE. The SPICE models for MOSFETS may change from time to time depending upon the process used for the fabrication of designs and so should be viewed as an example of the type of parameterisation expected. It should be noted that the simulator PROBE takes its input from this SPICE format output as well as the technology file, examples of which can be seen in Fig. 4.9. The technology files used with NET and PROBE allows NET to determine capacitance values between CIF layers and uses these

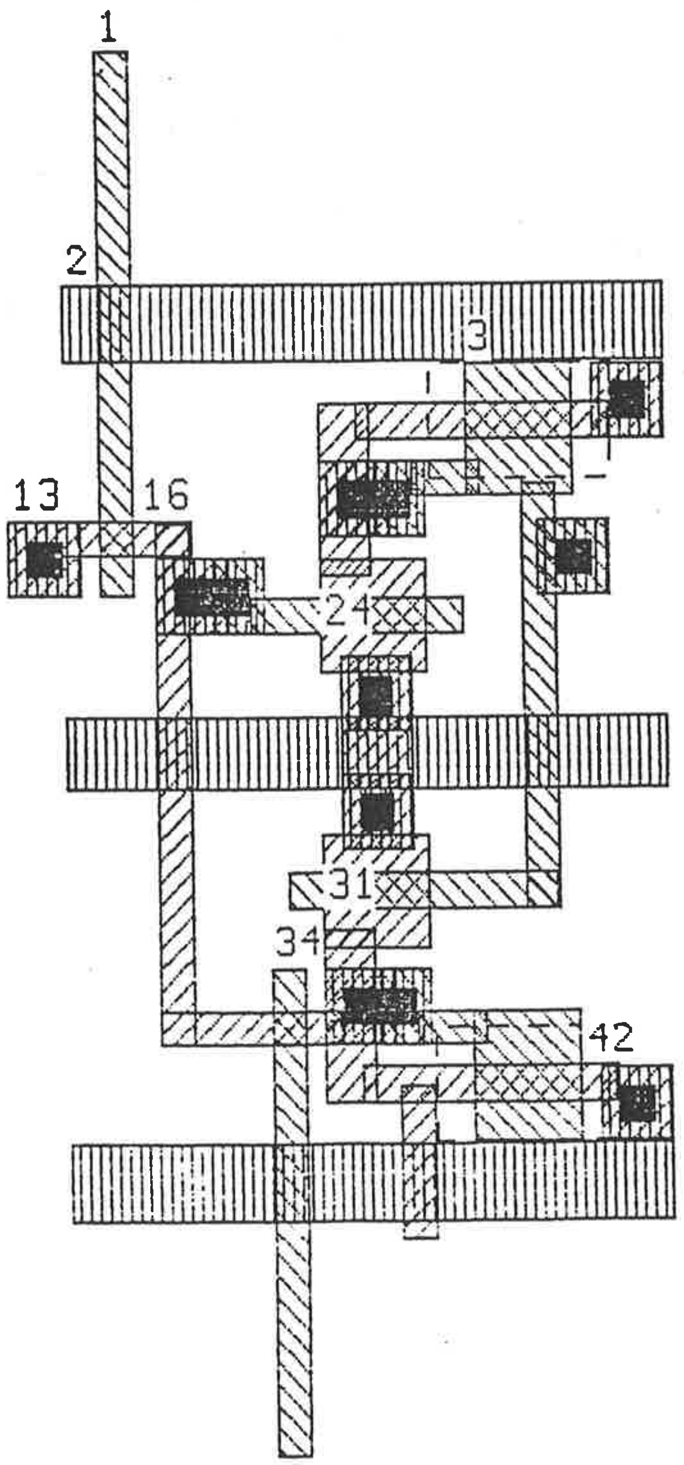


Figure 4.7 (a): An example of the node labels on the shift register produced with the NET node option.

```
  80 1
  81 1
  82 1 (nodefile created by NET from D1:[JNOONAN.THESIS.WORK]SHIFTREG.CIF;
  83 1  at 13:52:18.49 on 17-JAN-1986);
  84 1 DS 999;
  85 1 94 1 500,15000 NM;
  86 1 94 2 0,12000 NM;
  87 1 94 3 5500,11000 NM;
  88 1 94 13 -750,9000 NM;
  89 1 94 16 1000,9000 NM;
  90 1 94 24 3500,7500 NM;
  91 1 94 31 3500,4000 NM;
  92 1 94 34 2750,3250 NM;
  93 1 94 42 7000,2000 NM;
  94 1 DF;
  95 1 C 999;
  96 1 End

Technology is  nMOS, Lambda=250
The bounding box of the complete design is :-
   Lower left  (x,y)    ( -3, -12 ) Lambdas
   Upper right (x,y)    ( 33, 60 ) Lambdas

%NET-I-SCANLINE, scanline now passing y=15000
%NET-W-RENAMELABEL, attempt to rename label "2" to "3" ignored
%NET-I-SCANLINE, scanline now passing y=4500
%NET-I-OUTPUTDEV, writing devices and nets

Summary of Devices and Nets

Number of Nets            : 9
Number of Enhancement     : 4
Number of Depletion       : 2
Total Number of Devices   : 6

%NET-W-INVALIDLABEL, node label 1 without associated geometry
%NET-W-INVALIDLABEL, node label 16 without associated geometry
%NET-W-INVALIDLABEL, node label 34 without associated geometry
%NET-W-INVALIDLABEL, node label 24 without associated geometry
%NET-W-INVALIDLABEL, node label 31 without associated geometry
%NET-W-INVALIDLABEL, node label 42 without associated geometry
```

Figure 4.7 (b): An example of the listing output for the shift register produced with the NET listing option.

to calculate node capacitances, while PROBE derives the models for its transistors from this technology file.

Fig. 4.10 shows the output file produced by NET for use with MOSSIM and MOSSERC. This output is much less complex than that used for SPICE which indicates also the less complex nature of the higher level simulation tool MOSSIM.

The circuit extractor NET is currently used for the verification of all mask level layouts designed at the University of Adelaide.

4.3.5 Simulation

As discussed in Chapter II, the various levels of simulation that exist present various degrees of accuracy in the estimation of circuit operation as well as associated various degrees of complexity in simulation. The need for accuracy of simulation must be decided by the designer. Ideally, all parts of a design should be simulated as accurately as possible. This would imply that all parts of a design should be simulated with the lowest level simulation tool available, which in this case is SPICE.

However, design time restrictions often prevent the realisation of this ideal and as a result, the designer can be forced to settle for the less accurate simulation offered by the higher level tools such as PROBE or MOSSIM. In the case of the Signature Analyser design, the deadlines for entries to the first AUSMPC presented restrictions on design time which in turn prevented the completion of the ideal simulation. As a result only MOSSIM could be used as a verification tool at the simulation level prior to final design submission, however as an illustration of the various levels of simulation available, examples of simulated circuit operation for parts of the Signature Analyser will be presented for each of the simulation tools, SPICE, PROBE and MOSSIM.

As an example of the use of SPICE, the NET output for the Shift Register shown in Fig. 4.7(a) was used to produce the response shown in Fig. 4.11. It is very important at this stage to understand that the response predicted by SPICE is greatly

```
* SPICE FILE CREATED BY NET FROM D1:[JNOONAN.THESIS.WORK]SHIFTREG.CIF;
* AT 14:43:04.47 ON 20-JAN-1986
*
* NMOS SPICE ( VERSION 2G ) MODEL PARAMETERS
*
.MODEL ENH NMOS
+ VTO=0.84       GAMMA=0.47      PHI=0.52        LAMBDA=0.02
+ RSH=12         CGSO=1.6E-10    CGDO=1.6E-10    CGBO=1.7E-10
+ TOX=8.5E-8     PB=.539         JS=5.25E-4      XJ=5E-7
+ LD=0.8U        TPG=+1          UO=910          UCRIT=1E4
+ UEXP=0.43      UTRA=0.2        FC=0.5          CJ=1.1E-4
+ MJ=0.5         CJSW=1E-9       UCRIT=1E4       UEXP=0.43
+ UTRA=0.2       FC=0.5          CJ=1.1E-4       MJ=0.5
+ CJSW=1E-9
*
.MODEL DEP NMOS
+ VTO=-2.86      GAMMA=0.53      PHI=0.52        LAMBDA=0.02
+ RSH=12         CGSO=1.6E-10    CGDO=1.6E-10    CGBO=1.7E-10
+ TOX=8.5E-8     PB=.604         JS=5.25E-4      XJ=5E-7
+ LD=0.8U        TPG=+1          UO=800          UCRIT=1E4
+ UEXP=0.43      UTRA=0.2        FC=0.5          CJ=1.1E-4
+ MJ=0.5         CJSW=1E-9       UCRIT=1E4       UEXP=0.43
+ UTRA=0.2       FC=0.5          CJ=1.1E-4       MJ=0.5
+ CJSW=1E-9
*
C1 1 0 27.50F
C2 2 0 73.88F
C3 3 0 242.75F
C4 13 0 49.13F
C5 16 0 245.62F
C6 24 0 140.75F
C7 31 0 225.75F
C8 34 0 25.63F
C9 42 0 73.88F
*
M1 42 31 31 0 DEP L=15.00U W=5.00U AS=375.00P PS=140.00U AD=987.50P PD=220.00U
M2 2 3 3 0 DEP L=15.00U W=5.00U AS=725.00P PS=235.00U AD=975.00P PD=215.00U
M3 31 3 24 0 ENH L=5.00U W=15.00U AS=275.00P PS=60.00U AD=375.00P PD=140.00U
M4 24 16 3 0 ENH L=5.00U W=15.00U AS=725.00P PS=235.00U AD=275.00P PD=60.00U
M5 31 34 16 0 ENH L=5.00U W=5.00U AS=425.00P PS=175.00U AD=375.00P PD=140.00U
M6 16 1 13 0 ENH L=5.00U W=5.00U AS=162.50P PS=55.00U AD=425.00P PD=175.00U
```

Figure 4.8: The SPICE output file produced using NET for the shift register.

NMOS.TEC

```
!model of transistor here
!LAYER  AREA    PERIMETER
CNM     0.3E-04   0.0         METAL TO SUBSTRATE
CNP     0.5E-04   0.0         POLYSILICON TO SUBSTRATE
CNT     4.5E-04   0.0         GATE TO SUBSTRATE
CND     0.9E-04   8.0E-10     DIFFUSION TO SUBSTRATE
!END
```

CMOS.TEC

```
!model of transistor here
!LAYER  AREA    PERIMETER
CCM     0.3E-04   0.0         METAL TO SUBSTRATE
CCP     0.5E-04   0.0         POLYSILICON TO SUBSTRATE
CCT     4.5E-04   0.0         GATE TO SUBSTRATE
CCDN    0.9E-04   8.0E-10     N_TYPE DIFFUSION TO SUBSTRATE
CCDP    0.9E-04   7.0E-10     P_TYPE DIFFUSION TO SUBSTRATE
!END
```

Figure 4.9: The two technology files NMOS.TEC and CMOS.TEC used to provide SPICE MOSFET model definition, and layer to substrate capacitance of the active layers, for both NET and PROBE.

```
| title SHIFTREG
C 1 GND 28
C 2 GND 74
C 3 GND 243
C 13 GND 49
C 16 GND 246
C 24 GND 141
C 31 GND 226
C 34 GND 26
C 42 GND 74
d 31 31 42 1500 500 5500 2000
d 3 3 2 1500 500 5500 10500
e 3 24 31 500 1500 3500 4500
e 16 3 24 500 1500 3500 8000
e 34 16 31 500 500 2750 2750
e 1 13 16 500 500 500 9000


.title SHIFTREG
C1 1 vss 27.50f (2, 60)
C2 2 vss 73.88f (0, 48)
C3 3 vss 242.75f (22, 44)
C4 13 vss 49.13f (-3, 36)
C5 16 vss 245.62f (4, 36)
C6 24 vss 140.75f (14, 30)
C7 31 vss 225.75f (14, 16)
C8 34 vss 25.63f (11, 13)
C9 42 vss 73.88f (28, 8)
M1 42 31 31 vss depl 15.00u 5.00u (22, 8)
M2 2 3 3 vss depl 15.00u 5.00u (22, 42)
M3 31 3 24 vss enl 5.00u 15.00u (14, 18)
M4 24 16 3 vss enl 5.00u 15.00u (14, 32)
M5 31 34 16 vss enl 5.00u 5.00u (11, 11)
M6 16 1 13 vss enl 5.00u 5.00u (2, 36)
```

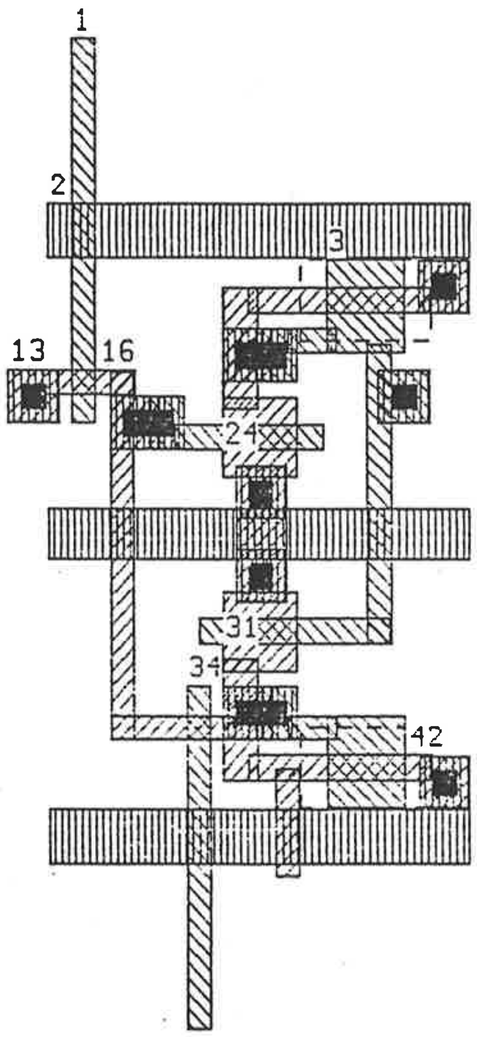


Figure 4.10: Above, the SIM output file produced using NET for the shift register. This file is used as input to the Berkeley tools, MOSSIM and MOSERC. Below, the NET output file used with PROBE and ELEC.

dependent upon the MOSFET model parameters. Using the Silicon Broker technique for circuit fabrication, it is very difficult to obtain accurate information regarding these parameters for a number of reasons. First, the various fabrication houses tend to jealously gaurd this type of information as proprietary in nature. Second, using a Broker/Foundry does not guarantee that the designs will be fabricated on a known process, only that designs adhering to a general set of Design Rules will function. This leaves a wide tolerance open to most of the MOSFET model parameters resulting in difficulty in predicting accurate values for the different parameters. Despite these facts, the SPICE simulation normally will provide the most accurate simulation results for a design and is desirable at all levels of the design to verify electrical operation of the circuit.

PROBE provides the next most accurate simulation of the design and Fig. 4.12 is the resultant simulation of the Shift Register using this tool. The results of this simulator, while less accurate than that of SPICE are produced much more quickly making PROBE a more attractive tool for designers without enough time to use SPICE.

Finally, MOSSIM has its output displayed in Fig. 4.13. This type of simulator provides absolutely no timing information, rather it gives an indication of the logical correctness of the circuit. Hence the cryptic form of the output.

As an indication of the time restrictions that can be placed on a designer or a team of designers, the AUSMPC 5/82 experience is an excellent example. The Signature Analyser design was completed in time only to allow simulation with MOSSIM. Restrictions on access to tools added to the time restriction reasons for this undesirable level of electrical operation checking. However, even had all of the tools discussed in Chapter II been available, there still would have been great difficulty in carrying out further simulations prior to the CSIRO imposed deadline for submission of projects. The SPICE and PROBE simulations shown in this section are examples only of the types of simulation most desirable for a prototype design prior to submission to fabrication of an IC design.

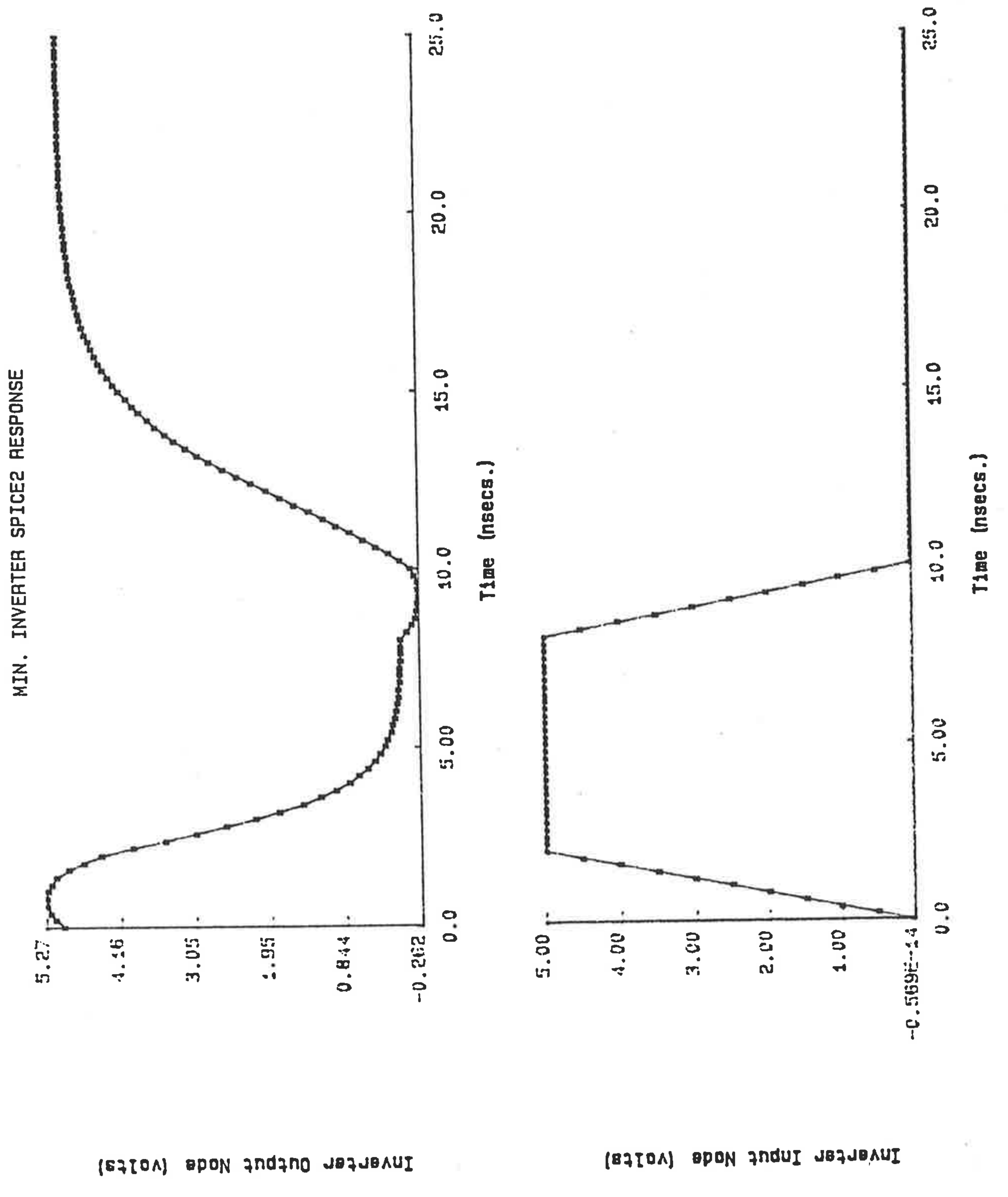


Figure 4.11: The results of a SPICE simulation of the shift register presented using a locally produced plotting program called GPLOT[91].

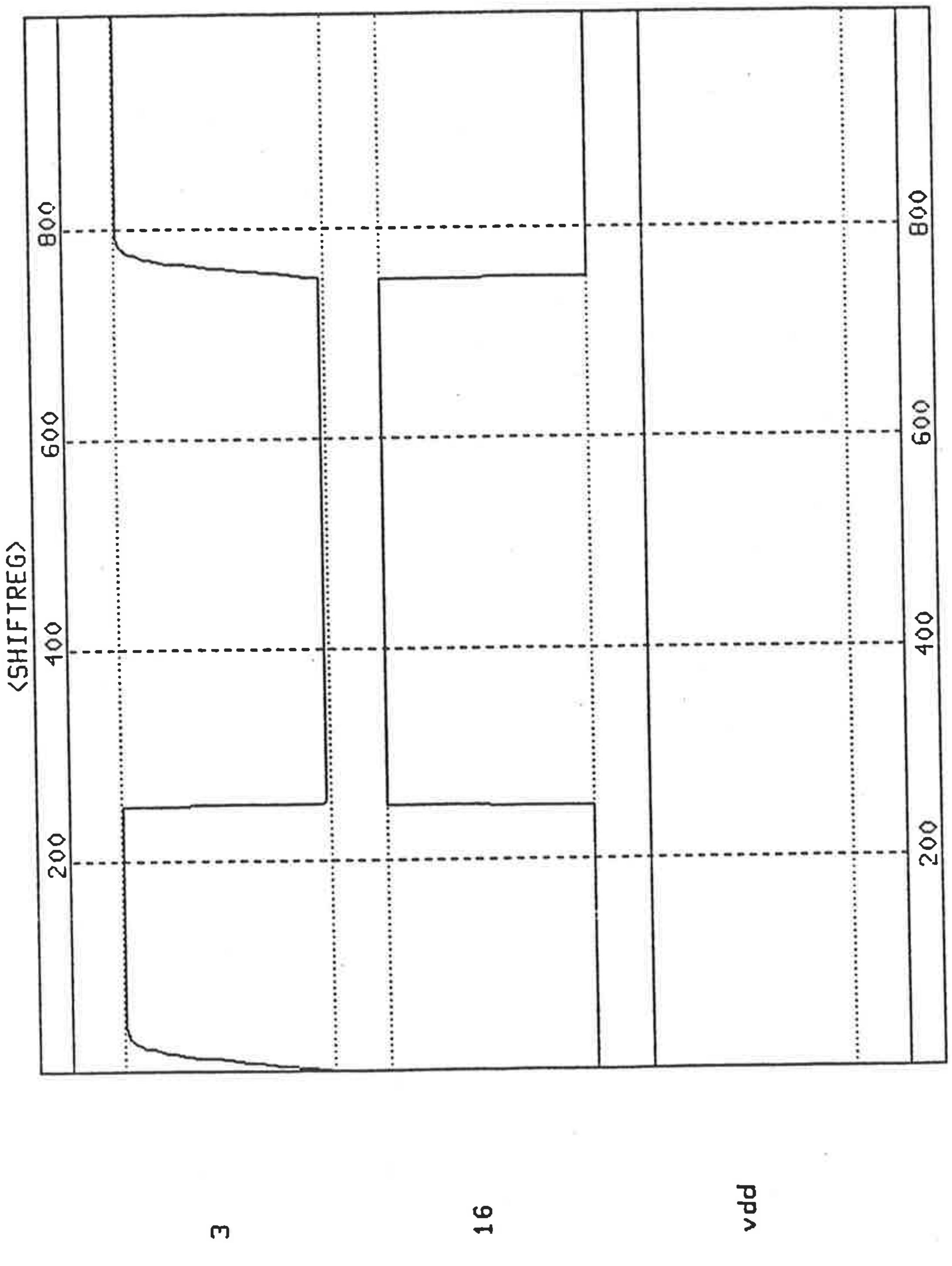


Figure 4.12: The results of a PROBE simulation of the shift register. The x axis shows time in nanoseconds while the y axis shows volts in the 0 to 5 volt range.

```
I
w PHI2SL PHI2NSL PHI1SL PHI1NSL SIGOUT SIGSRIN MSKOUT MSKSRIN vdd COMPOUT
w gnd PHI1DL PHI1NDL PHI2NDL PHI2DL DATOUT DATSRIN
V PHI1SL 00000000000000000100010010001000
V PHI1NSL 00000000000000000001000100010001
V PHI2SL 01000100010001000000000000000000
V PHI2NSL 00010001000100010000000000000000
V SIGSRIN 11110000
V MSKSRIN 11111111111111110000000000000000
V DATSRIN 11110000
V PHI1DL 00000000000000000100010001000100
V PHI1NDL 00000000000000000001000100010001
V PHI2NDL 01000100010001000000000000000000
V PHI2DL 00010001000100010000000000000000



JNSIM.LIS;4                    28-MAY-1982 12:16:47.80          Page 1

248 transistors, 202 nodes (161 pulled up)
initialization took 432 steps
>0100010001000100000000000000100010001000000000000000000000000000:PH12SL
>0001000100010001000000000000001000100010001000000000000000000000:PHI2NSL
>0000000000000000010001001000100000000000000000000100010010001000:PHI1SL
>0000000000000000010001000100010000000000000000000001000100010001:PHI1NSL
>XXXXXXXXXXXXXXX1111111111111111111000011110000111111111111111111:SIGOUT
>1111000011110000111100001111000011110000111100001111000011110000:SIGSRIN
>XXXXXXXXXXXXXXX1111111111111111111111111111111111111111111111111:MSKOUT
>1111111111111111000000000000000011111111111111110000000000000000:MSKSRIN
>1111111111111111111111111111111111111111111111111111111111111111:vdd
>XXXXXXXXXXXXXXXXXX0000XXX00001111000011110000111100001111000011 1:COMPOUT
>0000000000000000000000000000000000000000000000000000000000000000:gnd
>0000000000000000010001000100010000000000000000000010001000100100:PHI1DL
>0000000000000000010001000100010000000000000000000001000100010001:PHI1NDL
>0100010001000100000000000000100010001000000000000000000000000000:PHI2NDL
>0001000100010001000000000000001000100010001000000000000000000000:PHI2DL
>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX1111111111111111111110000111100001:DATOUT
>1111000011110000111100001111000011110000111100001111000011110000:DATSRIN
```

Figure 4.13: The results of MOSSIM simulations on the Signature Analyser design.

The time taken to complete the layout of this design was approximately three months. In that time, the designers had to become familiar not only with the CAD Tools, but also the PASCAL programming language, the VAX 11/780 and VMS, and the local *Text Editor* **LUDWIG**[85], as well as generate the design using a methodology foreign to them. The resultant design was considered good enough to be accepted on the first CSIRO AUSMPC against strong competition[17].

The number of design hours is difficult to estimate and the definition of a *design hour* is also not clear cut. The design was completed on paper in about 40 man hours. The time taken to put the design on the computer in the correct format was about 200 man hours. The time taken to fully debug the design in order to pass for AUSMPC 5/82 was around 200 man hours but this time could have been shortened if adequate design checking facilities were more readily available. Documentation time would have been approximately 100 man hours, although subsequently documentation aids such as extremely efficient typesetting and word processing programs have become available that ease even this time consuming task[75]. An approximation of total design hours for the Signature Analyser subdivided into component categories was as follows:- (a) Algorithim .05 of total, (b) Floorplan .1 of total, (c) Cell layout .2 of total, (d) Composition .2 of total, (e) Checking .45 of total.

4.4 IC Testing

Testing of a circuit on silicon is a non-trivial matter for many reasons[74][85][84]. Faults can occur at any stage of the design or fabrication process. The faults can be introduced by the designer or at some stage in the mask making, fabrication or packaging or other intermediate processes that are beyond the designer's control, particularly in an MPC run.

To verify whether or not an IC is operational, testing must be approached in an organised fashion. Maxwell[74] indicates the two basic types of IC testing required, (a) fabrication process testing and (b) design functional verification. Using a Silicon

Broker/Foundry, normally type (a) testing is done for the designer prior to designs being returned although this is not the case for all Silicon Brokers/Foundries. One such Foundry service offered in Australia for example leaves all testing, both type (a) and (b) totally up to the MPC designer[90].

4.4.1 Fabrication Process Testing

Before checking the functional operation of a design, the designer needs to be sure that there is no fault introduced in the intermediate processing steps that occur between the submission of the design in CIF format to the Broker and the return of the packaged prototype. This type of testing requires checking complete wafers using special test structures inserted in addition to the functional logic designs submitted by MPC participants. These tests determine whether or not the fabrication run has satisfactorily met specified requirements with respect to device thresholds, resistivities, electrical separation of layers, dielectric integrity, and metal step coverage. Device performance is also commonly determined with respect to propagation delay, DC characteristics, and dependence on temperature[74].

These types of test usually require analogue measurement and the test structures used are independent of the functions of the designs being fabricated. The structures are inserted into separate areas of the wafer, by the fabrication house and/or may be included in the form of a starting frame for the design as was the case for AUSMPC 5/82.

An example of the test structures used can be seen in the AUSMPC 5/82 Starting Frame[6] shown in Fig. 4.14. The starting frame is included in the top strip of each die type on the **MPW** (*Multi-Project Wafer*. It is used to determine whether or not the fabrication process has successfully produced working system level devices and allows some characterisation and performance measurements to be made. These measurements need not be carried out by the designers using the MPC implementation system, it is only provided for interest.

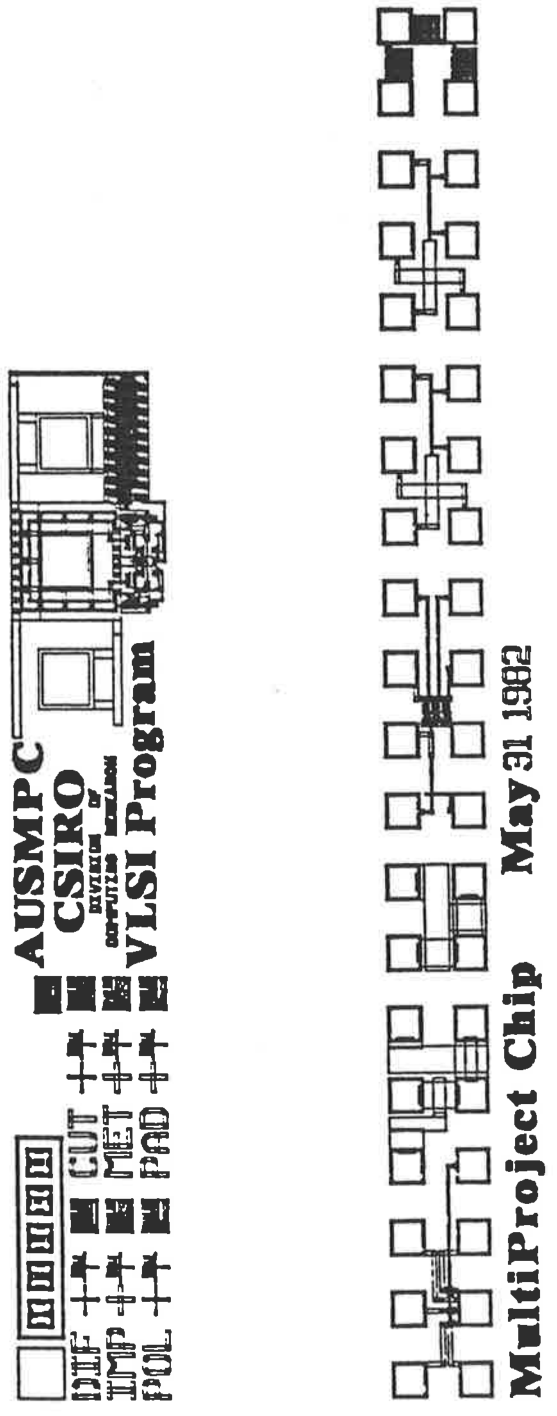


Figure 4.14: The starting frame used in AUSMPC 5/82.

The AUSMPC Starting Frame consists of, (i) Alignment Marks, (ii) Layer Codes, (iii) Critical Dimension Testers, (iv) Etch Test Patterns (ELLS), (v) Identification Code, (vi) Ring Oscillator and (vii) Test Structures. The details of this starting frame are taken from Clarke's *AUSMPC 5/82 Designer Documentation*[6], distributed to all designer participants of AUSMPC 5/82.

Both coarse and fine alignment marks are provided. Coarse alignment is provided by a square appearing on all layers in the top left hand corner of each die. Fine alignment can be done using the sequence of *squares* and *fortresses* which appear to the right of the coarse alignment squares. These alignment structures were taken from the report by Hon and Sequin[7].

The layer codes (DIF, IMP, POL, CUT, MET and PAD) appear under the alignment marks. Alongside each layer code are two critical dimension crosses and a set of L-shaped test patterns. Note that the IMP and PAD layers are invisible on photomicrographs of the die as opposed to their appearance on the diagram in Fig. 4.14 which is a plot of the complete starting frame using a single colour.

The line width for the first of the two critical dimension crosses is given in Table 4.1, while the width for the second cross is always 2λ which in this case equals $5\mu m$.

Critical Dimension Cross Line Width			
Layer	*Width*	*Layer*	*Width*
DIF	2λ	CUT	2λ
IMP	4λ	MET	4λ
POL	2λ	PAD	4λ

Table 4.1: Line width of the first of the two critical dimension crosses shown in Fig. 4.14.

The Etch Test Patterns which appear to the right of the second critical dimension cross for each layer are 26λ high and consist of Ells of two sizes. Five small Ells with one λ width and separation are nested in the upper right corner of four larger Ells having two λ width and spacing. Also included is a vertical two λ wide bar down the left side of the pattern to simplify measurement.

Each of the nine die types used on the Multi-Project Wafers used to provide design beds for AUSMPC 5/82 is identified by a single character (A...I) which can be seen by the naked eye. The identification character is 360μm tall and appears at the top right hand corner of each die.

The ring oscillator has nineteen stages and can be used to estimate the speed of the devices made using this fabrication process. It consists of nineteen identical inverters in a circle and a twentieth inverter acting as a buffer to drive a standard output pad. Each inverter has a pullup ratio of $Z_{pu} = 2 : 1$ and a pulldown ratio of $Z_{pd} = 1 : 2$, resulting in minimum geometry $k = 4$ inverters. The period of oscillation of the ring oscillator T equals twice the delay around the loop. Therefore the inverter pair delay is $\frac{T}{n}$, where n is between 19 and 20 since the last inverter drives two loads, (the first inverter and the buffer inverter). The connections to the pads for the ring oscillator are from left to right: Ground, Output, VDD.

The test structures are laid out between two strips of 20 probe pads. Each pad is 80μm by 80μm with an overglass cut of 70μm by 70μm. The pads are spaced apart by 80μm in both the vertical and horizontal directions. The test structures for AUSMPC 5/82 include, (i) small transistors, (ii) a large inverter, (iii) large transistors, (iv) small inverters (v) Diffusion and Polysilicon Van Der Pauw structures[74] and (vi) a contact tester. These structures allow among other things, the calculation of parameters useful for SPICE modelling and their use is detailed in the report by Clarke[74].

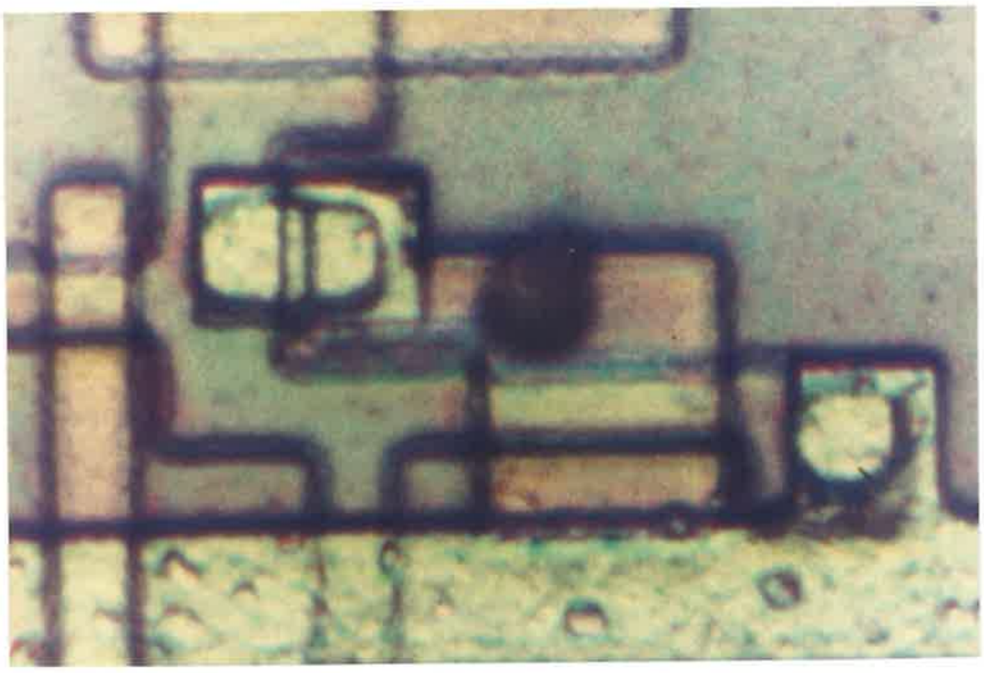
The fabrication process testing is useful to the Silicon Broker/Foundry as it can almost guarantee that any problems with MPC participants' circuits are due to design faults rather than faults produced in the mask making, fabrication or packaging process for which a Broker may potentially be held responsible. This is not always the case however. Through participation in AUSMPC 5/82, experience of problems caused by bonding was gained. Fig. 4.15 shows examples of the faults (severe gauging of a circuit to the point where wires are ruptured, or splashing of molten metal fragments upon a circuit causing possible short circuits) that can occur in the bonding process. These faults can render a design useless as occured in one of the examples shown in Fig. 4.15. Other types of problems that could potentially arise have been mentioned in Chapter III. Weste[48] has mentioned the need to consider small antifeature dimensions as possible sources of faults in the mask making process.

4.4.2 Design Functional Verification Tests

The type (b) testing mentioned by Maxwell[74], (design functional verification), must be completed by the designer. It assumes that fabrication has been successful and is necessary to determine whether or not the circuit works as intended. This testing problem can be subdivided into *Production Testing* as opposed to *Prototype Testing*[74]. Production Testing is performed on designs that have already proven to be functionally correct but individual chips may be faulty due to fabrication processing faults that are randomly scattered over the surface of the wafer. Production Tests are usually quite simple, *go/no go* tests[74].

Prototype Testing and fault finding is the type of testing that a typical MPC participant would do. This type of testing requires interpretation of the test data to determine the origins of faults if they exist. For the Signature Analyser, this type of testing was completed using a Logic Analyser/Function Generator.

A total of five chips were returned to each design team participating in AUSMPC 5/82. These came from two different fabrication sources, COMDIAL in Sunnyvale California, who fabricated a three inch wafer, and AMI in Idaho who fabricated a four inch wafer. The chips were packaged by Promex, (Palo Alto), who packaged the four inch set and Philips, (Adelaide South Australia), who packaged the three inch set. The duplication of fabrication was intended to reduce the probability of failure as much as possible.



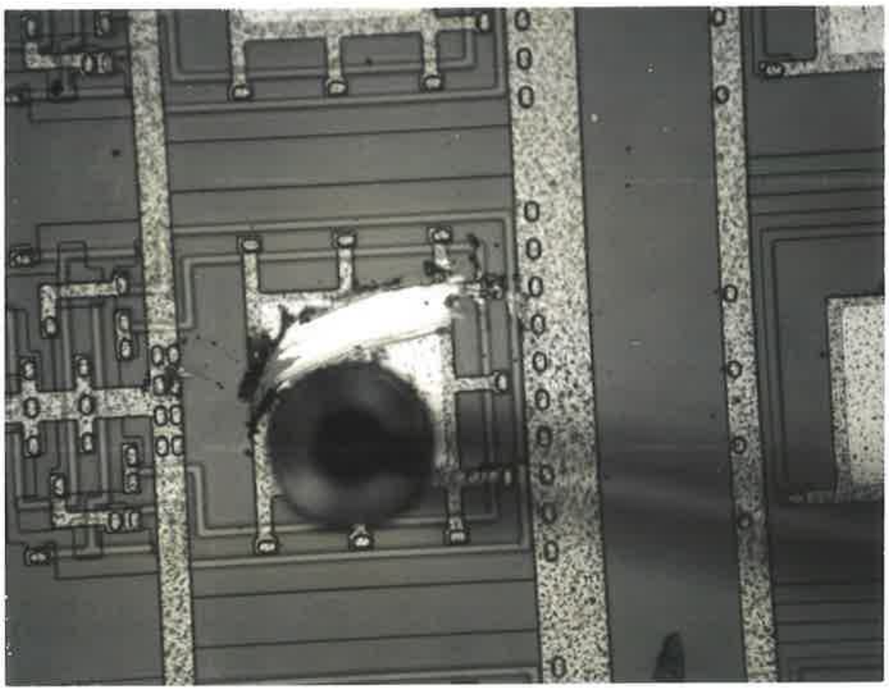


Figure 4.15: The picture above shows a small blob of metal splashed onto one of the shift registers on one of the Signature Analyser dice. This device was protected from such a fault because of the overglass used to cover the project. The picture below shows the result of a wiring failure to an output pad on a project from AUSMPC 5/82. The pad is damaged so badly that the connection to the circuit is ruptured making the device useless.

One of the project C4 chips was extensively tested. It was a Promex packaged chip. The Promex chips were used in the extensive test because they were the first to be returned. The Signature Analyser design is well partitioned. Each four bit shift register string can be separately accessed to observe its ability both to hold and to shift information through the design. The comparator output can also be observed to verify its operation. The tests can be placed into three groups ordered by their complexity. The three tests were:- (1) Shift and refresh all three registers (mask, signature, and input) with all one's and then all zero's and observe the comparator for correct operation. (2) Repeat the first test, but this time with an input which continuously oscillates between one and zero for each input of the shift registers. (3) Finally, using completely random inputs, observe all register outputs and comparator operation. The patterns used were generated on a programmable pattern generator at the CSIRO laboratories.

Also, in this series of tests the chip was pushed to its upper operative frequency limit which was found to be 2.85 MHz for the output of information from the registers. The results of such tests for the Signature Analyser design are displayed in Fig. 4.16. These tests[16.2] verify the total functional operation of the Signature Analyser and hence the tools used to complete the design.

4.4.2.1 **Regular Bit Pattern Tests**

The pictures in Fig. 4.16 show the input to the mask, signature and input shift registers as indicated by MSKIN, SIGIN and INPIN labels. The bottom 4 waveforms in each case show the output from the mask and signature shift registers, the comparator and the input shift register respectively denoted by MSKOP, SIGOP, CMPOP, and INPOP. The top traces seen on the pictures are unassigned.

For the purpose of clarity, these pictures do not show the two phase non overlapping clock. A combined pattern generator/logic analyser was used to provide the random input bit patterns, but an independent clock was used to generate the two phase non- overlapping clock signal for shifting through and refreshing the shift registers. The frequency of these clock signals was much greater than the clock component of the random bit patterns and in fact was altered at times during the testing procedure. A slight lag in the comparator output response can be seen in some cases showing the time taken for propagation of the bit comparison through the four legs of the chip. This is not obvious in all cases which indicates the alteration in clock frequency to a higher value in some cases causing a smaller propagation delay.

The first of the tests mentioned above is satisfied by observing the results of the subsequent tests. Fig. 4.16 (a) shows a series of pictures indicating the operation of the shift registers. Both their ability to shift information through and hold information is demonstrated. Initially ignoring the comparator and its output, the top picture shows the information being shifted into the three shift registers and the unaffected output verifying the ability of each of the Mask, Signature and Input shift registers to shift information straight through the circuit.

The second and third of the pictures shows the ability of each of the shift registers to hold information. The second picture shows the bit patterns shifted into the Signature and Input shift registers while the Mask shift register input is held high. Understanding that the comparator output (CMPOP) is determined by the result of the boolean equation:-

$$CMPOP = M.S.I + M.\overline{S}.\overline{I} + \overline{M}.P$$

where P is the previous comparator bit output, S is the current Signature bit, I is the current Input bit and M is the current mask bit, with M always high, the comparator output will be high if and only if the contents of the signature and input shift registers are exactly the same. (It is important to note that the P input to the first of the cascaded comparator PLA's is connected to VDD and therefore is held permanently high).

If $MSKIN$ is permanently high as it is in the second picture and this is shifted

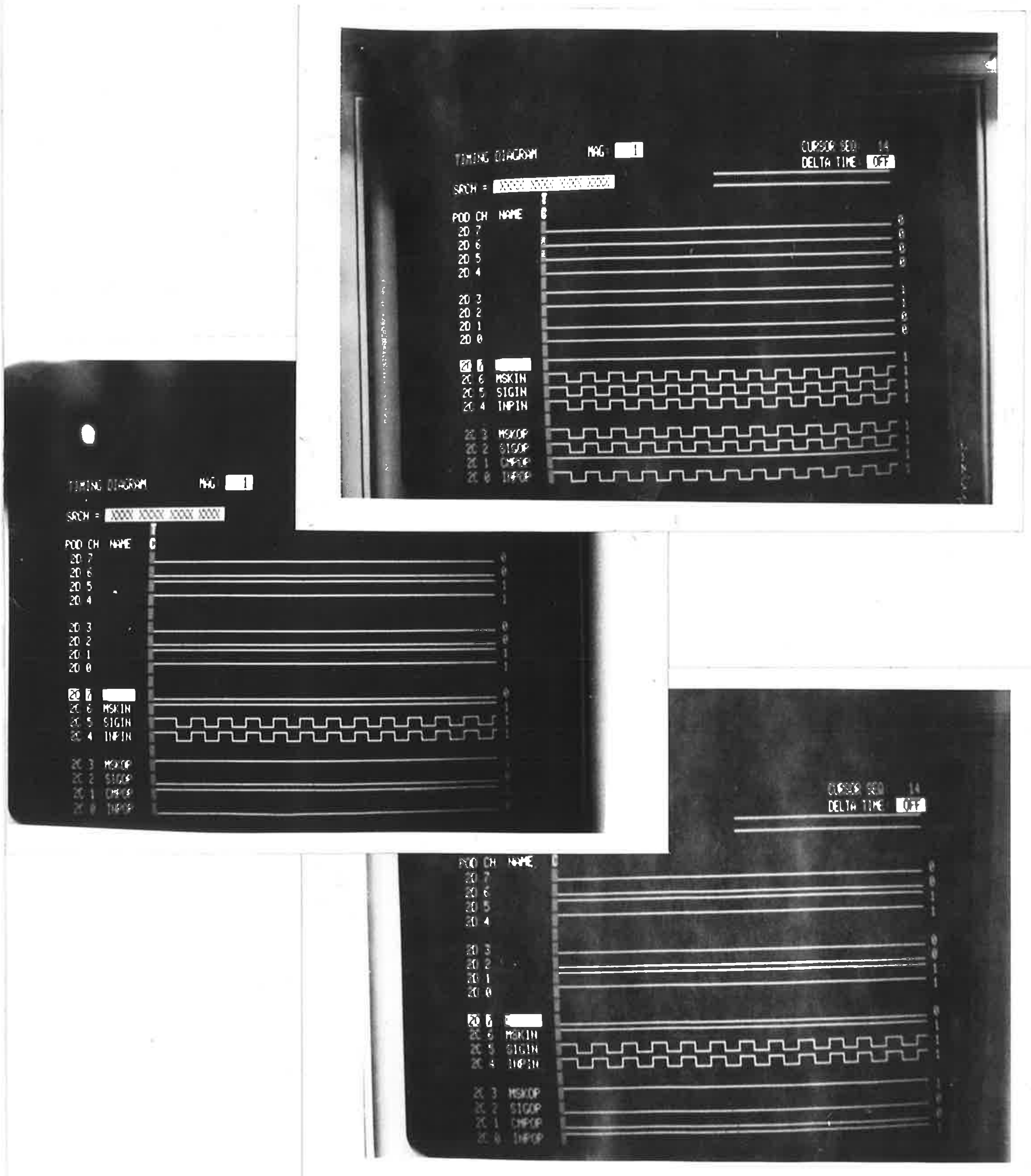


Figure 4.16 (a): Test results of the Signature Analyser, Project C4 AUSMPC 5/82. Pictures are numbered one, two and three from the top down.

through the mask shift register and then held, the Signature Analyser then checks all of the Input and Signature bits for comparison. If they are all the same, then the ***CMPOP*** will be high as in the second picture. This shows that the contents of the signature and input shift registers are exactly the same. In fact, they both contain the pattern shown at their inputs in the second picture.

The third picture shows the results of the comparison when none of the bits in the signature shift register agree with the bits in the input shift register with ***MSKIN*** again held high. The result of course is ***CMPOP*** = 0. The contents of the input and signature shift registers are again indicated by the patterns at their inputs however the fact that the output of the input shift register is high compared to the signature shift register being low shows that the pattern has been offset one bit resulting in all bits being opposite.

For the picture at the top of Fig. 4.16 (a), the clock inputs were set such that the information at the inputs of the shift registers is shifted through, while for the second and third pictures shown the information at the inputs was previously shifted into the shift registers. At the time of the picture being taken for the second and third pictures, the information was being stored in the shift registers.

4.4.2.2 Pseudo-Random Bit Pattern Tests

The pictures in Fig. 4.16 (b) show that the shift registers and comparator all function correctly under the pseudo-random bit pattern test. As explained in the previous section, when the mask shift register is set high, the comparison of the input and signature shift registers is enabled.

The first of the pictures in Fig. 4.16 (b) shows ***MSKIN*** held high with pseudo-random bit patterns shifting through the signature and input shift registers. Apart from verifying the ability of the signature and input shift registers to shift through pseudo-random bit patterns, the result of the comparator output in the first picture also verifies correct operation of the Signature Analyser in a dynamic mode, that is, with information being constantly shifted through.

Features of the comparator operation that are quite clear from this first picture are described here with no reference to precise timing because the aim of this design was to produce a functional design to verify the design tools without setting timing goals.

The fact that all unmasked bits of the signature and input shift registers must produce a high comparator output for ***CMPOP*** to be high, can be seen. For example in the first picture, the ***CMPOP*** can be seen to go low slightly before any change is detected in the levels of either the ***SIGOP*** or the ***MSKOP***. This time delay can be considered as T_F. This time delay indicates that as soon as the first bit of the comparison produces a low output, the whole comparison goes low while the change is not detected at ***SIGOP*** or ***MSKOP*** until the signal has been shifted through the whole four bits.

When the ***CMPOP*** output rises, it does so at precisely the same time as the change of either the ***SIGOP*** or ***MSKOP***. This shows that the ***CMPOP*** output will not go high until all four bits of the ***SIGOP*** and the ***MSKOP*** shift registers produce high comparator outputs.

The second picture in Fig. 4.16 (b) shows a the same pseudo-random input bit pattern as in the first picture and the results but this time for ***MSKIN*** held low. The results of the ***CMPOP*** can be seen to be held constantly high. This is because all of the comparator bits are ignored and the input to the first stage, tied high, is passed through to the output irrespective of any of the input conditions on ***SIGIN*** or ***INPIN***.

The third of the pictures shows a totally pseudo-random input pattern to all of the three shift register inputs, ***MSKIN***, ***SIGIN*** and ***INPIN***. This is a final verification of the complete dynamic funcional operability of the Signature Analyser design. The ***CMPOP*** provides correct information for all cases of the pseudo-random shift register inputs. Interesting features that can be seen are the glitches occuring on ***CMPOP*** when the ***MSKIN*** signal falls and the ***SIGIN*** signal changes state while

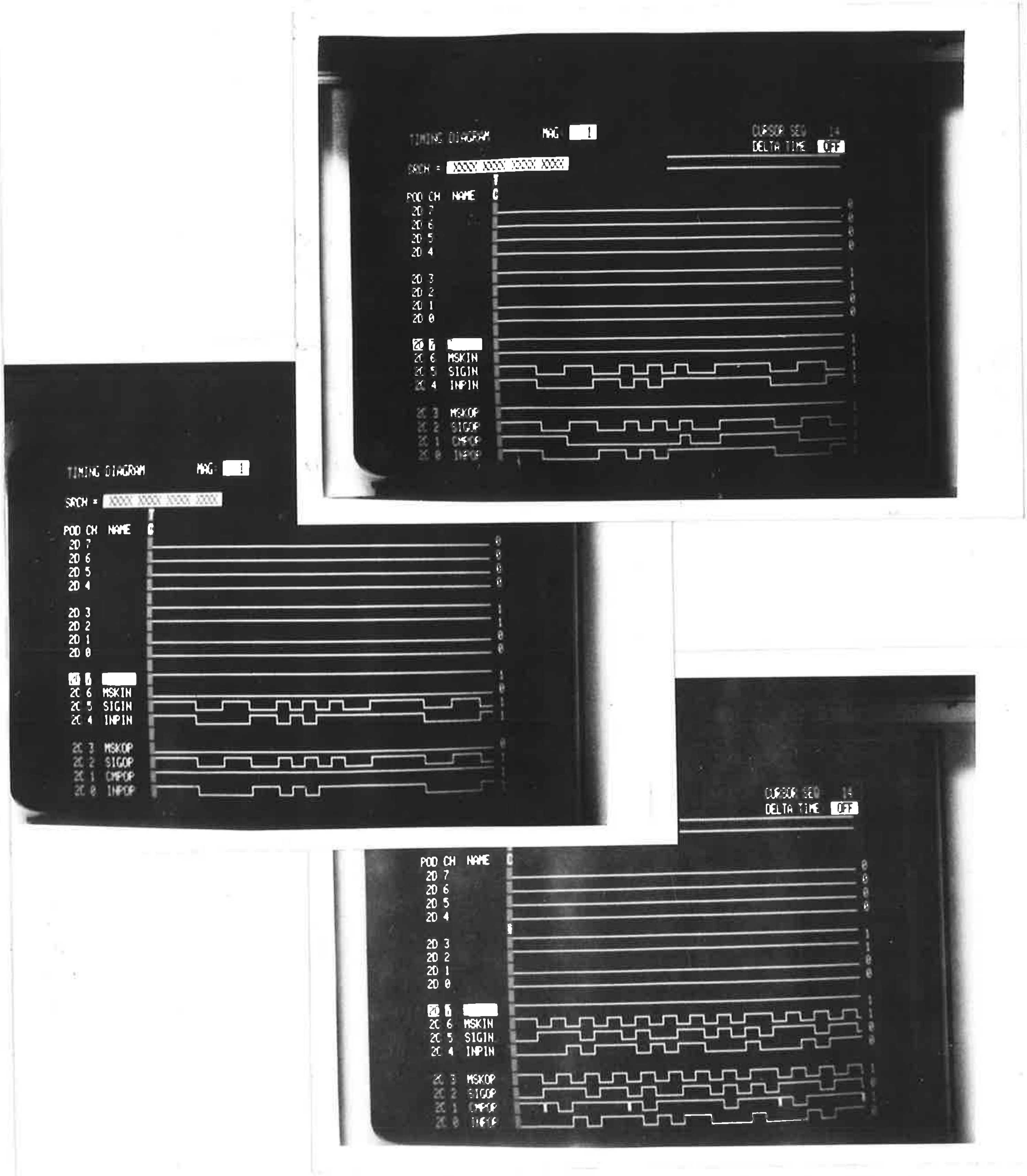


Figure 4.16 (b): Test results for the Signature Analyser using pseudo-random bit patterns. The pictures are numbered one, two and three from the top down.

INPIN is held constant, indicating a possible problem area for timing considerations in production runs of the design.

The testing of the chip has been extensive in its range of functional tests and not so extensive in the area of testing limits of operation, or non ideal operating conditions. However, for the expected environment of operation it would appear that the tests have been extensive enough to show 100% success both in terms of complete operation of all chip functions and in terms of successful operation of all chips, as simple functional tests, such as those used in Fig. 4.16 (a), were performed on the rest of the chips to prove their operation.

Overall, the whole experience can be thought of as a successful implementation of a prototype in silicon. The testing of the final products proved them to function exactly as designed and concluded a very interesting and satisfying experience.

4.5 Analysis of CSIRO MPC Efforts

The CSIRO VLSI department provided XX MPC runs over the period of some three years of operation of which AUSMPC 5/82 was the first. The CSIRO's efforts enabled research into silicon integrated circuit design using an nMOS process by providing the expert intermediate assistance required to let designers efficiently communicate with IC mask makers, fabrication lines, and packaging lines.

In order to successfully complete this task, the CSIRO provided assistance in the distribution of design layout and verification tools, as well as a number of standard cells such as pads which were fundamental to the successful operation of the circuits submitted by designers to these MPC runs. As well, the CSIRO completed the Fabrication Process Testing to provide maximum guarantee that error free dice were finally distibuted to the designers.

The CSIRO design tools such as BELLE were particularly easy to use and indispensible to the design process. Some designers who were not familiar with PASCAL had some initial difficulties but these were rapidly overcome. The only difficulty in the design process was the lack of design verification programs. However the lack of sufficient verification tools has subsequently led to the development of tools locally, (some of which are described in Chapters II and III), which solved the earlier problems.

For AUSMPC 5/82, due to the unavailability of design verification tools, a severe bottleneck was created adding even more pressure to the crucial design submission deadline. The verification tools could not be put on the Adelaide University computers, and as a result, an iterative design process that should have taken at most one and a half hours took one and a half weeks. Designs were ferried via magnetic tape to the CSIRO site where a CSIRO employee ran all designs through the design rule checker, and eventually returned the results. In one and a half weeks the design had passed through the design rule checker only three times. While the checking process was proceeding the designers could do little more than wait, which in the case of project C4 could have been fatal to design completion if the initial design faults had persisted for even one more iteration. MOSSIM was the last of the design verification programs to be used and this could only be done on the CSIRO machine. This was quite a difficult job to arrange also as the deadline was rapidly being approached and again due to the lack of distribution of tools, a single VAX 11/780 was being severely taxed by many users wishing to complete similar time consuming design checks.

All of these problems had to be and were solved. The development of tools as described in Chapters II and III allowed much greater independence for designers at the University of Adelaide to verify their designs. The networking of the University's computing systems to each other and to the **ACSNET**[73] has subsequently given even greater independence to designers. Designers could possibly bypass the Australian intermediaries in the design process and go straight to the US for fabrication should they be able to arrange the additional administrative matters involved in MPC dealings themselves. These administrative matters are non-trivial as experiences with MPC processes within Australia have proven. The CSIRO set and met deadlines extremely well and thereby offered participants of their MPC program a reliable method for the fabrication of IC design prototypes. For example, the deadline for submission of projects for the first AUSMPC, 5/82 was the 31st May 1982 and a package of bonded and unbonded die were returned to the designers by early August 1982 ready for testing.

While the CSIRO no longer provides MPC services, its brief involvement with IC design work acting as a Silicon Broker prompted similar competitive services from other commercial enterprises within Australia that currently offer similar MPC services using fabrication facilities within this country. The exposure of Australian IC designers to the well scheduled professional IC fabrication facilities offered by the companies existing in the United States allows Australia to understand standards that should be expected for IC fabrication.

4.6 A Control Unit for a Four Bit Microprocessor

The success of the Signature Analyser design submitted to AUSMPC 5/82 proves the worth of many of the layout and verification tools discussed in Chapter II as well as the PLA discussed in Chapter III. The tools from Chapter I that remain unverified as far as having produced a working nMOS IC is concerned were verified in the design of the project described in this section.

In 1984, Webber and Dunis at Adelaide University were assigned a project[76] consisting of an nMOS IC design of a control unit for a 4-bit microprocessor. This design was to be completed using any of the layout and verification tools discussed in Chapter II and also the PLAGEN program. When the design was completed, it was intended that it would be fabricated using the JMRC MPC process, depending on satisfactory design verification using the verification tools.

The control unit was to be used in conjunction with a data path chip fabricated on CSIRO AUSMPC 5/82 and designed by Lee and Loo[78], also as undergraduates at Adelaide University. The data path chip was tested by Loja[79] using a control unit built out of SSI, TTL chips and found to function completely. As a result it was decided to design and fabricate a complete control unit for the data path chip. An attempt was made at the design by two undergraduates in 1983[80] using the lowest level layout tools, BELLE etc. This design was not successfully completed for many reasons, among them the inability of the designers to cope with the inflexible space restrictions and submission deadlines applied to the CSIRO AUSMPC run the design was intended for.

Webber and Dunis, after experience with all of the layout tools mentioned in Chapter II as well as PLAGEN, chose to layout their design using PLAGEN and the Geometry Editor KIC. Their reasons for choosing KIC rather than other layout tools were the speed and ease of design that KIC offered[76]. The design was successfully completed by the two man design team within a six month part time design span. The design was then submitted to the JMRC November 1984 design MPC run and returned in May of 1985. Fig. 4.17 shows the plot of the design, completed using KIC, that was submitted for fabrication.

The testing of this design remains to be completed, however it is felt that the design has been simulated to such an extent with various tools that the chances of a completely functioning chip depend only on the ability of the fabrication process to yield a fault free die.

4.7 Analysis of JMRC MPC Efforts

A complementary MPC process to the CSIRO MPC process was set up in Sydney by the University of New South Wales in conjunction with the fabrication facilities at AWA at North Ryde in Sydney, as part of an Australian Government funded Joint Microelectronics Research Center (JMRC) of Excellence. Similar MPC services to those offered by the CSIRO were provded by the JMRC team including the Fabrication Process testing at an affordable price.

The main difference between the two services was that the CSIRO offered the ability to fabricate designs using a $\lambda = 2\mu$m process towards the end of the VLSI program's operation, and research (as opposed to MPC processes provided) was done

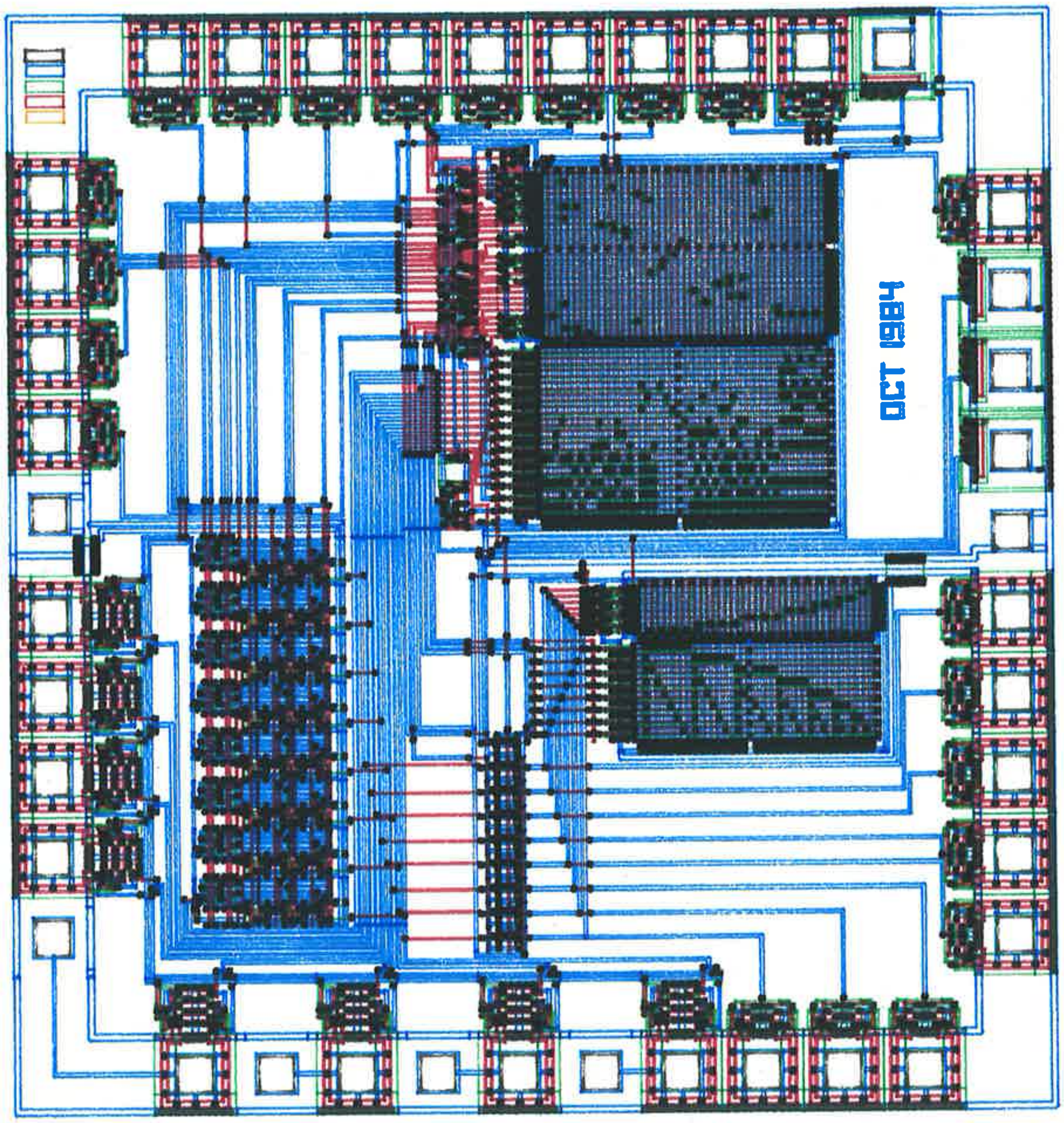


Figure 4.17: The completed version of the control unit of the data path chip designed with KIC and fabricated by JMRC.

by the CSIRO on a $\lambda = 1\mu m$ process. The CSIRO was able to do this because they used fabrication facilities in the US and therefore had the opportunity to pay for the use of some of the most advanced equipment available for the fabrication of IC's. On the other hand, JMRC were restricted to the use of the equipment purchased by AWA for its fabrication line. As a result of the small Australian demand for high quality IC fabrication equipment and the high cost of this type of equipment, the minimum feature size offered by the JMRC MPC facility is $5\mu m$ or $\lambda = 2.5\mu m$. While this restriction can be critical to circuits of the VLSI dimension, it still allows a growing Australian IC design and fabrication industry to develop. AWA can produce smaller scale designs that are commercially viable and at the same time open its doors to the Australian electronics research community through the JMRC medium. This enables wider use of this important facility for research that could lead to designs of a much larger scale, in turn providing the potential financial impetus for an Australian company to invest in much higher quality fabrication facilities than those currently existing in this country.

The advantage that the JMRC program has over the now completed CSIRO MPC program, is that JMRC has the potential to provide CMOS as well as nMOS MPC's. Progress is being made in Australian research organisations towards the implementation of a CMOS MPC process using AWA CMOS fabrication facilities to produce designs resulting from the tools discussed in Chapter III. A sample CMOS design is discussed in the following Chapter of this thesis.

Another very interesting difference between the two programs was the operating systems used on the computing equipment running the design tools. The JMRC was heavily committed to the use of UNIX as opposed to CSIRO which was just as heavily committed to the use of VMS. The reasons for these different committments can be speculated upon but are not as important as the results of the committments.

Whatever the reason for the different commitments, the result has seen a variety of tools developed for both operating systems within Australia. Those described within this thesis are primarily VMS oriented but in most cases work on both operating systems, largely due to the significant UNIX to VMS porting effort by researchers at the University of Adelaide.

The JMRC deadlines for submission and delivery are much more flexible than those used by the CSIRO. The benefits of this are that the designers have much more flexibility in altering the final design floorplan and so are not restricted as much by mistakes made in the early stages of the design process. The disadvantages of the more flexible deadlines are that designs can be late in returning to the MPC participants in the order of a few months. In a university research environment, this unexpected delay can be tolerated, it may not be tolerated in the stricter commercial environment.

4.8 **Conclusions**

The work discussed in this Chapter provides a detailed account of nMOS MPC design experiences using the tools discussed in Chapter II, and one of the tools detailed in Chapter III. The author's work involved the development of some of these tools, including KIC and PLAGEN, on the University of Adelaide computing systems as well as the design of various nMOS structures using these tools, the Signature Analyser detailed in this Chapter being one example. As well as design tool use and development, the author makes comments on the MPC process from the point of view of the design coordinator with experience of submitting designs to the JMRC process.

The tools at the University of Adelaide summarised in Table 2.6 have proven to be well suited for the mask level design of nMOS IC's. They provide functioning circuits in minimal design time. A Geometry Editor such as KIC is the most desirable for laying outcircuits which are space and electrical timing efficient, while PLA's are the quickest method of laying out any circuit although the resultant circuit will be inefficient with respect to silicon area used and will be much slower than a circuit carefully created using a Geometry Editor.

The verification tools developed at Adelaide University in conjunction with ISD, including the design rule checker CHECK, the circuit extractor NET and the simulator

PROBE, have completed the initially deficient tool set that was used in the early days of the first CSIRO AUSMPC's and the addition of the mask level Geometry Editor KIC to the tool set has provided the most efficient method of mask layout.

The Mask Level CAD tool set is complete for the fabrication of nMOS or CMOS integrated circuits at Adelaide University, even though only nMOS MPC processes have been used to date to prove this. Designs can now currently be fabricated reliably using the JMRC MPC 5 micron process using single layer metal. The only remaining barriers to IC prototyping at Adelaide University using this process are cost and invention.

CHAPTER V

CMOS DESIGN USING SYMBOLIC LEVEL CAD TOOLS

5.1 Introduction

After initial experiences with custom nMOS design, fabrication and CAD tool development using various MPC processes, and the final completion of a working nMOS CAD tool set as described in Chapters II, III, and IV, efforts were begun at Adelaide University to assist in the development of a higher level CAD tool set for custom VLSI design capable of providing more efficient designs for nMOS and, more importantly, for CMOS circuits.

The reasons for the change in technology base from nMOS to CMOS are summarised by Weste and Eshraghian[48] but can be most concisely expressed by saying that CMOS offers better performance at lower power and also scales extremely well to small feature size. Their work overcomes earlier barriers to MPC work with CMOS by adopting similar techniques to those used by Mead and Conway[8] with nMOS designs. Appendix G details the design rules specified by Weste and Eshraghian for use in a CMOS MPC process.

The VIVID CAD tool set described in Chapter III will be used to present a CMOS design experience as a means of verifying the ability of the tools to both fabricate a working CMOS design and to do it much more efficiently than would be possible using the mask level design tools discussed in Chapter II. The results of this work will be presented in CIF in a form that could ultimately be fabricated on a CMOS MPC process using the rules described in Appendix G.

A research project was commenced at Adelaide University in 1983 combining efforts from a signal processing group and an IC design group. The object of the research was to produce a new CMOS signal processing architecture capable of performing such tasks as Correlation, Lattice Filtering, and Fast Fourier Transforms[86][87]. The ***Transform and Filter Brick*** (**TFB**) is intended to comprise four ALU's, four Data Memories, an Input and an Output Processor, all connected by a Ring Bus structure under the control of a Stored Program Controller. These structures would be placed on a single CMOS chip, estimated to contain more than 200,000 transistors, to form a highly interconnected Parallel Arithmetic Processor for multi-purpose Signal Processing applications. Fig. 5.1 shows the proposed physical architecture of TFB.

Previously most computational intensive signal processing tasks necessitated the storage of digitised data and off-line processing on a general purpose computer. In many cases, on-line real time processing is required which calls for very fast data manipulation. Such signal processing tasks are intended to be completed by TFB. To enable completion of these tasks TFB will comprise a multiplier, a summer, a register for storage, a delay element and a divider[88].

A VLSI design problem such as TFB requires collaborative design team effort as well as the use of sophisticated IC design tools. The problem must be partitioned into smaller sections enabling individual designer attention. As part of the investigation into solutions to TFB sub-circuit design problems, this chapter looks at an adder structure.

The design requirements of TFB regarding design area, speed of operation and data transfer rates have prompted the need to investigate a number of different adder structures with respect to these important features and others such as their ease of adaption for multiplication and their suitability for implementation in silicon as a part of a larger system. The investigation is to be carried out using the design rules given by Weste and Eshraghian[48] and will be fabricated for testing. One of these adder structures will be presented in detail in this chapter as an illustration on the use of the higher level CAD tools.

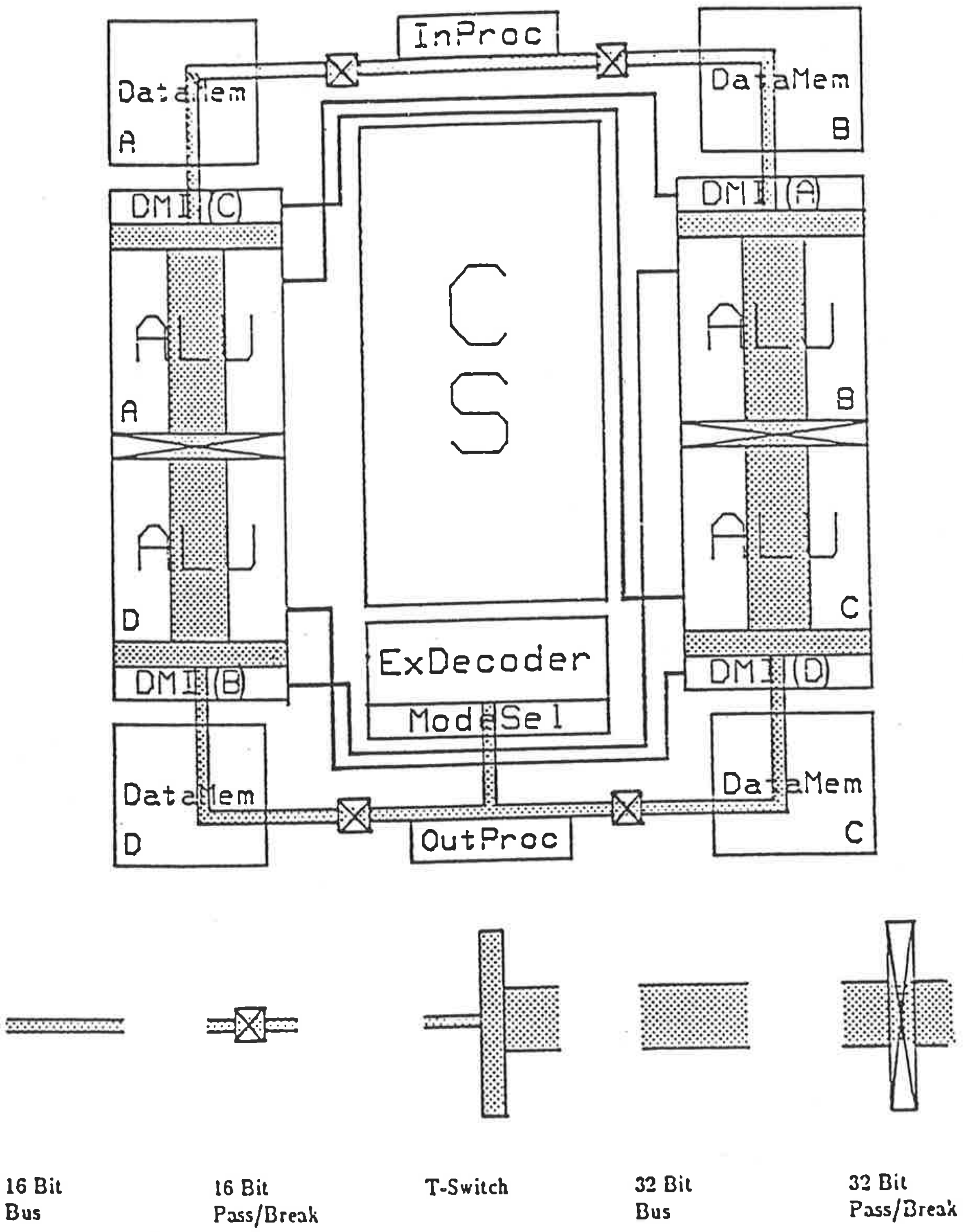


Figure 5.1: The Physical Architecture of the Transform and Filter Brick, TFB.

5.2 A CMOS Adder Design

The combinatorial adder referred to in Weste and Eshraghian[48] is the structure to be studied in detail. The simplest approach to designing an adder is to use standard logic gates to implement the Boolean equations defining addition of binary digits. The Boolean equations defining the addition operation for a ripple-through adder are expressed by equations 5.1 and 5.2.

$$SUM_{OUT} = A.B.CARRY_{IN} + A.\overline{B}.\overline{CARRY_{IN}} + \overline{A}.\overline{B}.CARRY_{IN}$$

$$+\overline{A}.B.\overline{CARRY_{IN}} \dots\dots (5.1)$$

$$CARRY_{OUT} = A.B + A.CARRY_{IN} + B.CARRY_{IN}$$

$$= A.B + CARRY_{IN}.(A+B) \dots\dots (5.2)$$

The variables used in the equations are A and B, the two input bits to be added together, and $CARRY_{IN}$ and $CARRY_{OUT}$, the previously generated carry and the current generated carry bits respectively. The gate schematic of the adder can be seen in Fig. 5.2.

The design of one bit of the combinatorial adder is shown in floorplan form in Fig. 5.3. This floorplan form of the design is available as a standard hierarchical view of the design using the VIVID system. Fig. 5.2 provides the guidelines for the symbolic layout of the transistor schematic which in turn can be compacted to a very efficient mask layout form using systematic symbolic layout techniques.

A technique for efficient layout of CMOS designs is the compounding of individual gates to implement a particular design, as described by Weste and Eshraghian[48]. An advantage is gained by reducing the number of transistors required to implement a Boolean function using this method rather than composing a number of individual gates together to perform the same function. The main disadvantage of compounding gates in this manner is that the circuit may be slower than other implementations.

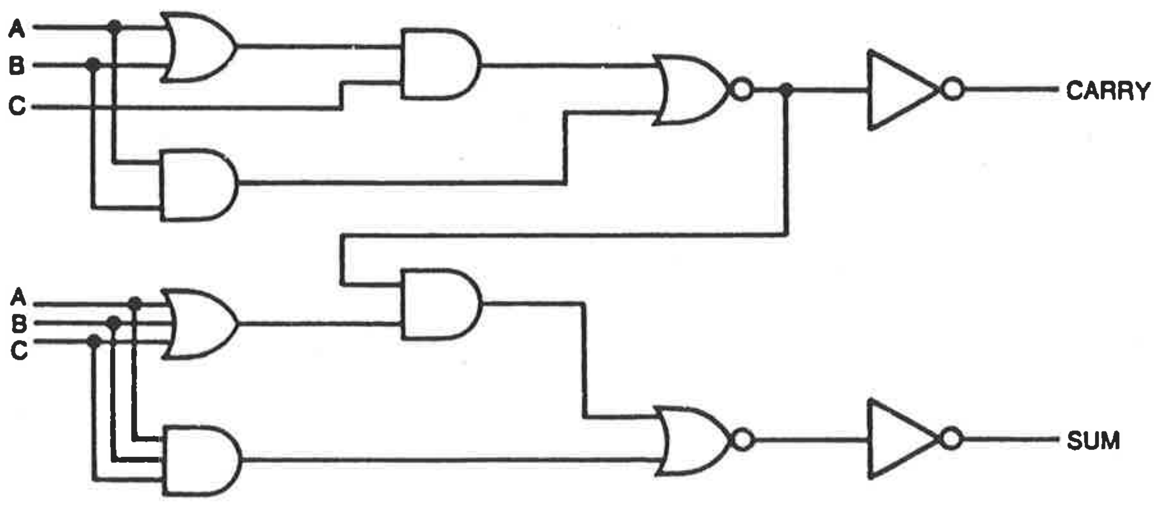


Figure 5.2: The logic gate schematic of the combinatorial adder.

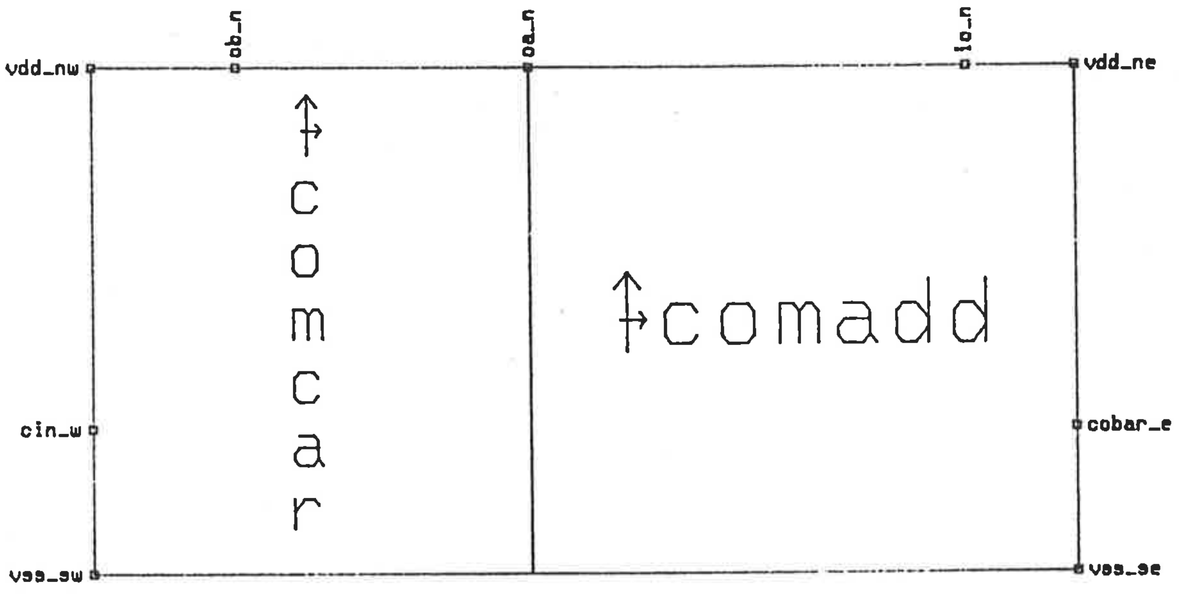


Figure 5.3: The Floorplan of one bit of the combinatorial adder.

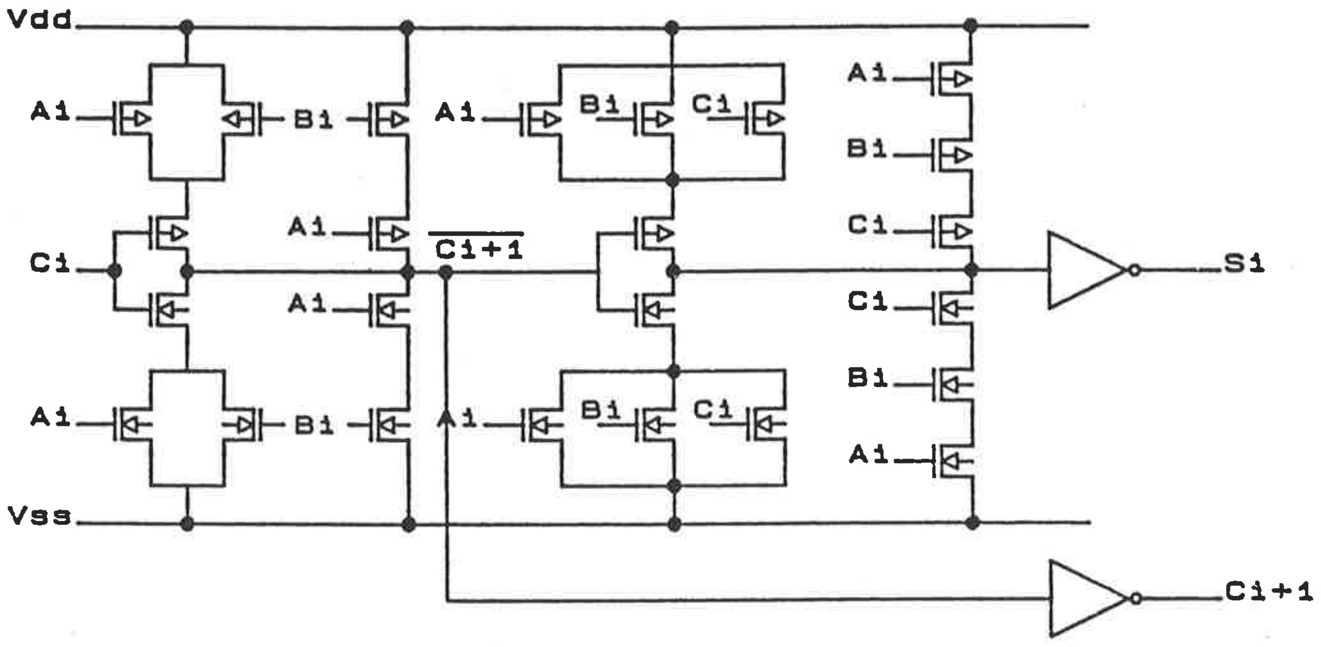


Figure 5.4: The transistor schematic of the combinatorial adder.

This disadvantage can be overcome to some extent by varying the p and n type ratios of the circuitry to speed up critical paths.

The one bit combinatorial adder is shown in transistor schematic form with its component parts identified in Fig. 5.4. This particular adder design uses the inverted carry output ($\overline{CARRY_{OUT}}$) in the generation of the sum output SUM. The equation used for this sum output generation is

$$\overline{SUM_{OUT}} = \overline{\overline{CARRY_{OUT}}.(A + B + CARRY_{IN}) + A.B.C} \qquad (5.3)$$

Equation 5.3 can be shown to be the same as the complement of equation 5.1 (see Appendix H) but is more efficient in terms of gate complexity for the description of the sum generation circuitry and is consequently the equation used to describe the sum generation circuit in Fig. 5.4. Because the sum output depends on the carry output, there will be a delay in sum output generation with respect to the carry output generation.

The carry output shown in Fig. 5.4 as $\overline{C_{i+1}}$, is generated according to the complement of equation 5.2, resulting in equation 5.4.

$$\overline{CARRY_{OUT}} = \overline{A.B + CARRY_{IN}.(A+B)} \quad \ldots\ldots (5.4)$$

The two stages of a one bit adder, the carry and sum generation circuits, can be cascaded n times to form an n-bit parallel adder. The carry output will need to ripple through the n stages to provide the final n-bit result. A schematic of a ripple carry adder can be seen in Fig. 5.5 (a). The final result of the n-bit addition will be delayed in the worst case by $T_D = nT_C$ where T_C is the delay in the generation of the carry output in one bit of the circuit (ie. the time taken to generate the output from the carry generation circuit shown in Fig. 5.4).

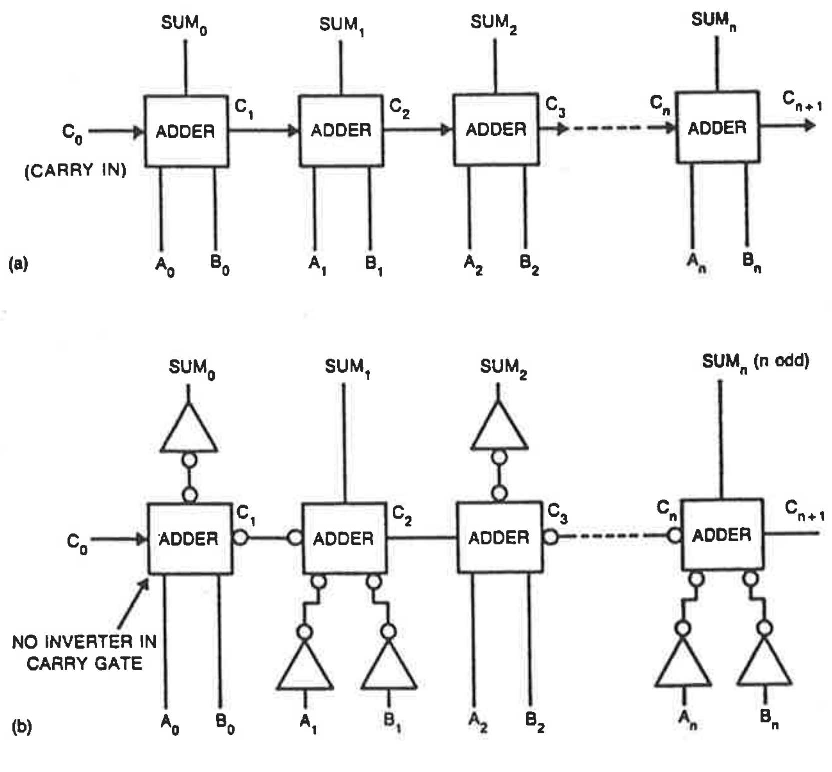


Figure 5.5: (a) Above can be seen the schematic diagram of a cascaded ripple carry adder. (b) Below is the schematic of the same adder using complemented outputs and inputs to speed up the critical carry propagation path.

Using this type of design to obtain the greatest speed of addition, the aim must therefore be to reduce T_C to its lowest value. This requires the optimum design with respect to speed of generation of the carry output in Fig. 5.4. The first optimisation is to generate only the complemented form of the carry out as the signal to be transferred between blocks. Using the complemented forms of both the sum and carry outputs (as shown in equations 5.3 and 5.4) and cascading to form the final n-bit adder while applying alternately complemented and non-complemented inputs, results in alternate non-complemented and complemented outputs as shown in the schematic diagram in Fig. 5.5 (b).

Using the complemented outputs for sum and carry generation results in the reduction of T_C by T_I, where T_I is the delay through the inverter used to produce the non-complemented form of $CARRY_{OUT}$. As the carry path is the critical path for speed of operation of a ripple carry parallel adder, this reduction of T_C accumulates with each stage of a parallel adder. The delays due to inversion of inputs and outputs for the individual adder stages can be made negligible in comparison to T_D depending upon the specifications of a particular adder design.

Appendix H shows the equivalence of logic equations implemented in the circuit in Fig. 5.4 with the inputs in both complemented and uncomplemented form and their respective results. Because of the use of alternate complemented/uncomplemented inputs, it becomes necessary to distinguish odd numbered stages from even numbered stages of the cascaded adder circuit. Therefore, the design of the adder has been built in two bit form as this is the fundamental block with which a parallel adder of this type can be constructed without any low level symbolic manipulation other than routing.

This fundamental two bit block can be seen in Fig. 5.6 in floorplan form, subdivided into its component parts. The odd and even one bit versions of the adder consist of the two component cells shown in Fig. 5.8. The multiplexer cells shown in Fig. 5.10 form the component sections of the outputs for the individual odd and even numbered stages of the addition circuitry.

An odd numbered multiplexer stage uses uncomplemented inputs to produce complemented outputs for the sum and carry result. The even numbered multiplexer stage produces complemented inputs for the sum, and the sum generation circuit uses the complemented carry from a previous adder stage, resulting in uncomplemented outputs.

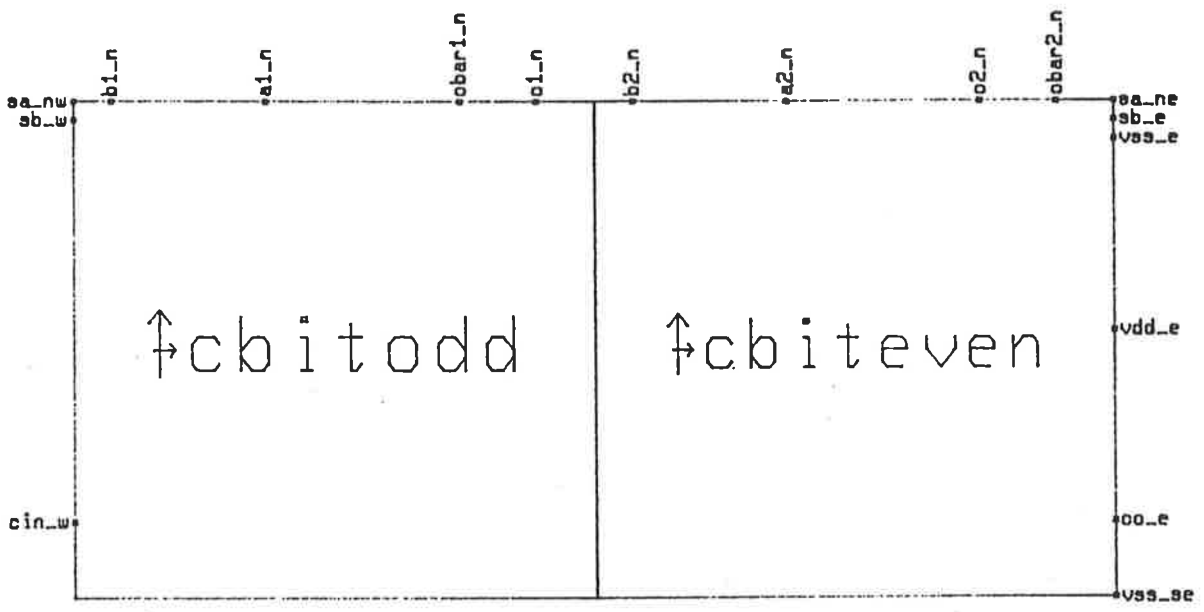


Figure 5.6: The Floorplan of the two bit building block used for cascading the ripple carry adder stages.

This adder structure can be used without significant modification for the operations $A - B$ and $B - A$ as well as $A + B$. The inputs to the sum and carry generation circuitry can come either through the inverters or straight through the multiplexer unaltered, as determined by the inputs s_a and s_b.

These two inputs are used to determine the operation of the adder. If both $s_a = 1$ and $s_b = 1$ then the operation performed will be $A + B$. If $s_a = 0$ and $s_b = 1$ the operation performed will be $B - A$, while if $s_b = 0$ and $s_a = 1$ the operation performed will be $A - B$. The condition $s_a = 0$ and $s_b = 0$ is a prohibited condition for this design. For the operations $A - B$ and $B - A$ to be completed, the carry input to the first stage needs also to be altered from 0 as it should be for $A+B$, to 1. This task is completed by attaching a carry in generation circuit as shown in Fig. 5.11 to the first stage of the cascaded parallel ripple through adder.

Once the individual symbolic leaf cell functions have been specified at the floorplan level, the detailed layout of the design can be commenced. Using the VIVID design tools, this can be done using either the ABCD language, or more efficiently using the symbolic graphic editing tool ICE.

5.3 Symbolic Layout of the Combinatorial Adder

The first task is to lay out the carry and sum generation circuits in their most efficient form. Using equations 5.3 and 5.4 as the guides for the compounding process, the transistor schematic shown in Fig. 5.4 results. Using gate compounding techniques[48] the results of the carry generation stage can be seen in Fig. 5.7 (a). Using the uninverted form of the LHS of equation 5.4, $A.B + CARRY_{IN}.(A+B)$ the n side is first constructed. The logical AND expression $A.B$ and the OR expression $A+B$ are shown constructed in switch form first. Then the complete expression implemented below.

Taking the p-side requires the complement of the expression to be used. The LHS of equation 5.4 then reduces to $(\overline{A}+\overline{B}).(\overline{CARRY_{IN}}+\overline{A}.\overline{B}) = \overline{CARRY_{IN}}.(\overline{A}+\overline{B})+\overline{A}.\overline{B}$. Then AND and OR terms may once again be constructed in switch form to provide the results shown in Fig. 5.7 (b). The same techniques applied to the LHS of equation 5.3 results in the transistor schematic for the sum generation circuit shown in Fig. 5.4.

This transistor schematic must be optimised to provide fastest speed of addition and smallest silicon area before the final symbolic layout form can be completed. Initially the design may be layed out using single devices in the positions indicated by the transistor schematic. Once this is done, simulations can be run to check the speed of operation of the first layout and successive refinements and simulations made until the optimised circuit is resolved.

Considering the combinatorial adder shown in Fig. 5.4 and the boolean equations

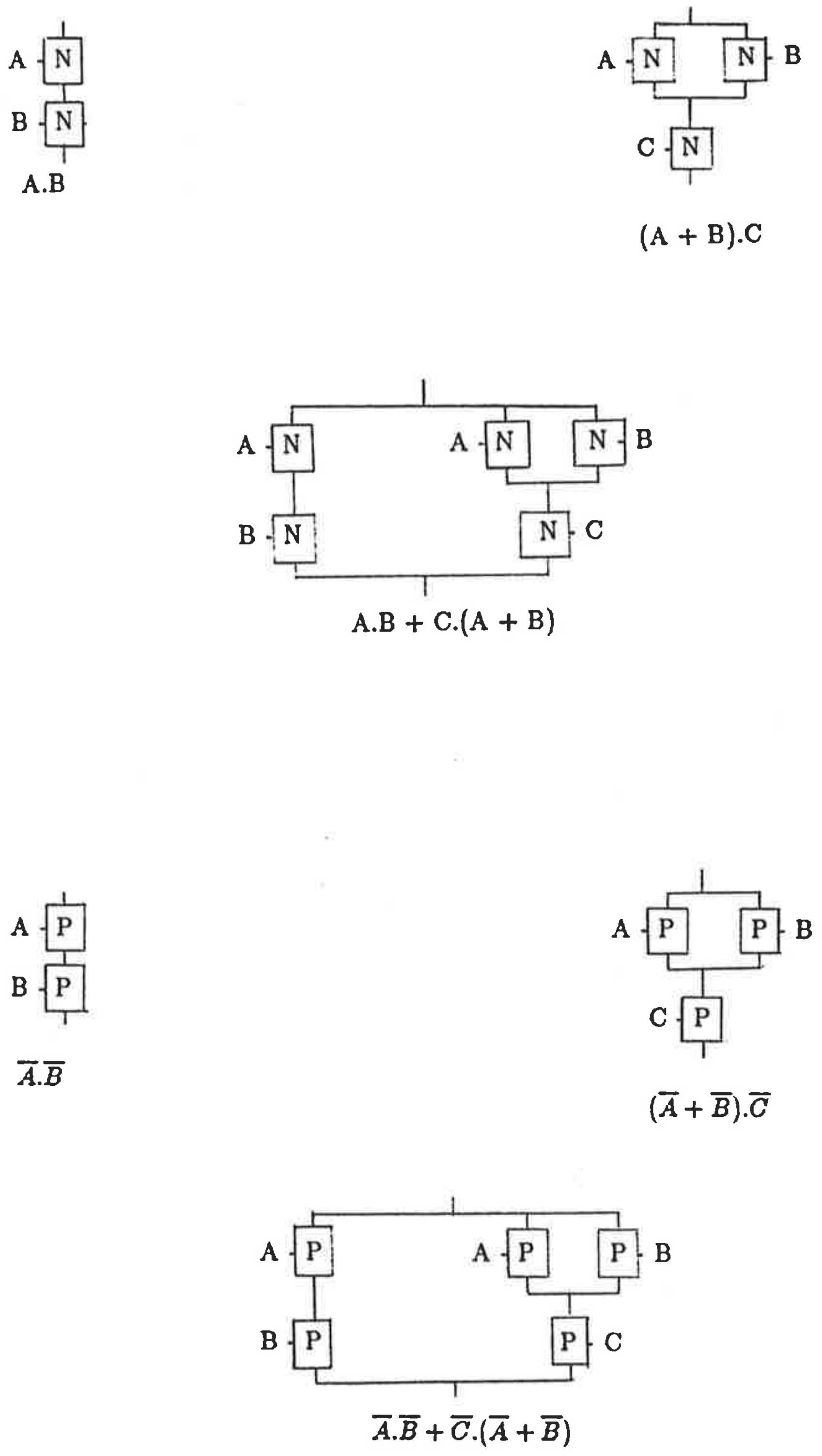

A
N
B
N
A.B
A
N
N
B
C
N
(A + B).C
A
N
A
N
N
B
B
N
N
C
A.B + C.(A + B)
A
P
B
P
$\overline{A}.\overline{B}$
A
P
P
B
C
P
$(\overline{A}+\overline{B}).\overline{C}$
A
P
A
P
P
B
B
P
P
C
$\overline{A}.\overline{B}+\overline{C}.(\overline{A}+\overline{B})$

Figure 5.7: (a) Above can be seen the process involved in the construction of the n-type transistor circuit for the function described in equation 5.4. (b) Below is the construction of the p-type transistor circuit for the same function.

(equations 5.3 and 5.4) that this transistor schematic is meant to describe, it can be seen that the positioning of the devices has been optimised.

The circuit schematics shown in Fig. 5.7 could have alternative arrangements, however three other important considerations dictate the layout shown. First, the transistors switched by the $CARRY_{IN}$ signal should be close to the output, minimising the influence due to body effect of these transistors.

Second, all transistors in the sum generation circuit whose gates are connected to $\overline{CARRY_{OUT}}$ should be made of minimum size to minimise the capacitive load on the signal output. This loading principle also dictates the positioning of the carry generation circuit with respect to the sum generation circuit, as routing lengths should be minimised for lines connected to $\overline{CARRY_{OUT}}$. When routing, the most efficient layers (metal and poly) should be used.

Third, the sizing of series transistors should subsequently be determined by simulation. This is an iterative process made possible only by the use of a quick response simulator such as FACTS, described in Chapter III. Simulators such as SPICE would take too long to make this an effective design procedure, while simulators of the MOSSIM type are not designed to provide the necessary timing information. A simulator such as PROBE would provide the timing information although only after compaction of the design, and extracton at the mask level, again making the design iteration a more tedious step than using FACTS. The symbolic layout results of such design iterations using FACTS for both the sum and carry generation circuits are shown in Fig. 5.8.

These layouts alone do not allow the cascading required to produce the ripple carry parallel adder. Inverters must be added to the inputs and outputs in an appropriate form to provide the alternative complement/non-complement inputs and outputs required by alternating bits of such a cascaded structure. Allowing for efficient cascading of a single block to produce an n-bit ripple carry parallel adder, the easiest solution is to produce a 2-bit version of the adder. The two bit version will provide consistent

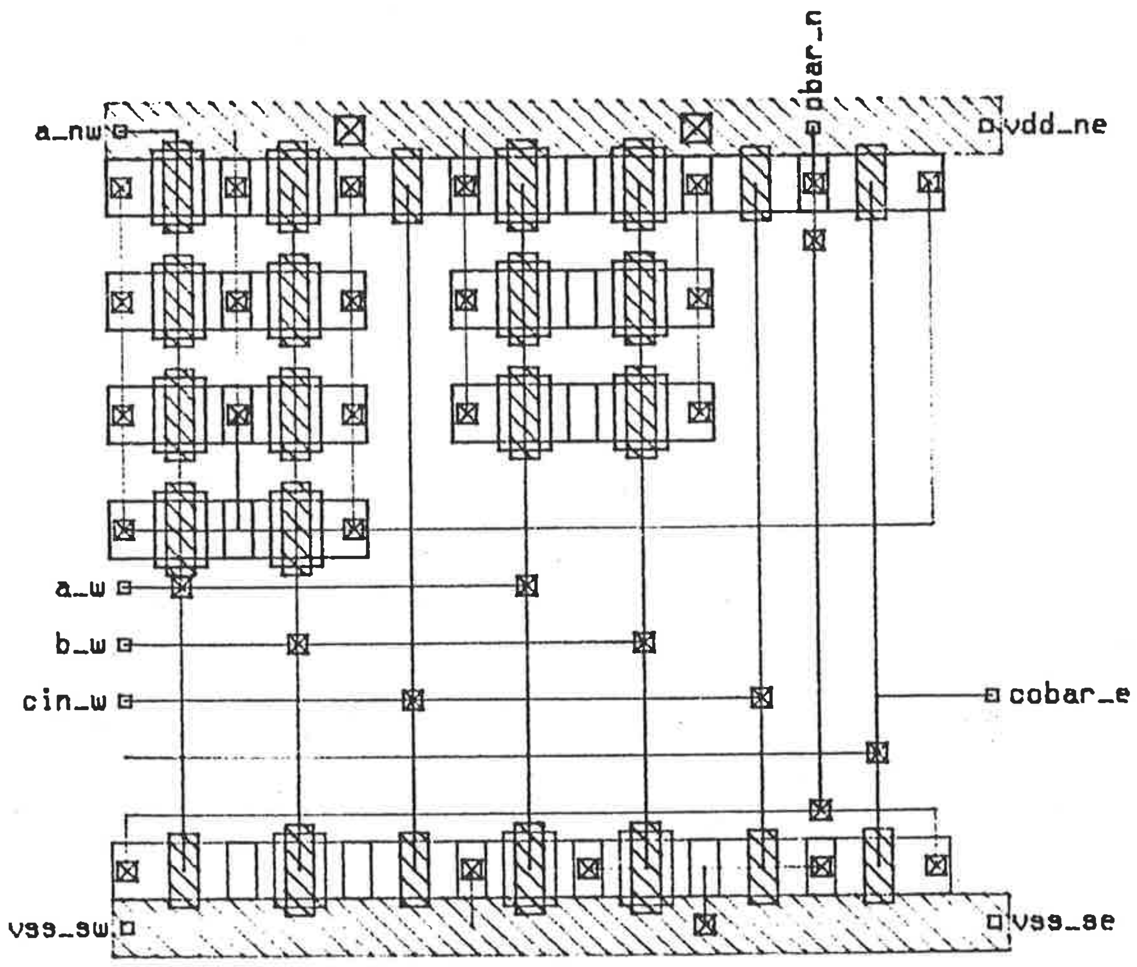


Figure 5.8 (a): The symbolic layout of the sum generation stage of the combinational adder.

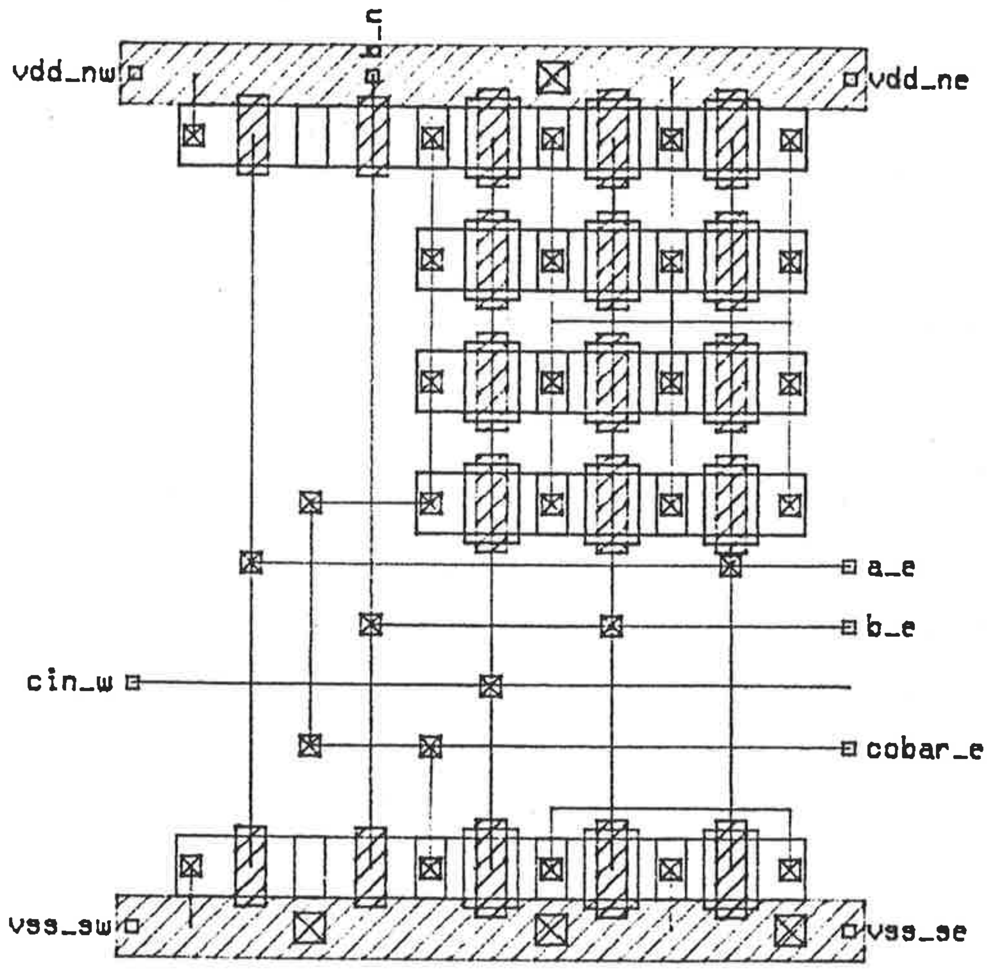


Figure 5.8 (b): The symbolic layout of the carry generation stage of the combinational adder.

inputs and outputs (no complementing required) at all cascade points. It does this by containing within the 2-bit block an odd and an even addition stage where the inputs to the adder are controlled by two lines s_a and s_b. This control function allows the adder to complete addition or subtraction of the inputs.

The circuits used to multiplex the inputs to the odd and even numbered bits of the 2-bit adder block can be seen in logic layout form in Fig. 5.9. These circuits, (a) and (b), are different only in the configuration of the transmission gates used to multiplex the input signals to their respective adder inputs. It can be clearly seen from the schematic that the state of s_a and s_b dictate which of the three operations $\boldsymbol{A+B}$, $\boldsymbol{A-B}$, or $\boldsymbol{B-A}$, will be performed by the 2-bit adder block. (Note that $s_a = 0$ and $s_b = 0$ is disallowed). The outputs on the other hand are hard wired to provide both the uncomplemented and complemented forms of the adder output. The results of the symbolic layout of the circuits schematically described in Fig. 5.9 are shown in Fig. 5.10.

To complete an adder using cascaded 2-bit blocks as described in this section, an input block for the the first $\boldsymbol{CARRY_{IN}}$ must be completed. Assuming that the first inputs are to be un-complemented, the control signals s_a and s_b need to be decoded to generate a $\boldsymbol{CARRY_{IN}}$ for the first stage such that if s_a and s_b are both 1, then $\boldsymbol{CARRY_{IN}} = 0$ otherwise $\boldsymbol{CARRY_{IN}} = 1$. The symbolic layout for this carry in generation circuit is shown in Fig. 5.11.

The symbolic layout for each of these designs was completed using the tool ICE. In the time it took to complete the layout for the Signature Analyser design, this design was totally layed out and simulated using FACTS to verify not only the design operation, but to give details of timing using preliminary models for estimated CMOS processes. The symbolic layout of a design on a virtual grid provides great flexibility in the modification of layouts in conjunction with fast simulations to provide the fastest, most compact layout of a design.

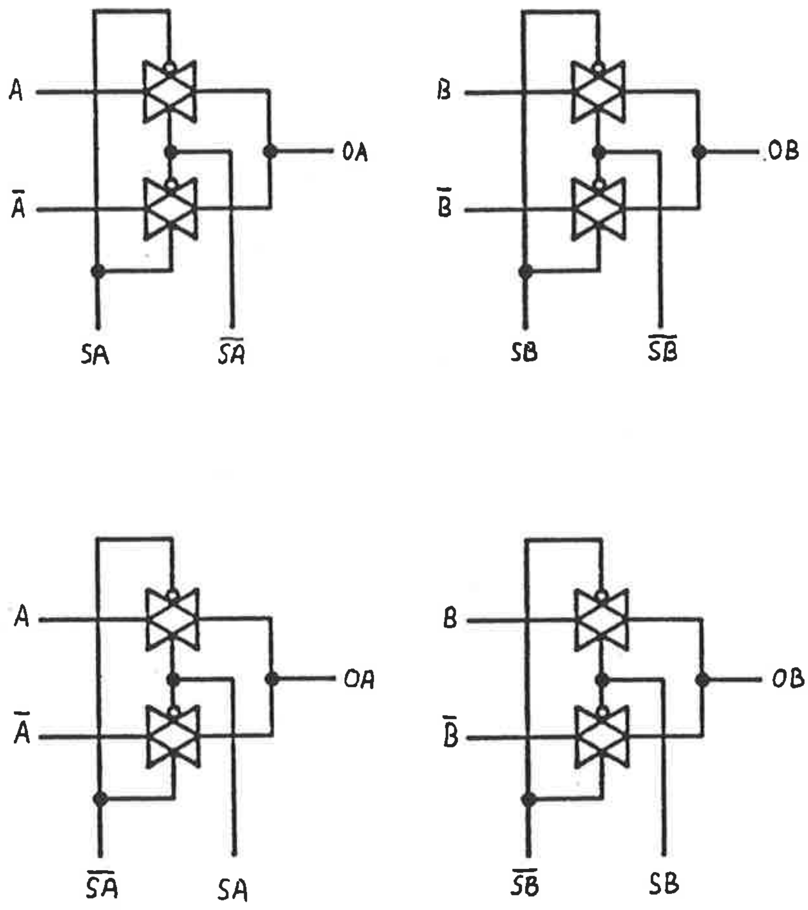


Figure 5.9: (a) Above the logic diagram describing the odd numbered multiplexer for the combinational adder. (b) Below, the even numbered multiplexer logic diagram.

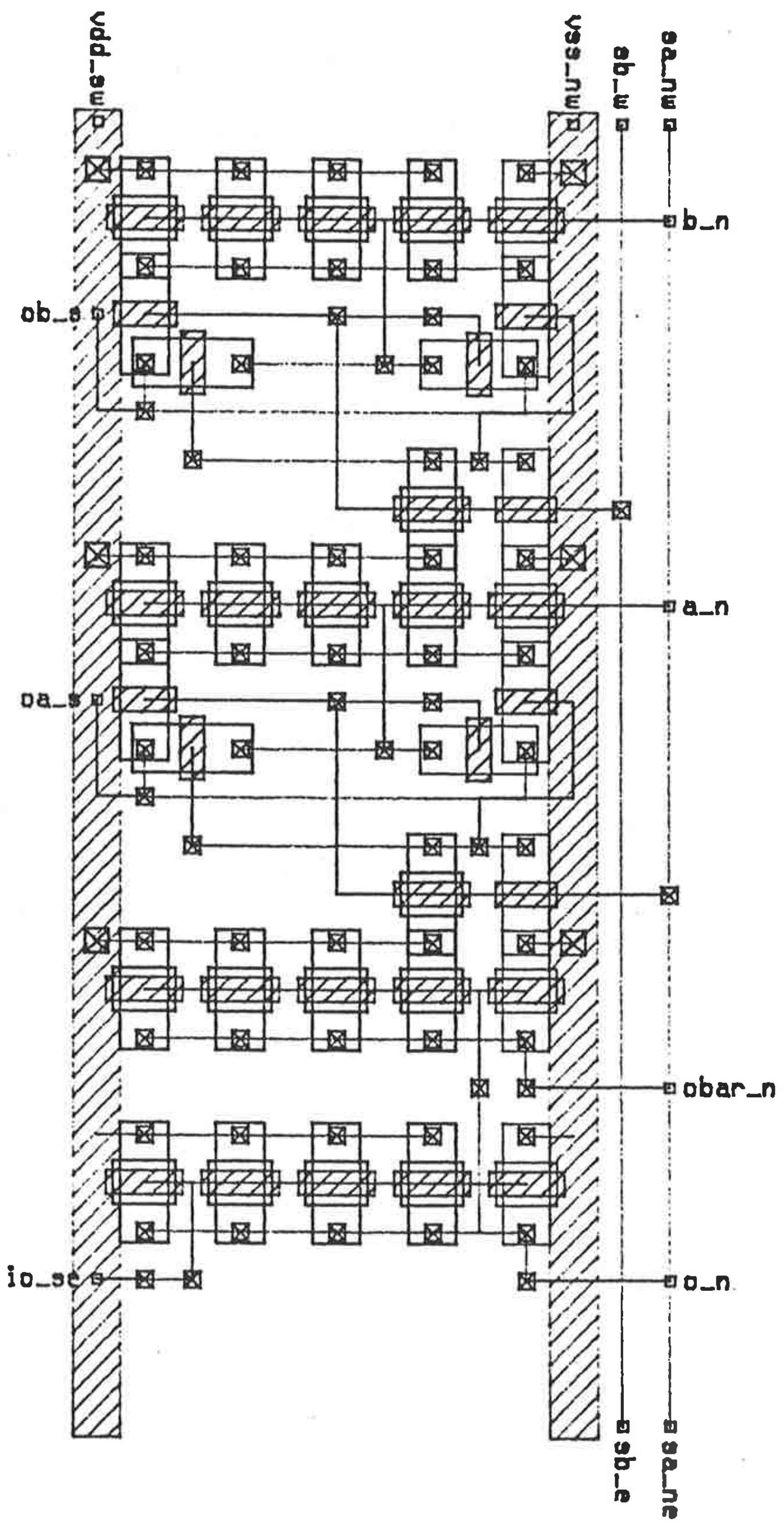


Figure 5.10 (a): The symbolic layout of the odd numbered multiplexer.

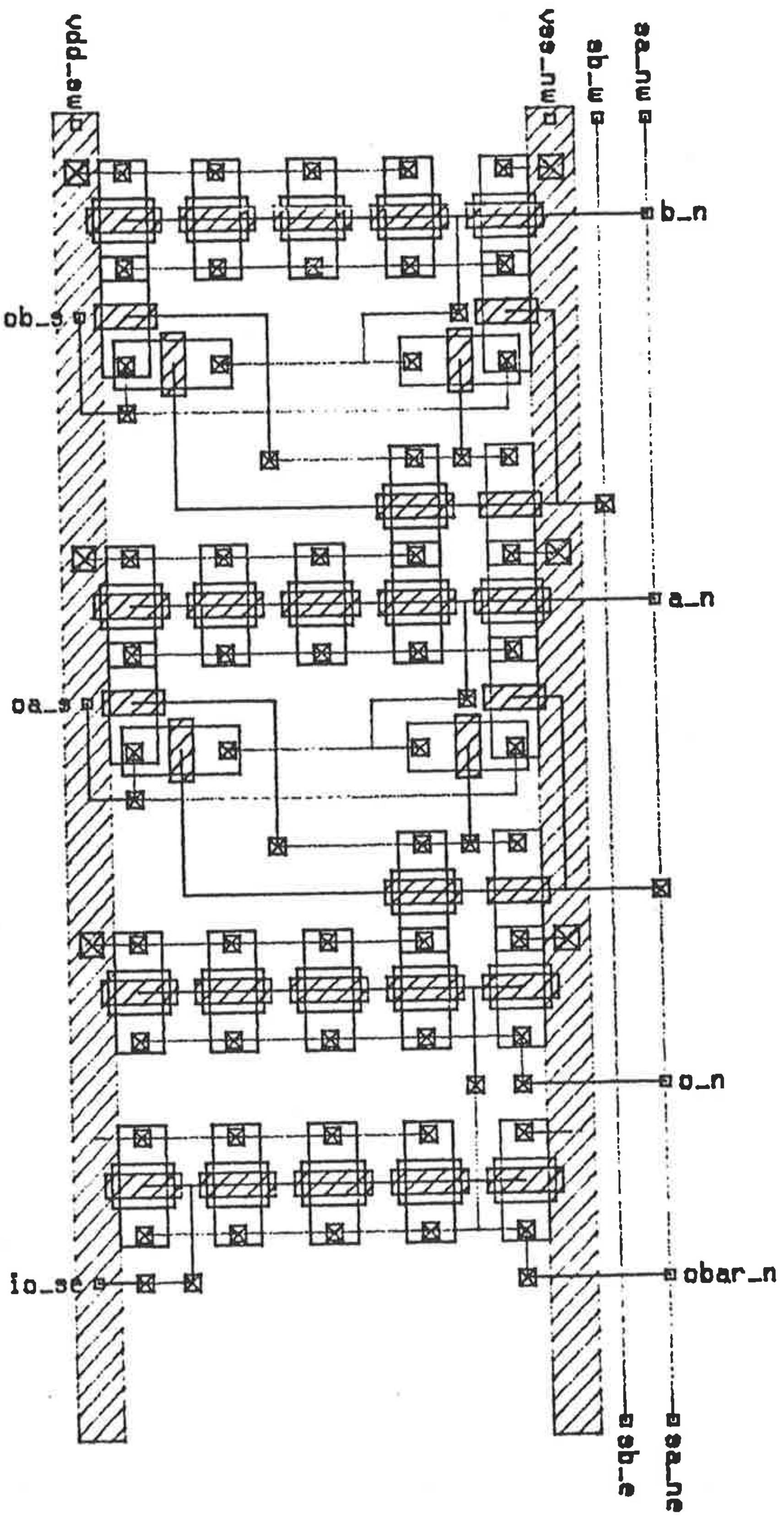


Figure 5.10 (b): The symbolic layout of the even numbered multiplexer.

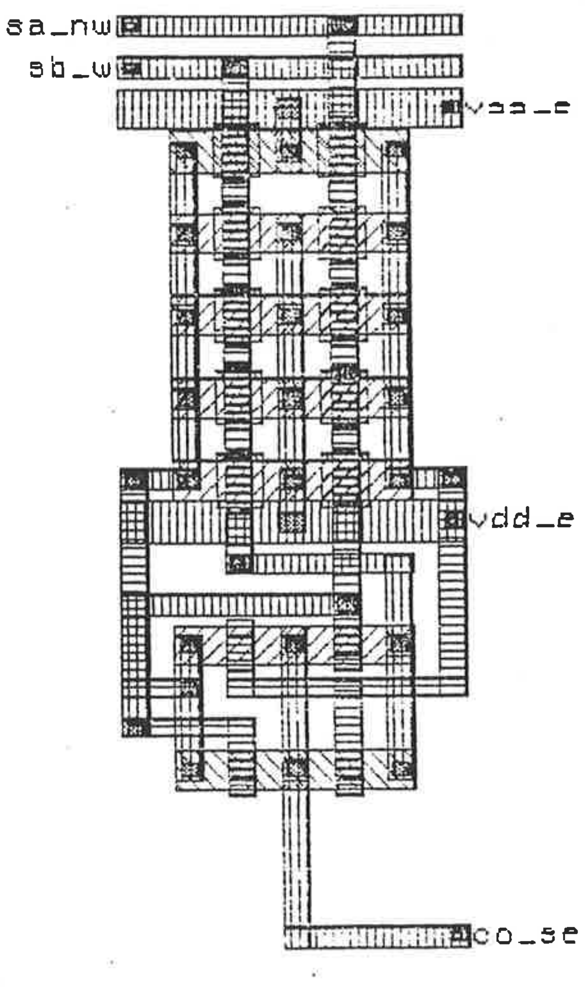


Figure 5.11: The symbolic layout of the carry in generation circuit.

The biggest advantage found in using ICE to lay out the design was that errors were made less frequently because (i) circuit elements (p and n type transistors, cuts and wires) were being manipulated rather than geometries and (ii) the layout was graphically edited rather than textually edited. The first advantage is the reason that symbolic level layout tools are preferrable to mask level layout tools and ICE therefore preferable to KIC.

5.4 High Level Extraction and Simulation

Use of the circuit extractor ABSTRACT and the simulator FACTS, enables a quick response simulation check of a design during the circuit design phase. The fast response of the extraction and simulation process is a key factor in the effectiveness of the VIVID tool set. It enables a designer to detect, almost immediately, errors made in the symbolic layout process. Combined with the layout advantages of symbolic level tools, the fact that the circuit extractor uses ABCD as input and that the simulator provides timing information using circuit models, means that a layout/simulation iteration in the design process can be executed much faster than the same iteration in the equivalent mask level design procedure. These points will be illustrated using the adder design simulation results with the VIVID tools and the results gained from using the compacted CIF version of the design with the mask level tools described in chapters II and IV.

The simulated results for each of the component parts of the adder design are presented in Fig. 5.12 (a)....(q) along with some examples of extracted circuits produced by ABSTRACT. Each component of the design has an expected result for its simulated output and these expected results are presented in tables in the following text as an analysis of the operation of the adder. In all cases the inputs and the outputs are nominated for the separate parts of the design. The inputs have signals applied to them by way of commands given from the simulator while the simulated results are returned through the outputs.

ABSTRACT uses the VIVID MTF system, which calculates the electrical parameters associated with each circuit element. These values are estimates, but are relatively accurate for all primitives except wires, which are directly dependent on the final size of the layout. For the purposes of presentation in the following sections, c_{in}, a, b, o and c_o are respectively used to denote $\boldsymbol{CARRY_{IN}}$, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{SUM_{OUT}}$ and $\boldsymbol{CARRY_{OUT}}$ as used in previous equations.

5.4.1 Simulation of the Carry Generation Stage

The inputs for the carry generation stage are c_{in}, a, and b while the output is $\overline{c_o}$. Table 5.1 presents the input patterns applied to the carry generation circuit to test its operation for the critical input conditions causing the worst possible time delay in the addition result, together with the expected result T_C shown in Fig. 5.12 (b). The extracted circuit for the carry generation layout is shown in Fig. 5.12 (a).

c_{in}	a	b	$\overline{c_o}$
1	0	1	0
0	0	1	1
0	0	1	1
1	0	1	0
1	0	1	0
0	1	0	1
0	1	0	1
1	1	0	0
1	1	0	0
0	1	0	1

Table 5.1: The input conditions causing the worst case carry generation time delay T_C. This delay is critical as it results in the longest time delay for an addition result, $T_D = nT_C$ and therefore specifies the maximum speed of operation of an n-bit adder.

The important features that can be seen in the FACTS simulated results shown in Fig. 5.12 (b) are as follows:-

(i) The expected output agrees with the simulated results.

(ii) The worst propagation time for the carry generation circuit is determined by the FACTS model to be $T_C = 7ns$. This would imply that for an n bit parallel carry ripple through adder, the worst overall propagation time could be

$T_D = nT_C = n.7ns$. This figure can be checked against the simulated results for an adder where $n = 8$ as shown in section 5.4.8 where a more accurate estimate of T_C can be obtained by looking at the propagation delay over a number of stages.

The simulation indicated in Table 5.1 examines the most critical delays for the carry generation circuit. The critical time delay occurs only when the inputs a and b are opposite. When an addition is carried out on an n-bit parallel ripple carry adder, all inputs a_n and b_n are presented to the adder at the same time. Initially, all n adder stages will produce sum and carry outputs using their immediate inputs. As the carry output from the first stage ripples through the n stages, the carry out from a particular stage i will only be altered from its initial level if the inputs a_i and b_i are opposite and the previous carry output goes through a transition.

c_{in}	a	b	s_o	c_o
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 5.2: The Truth Table for Addition. Note that c_o is only affected by c_{in} for the cases where $a \neq b$. Whenever $a = b$, c_o is independent of c_{in}.

This can be seen by examining the above truth table, Table 5.2. The carry output for an addition operation will only make a transition from 1 to 0, or 0 to 1 if the carry input changes when $a = 0$ and $b = 1$ or when $a = 1$ and $b = 0$. The cases $a = b = 0$ and $a = b = 1$ generate their respective carry outputs at the time that the inputs are applied to the adder. These carry outputs generated at that time will not be altered by any transistion of a previous carry input to that particular stage at any subsequent time. This implies that the worst case delay for the final result of an addition from a parallel ripple carry adder would be when all a_i and b_i are opposite. The resultant delay would then be $T_D = nT_C$ as indicated previously.

```
.model nenh nmos
+ vto=0.779              phi=0.60
+ uo=4.0e-02             nsub=1.0e+22
+ xj=.0385u              tox=500.0e-10
+ cj=4.2e-4              cjsw=9e-10
+ gl=0.7
*
.model penh pmos
+ vto=-0.98              phi=0.60
+ uo=1.5e-02             nsub=8.158e+20
+ xj=.146u               tox=500.0e-10
+ cj=2.5e-4              cjsw=4.5e-10
+ gl=0.7
*
*
md5 IO a vss vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md4 vss b IO vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md12 IO cin cobar vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md1 I1 a vss vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md22 I2 cin cobar vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md20 I2 cin cobar vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md18 I2 cin cobar vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md17 vdd b I2 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md16 I2 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md15 I2 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md14 vdd b I2 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md7 vdd b I2 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md6 I2 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md2 cobar b I1 vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md8 cobar b I3 vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md10 vdd b I2 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md11 I2 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md3 I2 cin cobar vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md9 I3 a vdd vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
cb b vss 39.30f
ca a vss 42.60f
ccin cin vss 36.00f
ccobar cobar vss 43.50f
cIO IO vss 12.60f
cI2 I2 vss 33.60f
```

Figure 5.12 (a): The extracted circuit obtained using ABSTRACT for the carry generation circuit.

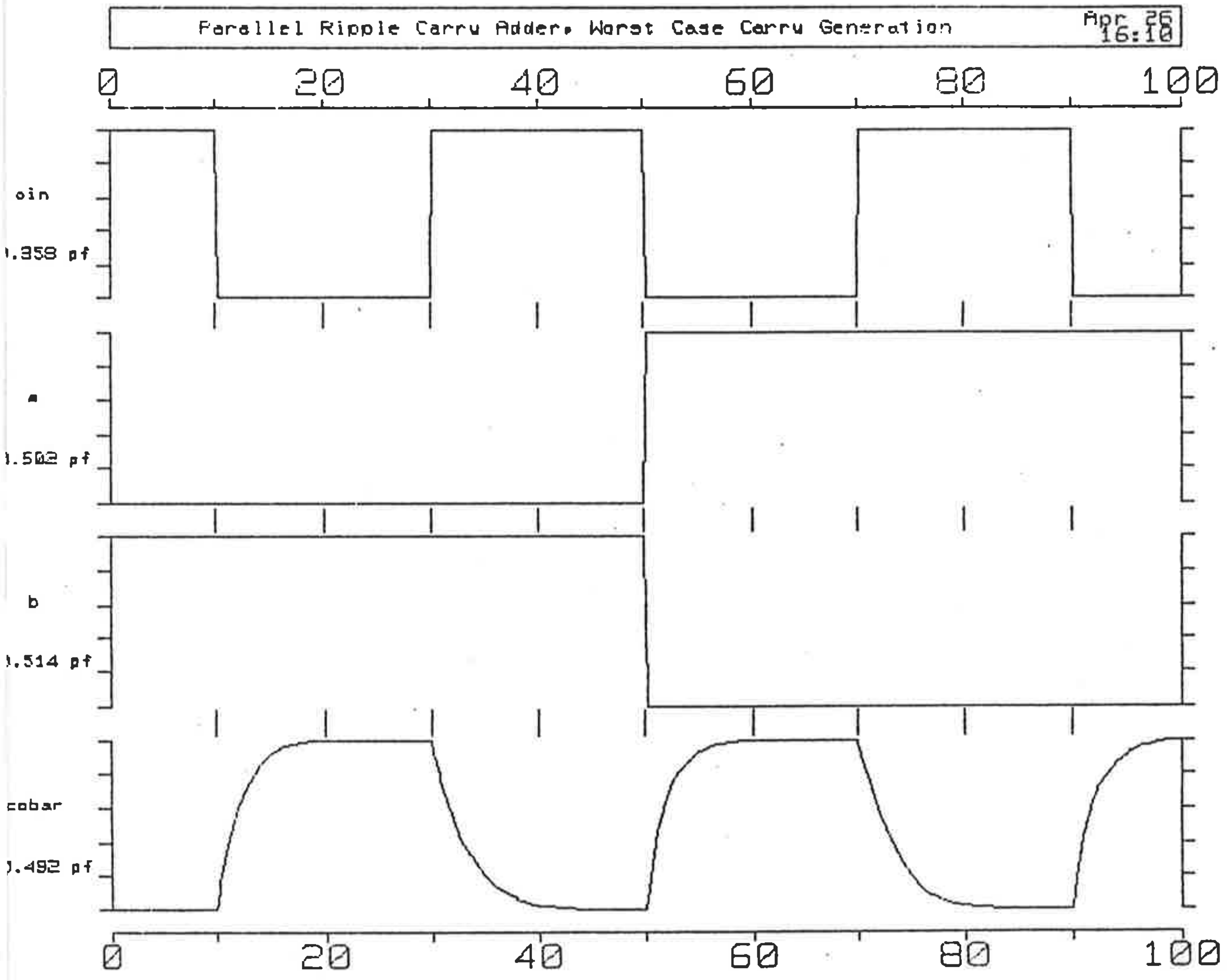


Figure 5.12 (b): The FACTS simulation of the carry generation circuit for worst case carry propagation.

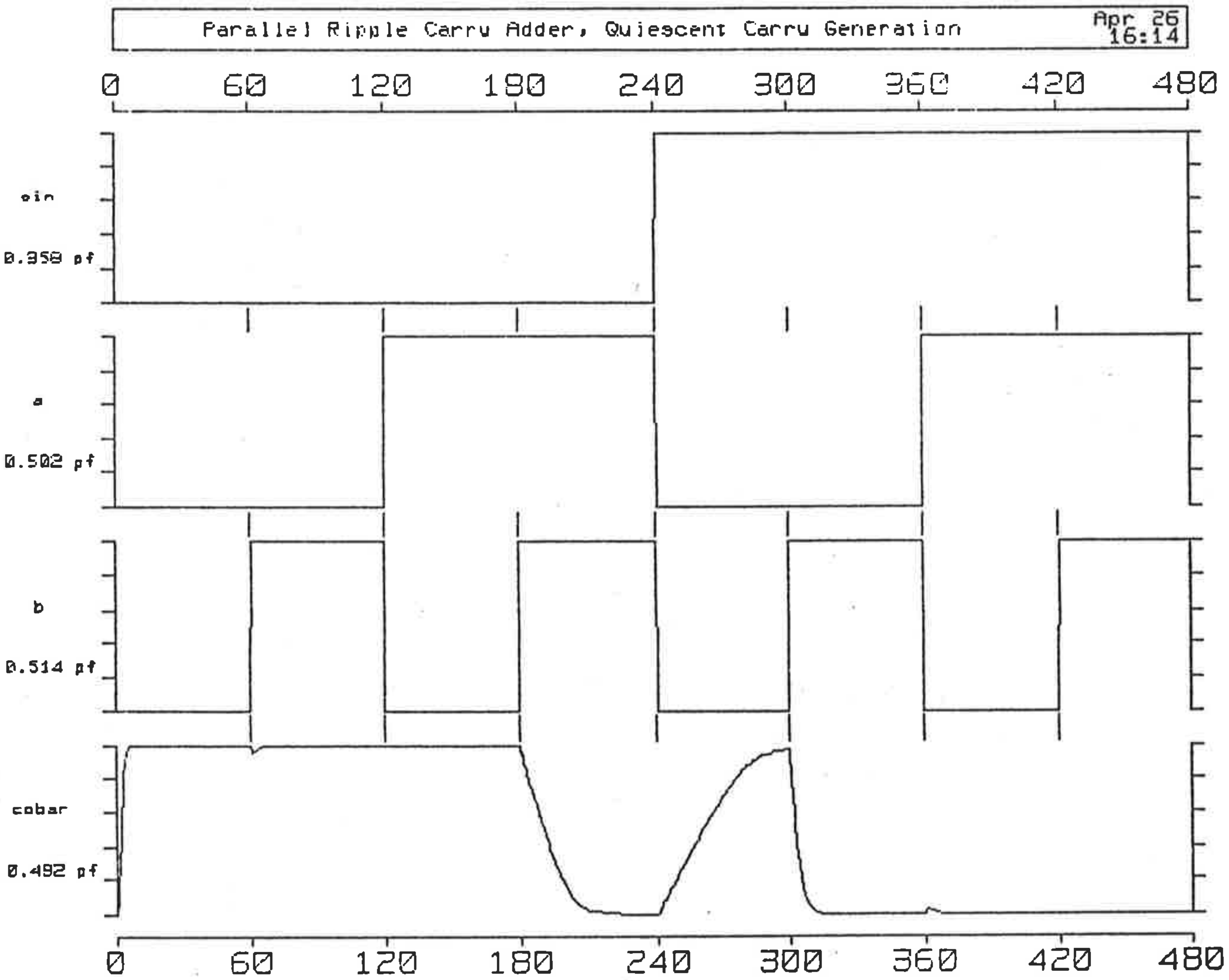


Figure 5.12 (c): The FACTS simulation of the carry generation circuit for all possible input conditions.

It is also important to note that the carry output generated when $a = b$ also has a worst case time delay associated with it, T_{QC}. This can be ignored as long as $T_{QC} \leq T_D$. Should the number of stages be such that $T_{QC} > T_D$, optimisation of the layout presented here would have to take place as the critical circuit delay would then become the offending worst case quiescent carry generation condition, which in this case can be recognised from Fig. 5.12 (c) to be when c_{in} is high, a is low, and b changes from high to low. These conditions result in a figure of $T_{QC} = 50ns$. The input patterns and results obtained in the simulation can be compared with those indicated in Table 5.3.

c_{in}	a	b	$\overline{c_o}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 5.3: The Simulation Test for the complete set of possible input conditions for the Carry Generation Circuit is defined by the above truth table where c_{in}, a and b are the inputs and $\overline{c_o}$ is the output.

5.4.2 Simulation of the Sum Generation Stage

The inputs to the sum generation stage are c_{in}, a, b and $\overline{c_o}$ while the output is $\overline{o}$. Table 5.4 presents the input patterns applied to the sum generation circuit and the expected results for all possible combinations of input conditions. The extracted circuit for the sum generation layout is given in Fig. 5.12 (d).

The important features of the simulated results as shown in Fig. 5.12 (e) are as follows:-

(i) The results of the simulation agrees with the expected results.

(ii) The worst quiescent sum generation time is determined to be $T_{QS} = 50ns$. It has not been considered critical to optimise this time delay for this particular

```
.model nenh nmos
+ vto=0.779              phi=0.60
+ uo=4.0e-02             nsub=1.0e+22
+ xj=.0385u              tox=500.0e-10
+ cj=4.2e-4              cjsw=9e-10
+ gl=0.7
*
.model penh pmos
+ vto=-0.98              phi=0.60
+ uo=1.5e-02             nsub=8.158e+20
+ xj=.146u               tox=500.0e-10
+ cj=2.5e-4              cjsw=4.5e-10
+ gl=0.7
*
*
md9 vss b I0 vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md7 I0 a vss vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md6 I2 b I1 vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md3 vdd cin I3 vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md30 I3 cobar obar vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md4 obar cin I7 vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md1 vdd a I3 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md2 I3 b vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md8 I1 a obar vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md10 vss cin I2 vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md13 I0 cin vss vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md14 obar cobar I0 vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md15 vdd a I3 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md16 vdd a I3 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md17 I3 b vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md18 I3 b vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md21 vdd a I3 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md22 I3 b vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md24 I4 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md25 I5 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md26 I6 a vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md27 I7 b I6 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md28 I7 b I5 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md29 I7 b I4 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
cobar obar vss 48.70f
ca a vss 38.10f
cb b vss 40.50f
ccin cin vss 44.70f
ccobar cobar vss 44.40f
cI0 I0 vss 8.40f
cI3 I3 vss 67.20f
cI7 I7 vss 8.40f
```

Figure 5.12 (d): The extracted version of the sum generation circuit.

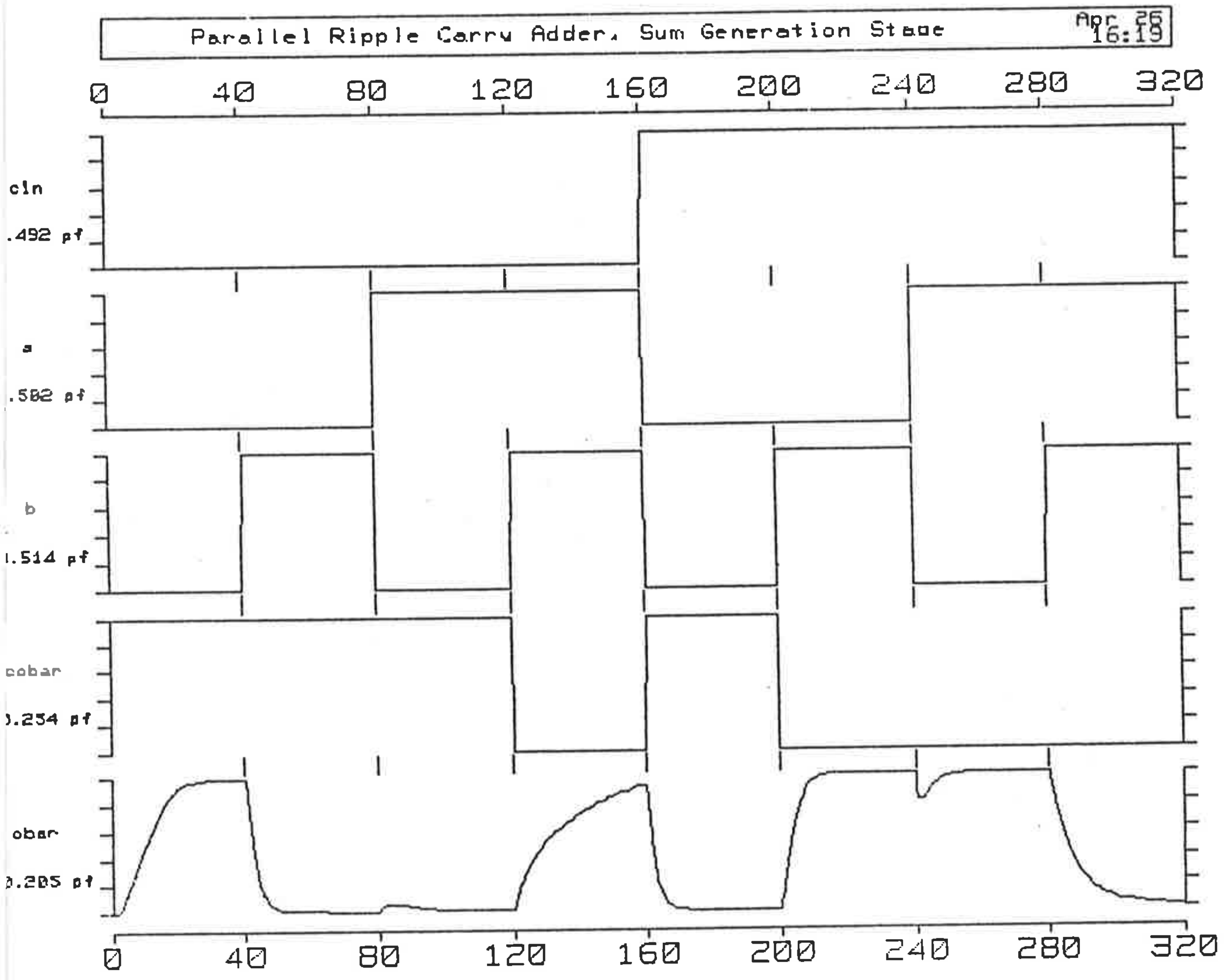


Figure 5.12 (e): The FACTS simulation of the sum generation circuit for all possible input combinations.

design. The reason behind this strategy is that the adder is to be an eight bit adder and so the crucial time delay is $nT_C = 8T_C = 56ns$ (see simulation of the eight bit adder, section 5.4.7). This delay is longer than T_{QS} and therefore T_{QS} need not be optimised. This time delay becomes critical only when the number n reduces to the point where $T_{QS} > nT_C$. By the results of the FACTS simulation for this design, the sum generation circuit would need optimisation only if the number of bits in the adder were to be $n < 8$.

c_{in}	a	b	$\overline{c_O}$	$\overline{o}$
0	0	0	1	1
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	0	0

Table 5.4: The Simulation Test of the Sum Generation Circuit using all possible combinations of input conditions.

5.4.3 **Simulation of a One Bit Adder**

The inputs to the one bit adder are c_{in}, o_a and o_b while the outputs are $\overline{o}$ and $\overline{c_O}$. Table 5.5 presents the input patterns applied to the one bit adder circuit and the expected results for all possible input combinations. Note that the difference between this table and table 5.2 is that the ouputs are complemented. The extracted circuit for the one bit adder layout is similar to the combination of the first two extracted circuits.

The important features of the simulated results as shown in Fig. 5.12 (f) are as follows:-

(i) The reults of the simulation agree with the expected results.

(ii) The connection of the two previously simulated stages provides no unexpected problems for the design. Again the worst cases of sum generation T_{QS} and carry

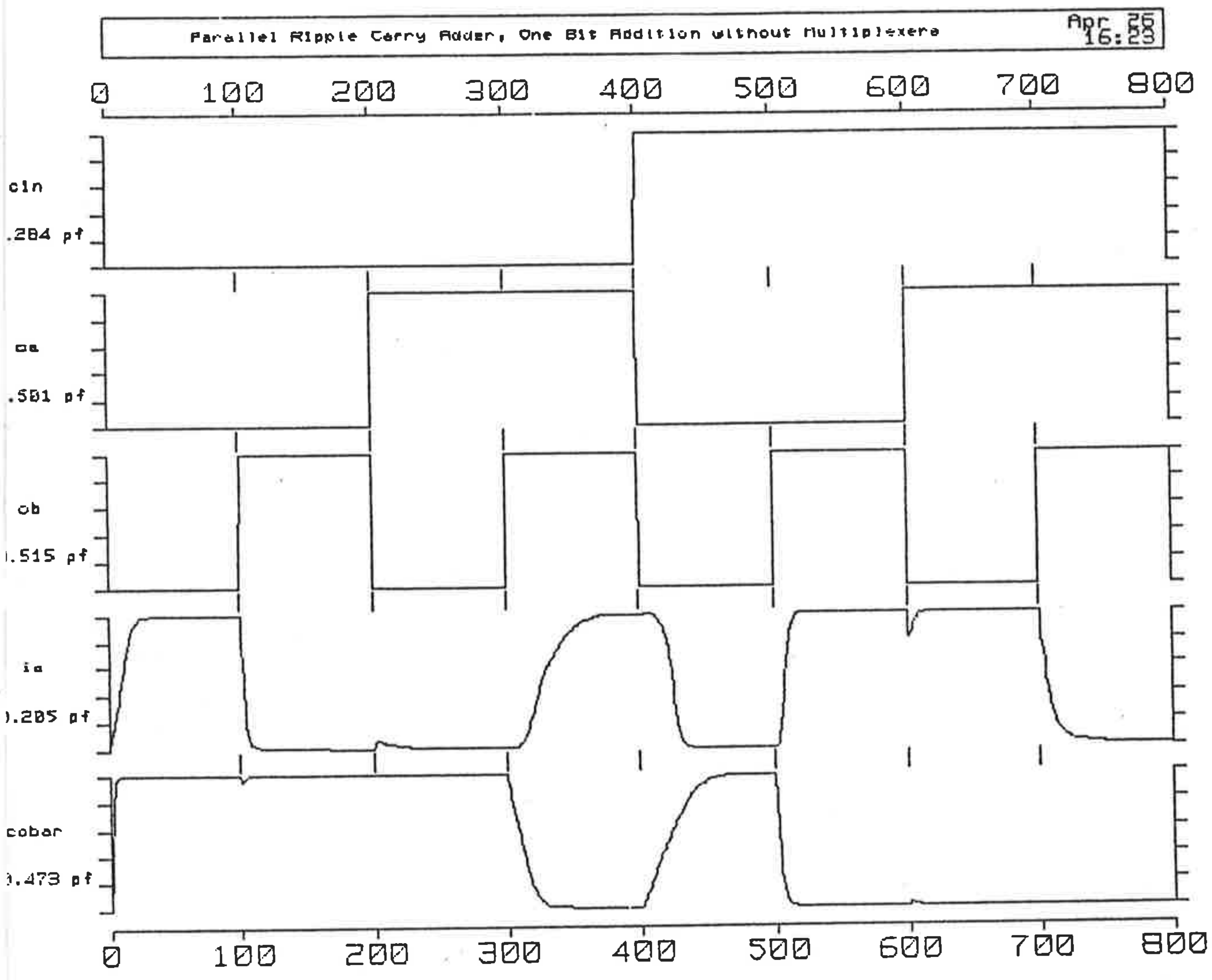


Figure 5.12 (f): The FACTS simulation of a one bit adder consisting of the carry and sum generation circuits cascaded together.

generation T_C can be seen to verify the results determined in the previous two sections.

c_{in}	o_a	o_b	$\overline{o}$	$\overline{c_o}$
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	0

Table 5.5: The Simulation Test of the One Bit Adder Circuit for all possible input combinations.

5.4.4 Simulation of the Odd and Even Multiplexers

The multiplexer stages are used as a hardware interface to control the inputs to the one bit version of the parallel ripple carry adder so that the alternate complemented/uncomplemented forms of the inputs remain consistent with the addition stage. This removes any necessity for inversion of the various inputs to the different adder stages at the time of composition of the adder into a larger system. The only interfacing required at the time of composition of the adder, apart from direct connection of inputs and outputs, will be the connection of the two control inputs s_a and s_b. As described previously, these inputs will determine which of the functions $A + B$ ($s_a = 1, s_b = 1$), $A - B$ ($s_a = 1, s_b = 0$), or $B - A$ ($s_a = 0, s_b = 1$) is performed.

The inputs to the odd and even multiplexers are therefore the same. They are s_a, s_b, a, b, and i_o, while the outputs are o_a, o_b, o and $\overline{o}$. It should be noted that the multiplexers provide both the complemented/uncomplemented inputs and outputs to and from the adder stages and i_o is the sum generated output to the multiplexer output. The differences between the odd and even multiplexer stages are the physical positioning of the outputs o and $\overline{o}$ and the results obtained from o_a and o_b for various inputs a and b. Tables 5.6 and 5.7 present the input patterns and the results obtained for the simulations of the odd and even numbered multiplexer stages.

The important features in the simulation results for the odd numbered multiplexer shown in Fig. 5.12 (g) are as follows:-

(i) The results are as expected in Table 5.6.

(ii) The outputs from o_a and o_b indicate that the odd numbered stages of the multiplexer provide uncomplemented inputs to the carry and sum generation circuits. This means that the outputs from both the carry and sum generation circuits will be in the complemented form.

(iii) The rise and fall times of the outputs o_a and o_b, or o and $\overline{o}$ will not present considerable delays in the addition process. The greatest delay in any of these rise and fall times is in the order of $T_M = 7ns$. This delay can be directly added to the worst case carry generation delay to produce the overall addition delay $T_A = T_M + nT_C = T_M + T_D = 63ns$.

The important features in the simulation results for the even numbered multiplexer shown in Fig. 5.12 (h) are as follows:-

(i) The results are as expected in Table 5.7.

(ii) The outputs from o_a and o_b indicate that the even numbered multiplexer stages provide complemented inputs to the carry and sum generation circuits. This means that the outputs from the sum and carry generation circuits are uncomplemented.

(iii) As in case (iii) for the odd numbered multiplexer, the rise and fall times of the outputs again present small delays to the overall operation of the adder. This delay can again be estimated to be in the order of $T_M = 7ns$.

The simulated results in Figs. 5.12 (g) and (h) show that the multiplexers will not significantly limit the speed of operation of the carry or sum generation stages and in fact buffer both the input and output stages. It is apparent that the overall addition delay $T_A = 63ns$ using the multiplexers. Note that the combination $s_a = s_b = 0$,

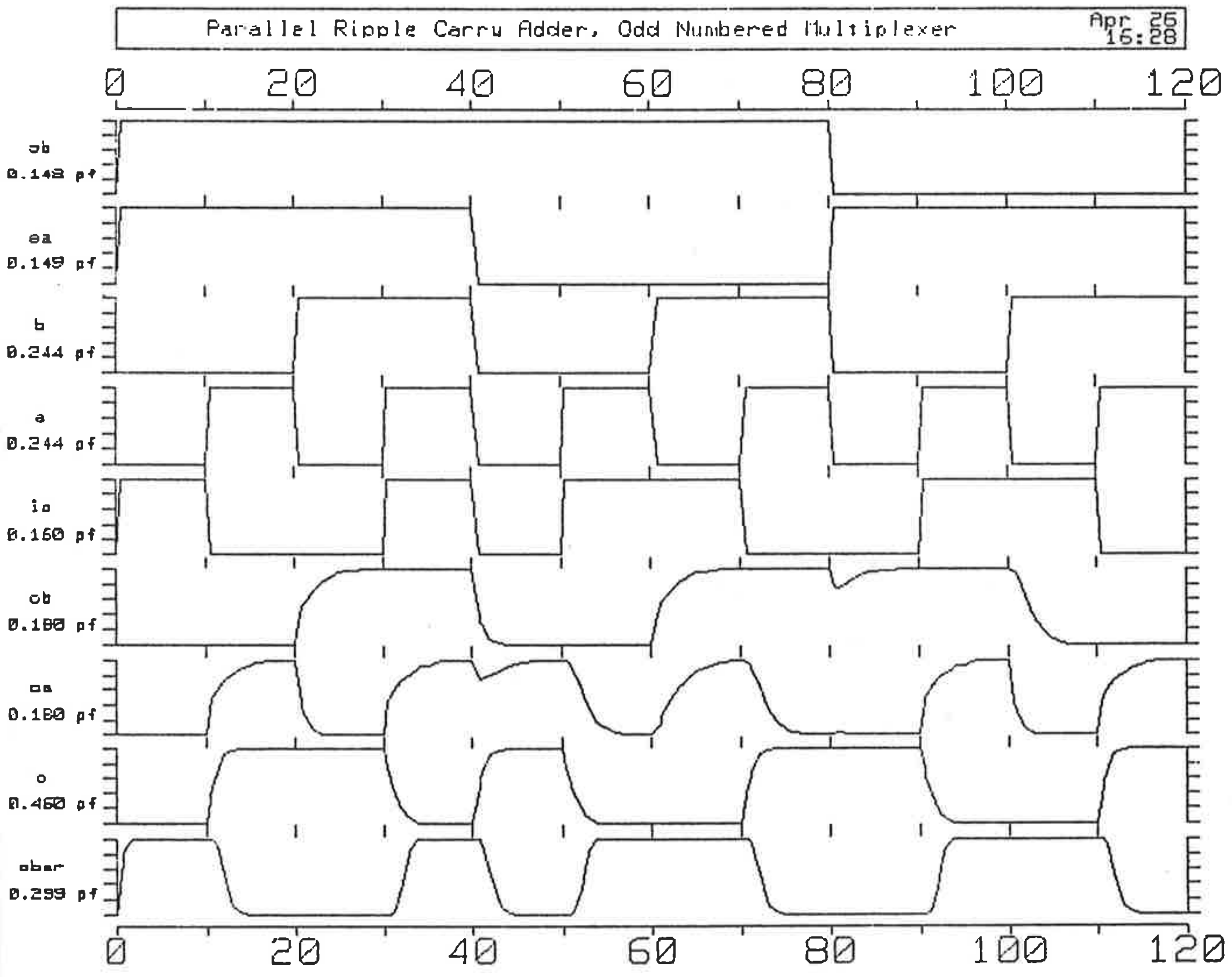


Figure 5.12 (g): The FACTS simulation of the odd numbered multiplexer.

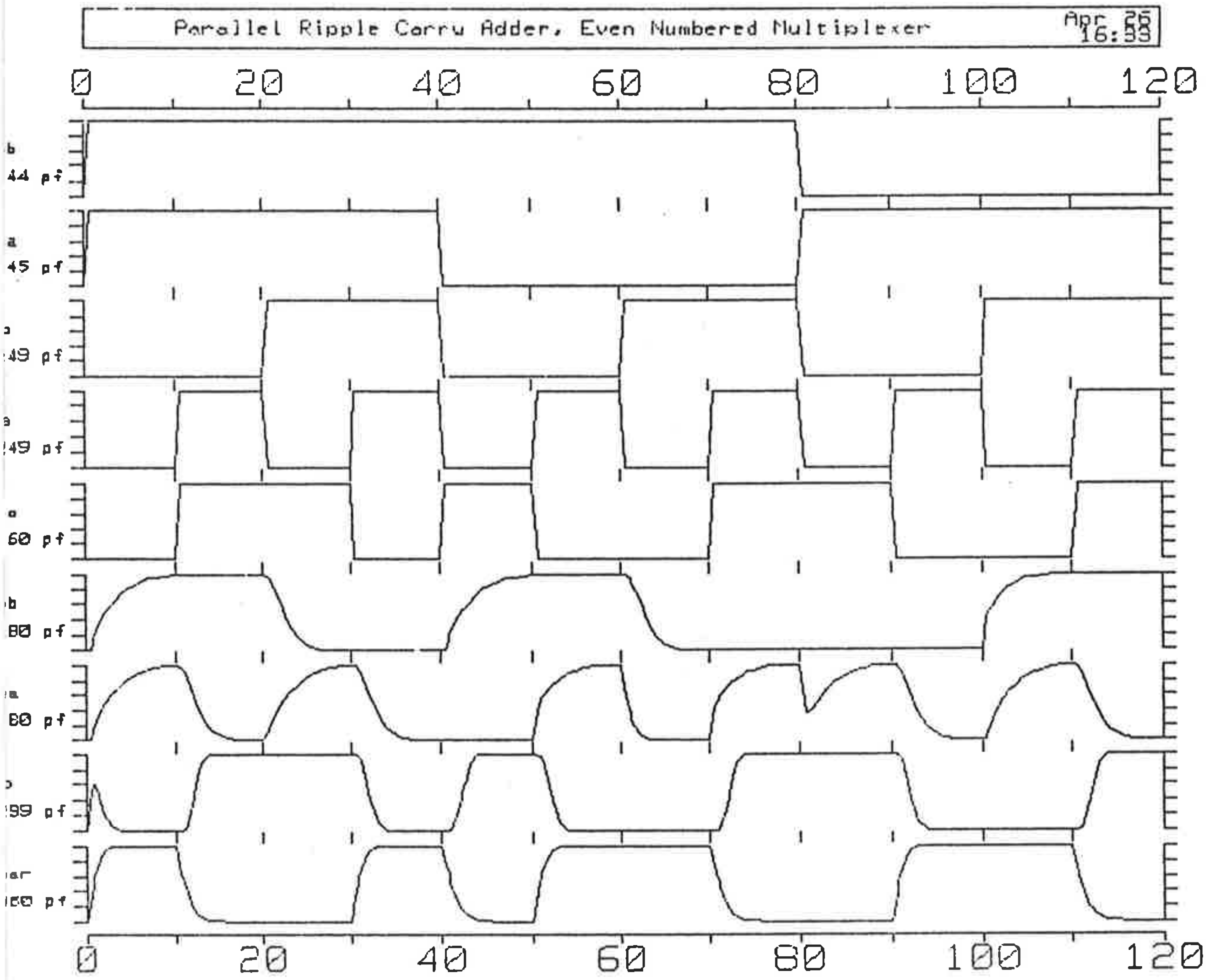


Figure 5.12 (h): The FACTS simulation of the even numbered multiplexer.

produces $-(A+B)-2$, not $-(A+B)$ as expected. To achieve the $-(A+B)$ operation using this design requires a two pass operation where $A+B$ is performed then $0-(A+B)$.

Operation	s_b	s_a	b	a	io	o_b	o_a	o	$\bar{o}$
A + B	1	1	0	0	1	0	0	0	1
	1	1	0	1	0	0	1	1	0
	1	1	1	0	0	1	0	1	0
	1	1	1	1	1	1	1	0	1
B - A	1	0	0	0	0	0	1	1	0
	1	0	0	1	1	0	0	0	1
	1	0	1	0	1	1	1	0	1
	1	0	1	1	0	1	0	1	0
A - B	0	1	0	0	0	1	0	1	0
	0	1	0	1	1	1	1	0	1
	0	1	1	0	1	0	0	0	1
	0	1	1	1	0	0	1	1	0

Table 5.6: The simulation test of the odd numbered multiplexer for all valid combinations of input conditions. The outputs o_a and o_b are used as inputs to the one bit adder whereas the input io is the expected one bit adder sum generated output.

Operation	s_b	s_a	b	a	io	o_b	o_a	o	$\bar{o}$
A + B	1	1	0	0	0	1	1	0	1
	1	1	0	1	1	1	0	1	0
	1	1	1	0	1	0	1	1	0
	1	1	1	1	0	0	0	0	1
B - A	1	0	0	0	1	1	0	1	0
	1	0	0	1	0	1	1	0	1
	1	0	1	0	0	0	0	0	1
	1	0	1	1	1	0	1	1	0
A - B	0	1	0	0	1	0	1	1	0
	0	1	0	1	0	0	0	0	1
	0	1	1	0	0	1	1	0	1
	0	1	1	1	1	1	0	1	0

Table 5.7: The simulation test of the even numbered multiplexer for all valid input combinations.

5.4.5 Simulation of the Carry In Generation Circuit

Considering that the multiplexing circuitry has been included with the adder stages in an effort to design an easily cascadable building block, a final leaf cell must be added to the first stage of the cascaded adder cell to make the design complete. This leaf cell is the carry input generation circuit to the first stage of the adder.

```
.model nenh nmos
+ vto=0.779               phi=0.60
+ uo=4.0e-02              nsub=1.0e+22
+ xj=.0385u               tox=500.0e-10
+ cj=4.2e-4               cjsw=9e-10
+ gl=0.7
*
.model penh pmos
+ vto=-0.98               phi=0.60
+ uo=1.5e-02              nsub=8.158e+20
+ xj=.146u                tox=500.0e-10
+ cj=2.5e-4               cjsw=4.5e-10
+ gl=0.7
*
*
md17 sb sa co vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md16 co I0 I1 vdd penh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md15 co sa I1 vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md14 sb I0 co vss nenh l=3.0u w=7.0u ad=56.0p pd=30.0u as=56.0p ps=30.0u
md13 I0 sa vss vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md12 vss sb I1 vss nenh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md11 vdd sb I1 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md10 I0 sa vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md9 I0 sa vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md8 vdd sb I1 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md7 vdd sb I1 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md6 I0 sa vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md5 I0 sa vdd vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
md4 vdd sb I1 vdd penh l=3.0u w=14.0u ad=112.0p pd=44.0u as=112.0p ps=44.0u
cco co vss 21.00f
csb sb vss 40.20f
csa sa vss 43.50f
cI0 I0 vss 29.70f
cI1 I1 vss 30.60f
```

Figure 5.12 (i): The extracted version of the first stage carry in generation circuit.

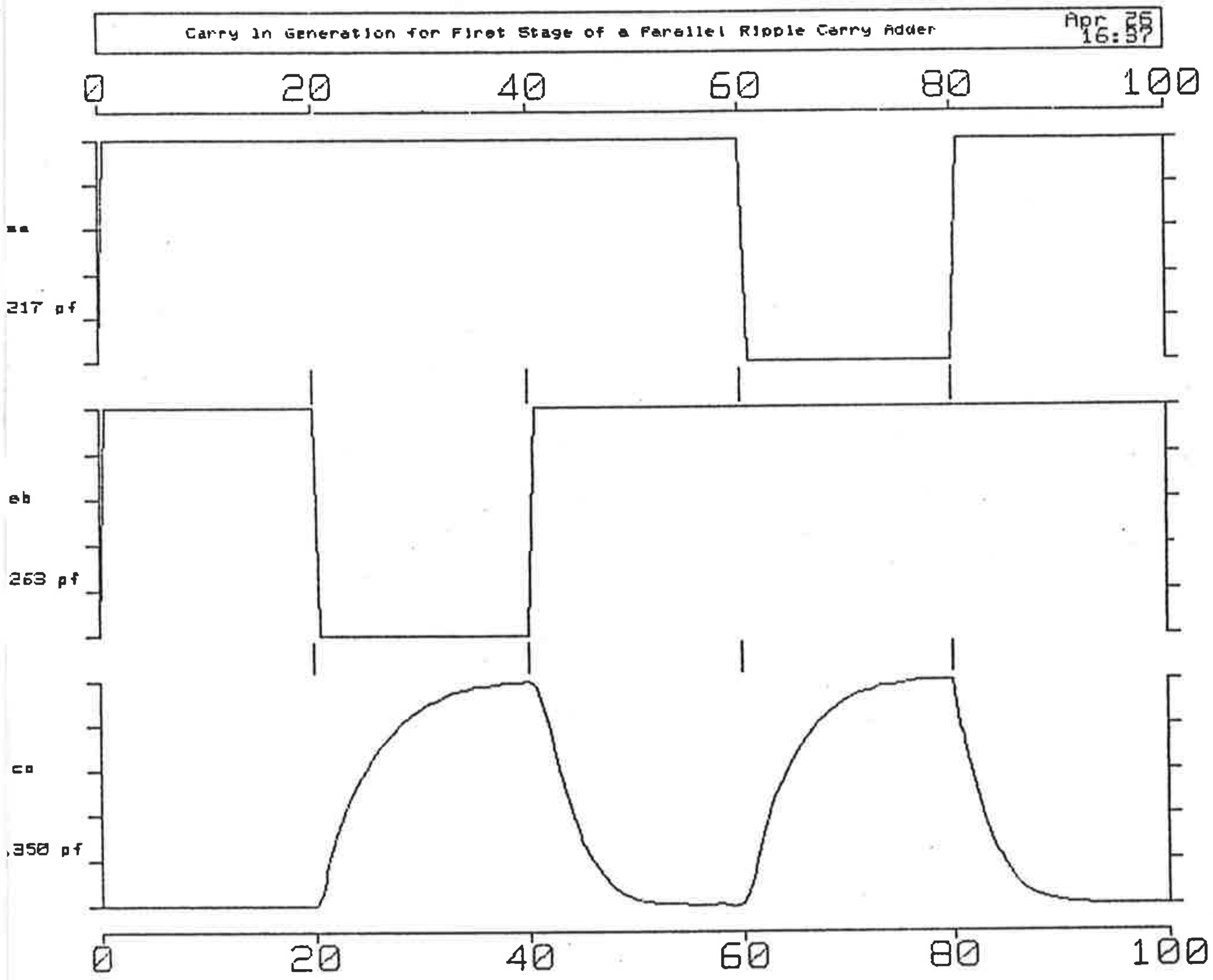


Figure 5.12 (j): The simulation of the first stage carry in generation circuit.

The inputs to the carry in generation circuit are s_a and s_b while the output is c_o. Table 5.8 presents the input patterns applied to the carry in generation circuit and the expected results are shown for all valid combinations of s_a and s_b. The extracted circuit for the carry in generation circuit is given in Fig. 5.12 (i).

s_a	s_b	c_o
1	1	0
1	0	1
1	1	0
0	1	1
1	1	0

Table 5.8: The simulation test of the carry in generation circuit for all valid combinations of s_a and s_b. Note that $s_a = s_b = 0$ is not a valid combination.

The important features in the simulation results in Fig. 5.12 (j) are as follows:-

(i) The results are as expected in table 5.8.

(ii) The output co shows that for $s_a = s_b = 1$, $c_{in} = 0$, while if $s_a \neq s_b$, $c_{in} = 1$ which is consistent with the required inputs to the first (odd numbered) stage. That is for $A + B$, the carry in to the first stage should be 0. For the operations $A - B$ or $B - A$, the carry in should be 1.

5.4.6 Simulation of a Two Bit Adder

The fundamental cascadable adder building block that can be constructed using the design so far discussed and simulated is the two bit adder block. Using the even and odd multiplexer design blocks on top of two cascaded one bit adders provides even and odd one bit cells cascaded to form a two bit block that can be replicated to the designer's specifications to form an n-bit adder.

The inputs to the two bit adder are a_1, a_2, b_1, b_2, s_a, s_b, and c_{in} while the outputs are o_1, o_2, $\overline{o_1}$, $\overline{o_2}$ and c_o. Table 5.9 presents the input patterns applied to the two bit adder and the expected outputs.

Operation	s_b	s_a	c_{in}	$a2$	$b2$	$a1$	$b1$	o_2	c_{o2}	o_1
A + B	1	1	0	0	0	0	0	0	0	0
	1	1	0	0	0	0	1	0	0	1
	1	1	0	0	0	1	0	0	0	1
	1	1	0	0	0	1	1	1	0	0
	1	1	0	0	1	0	0	1	0	0
	1	1	0	0	1	0	1	1	0	1
	1	1	0	0	1	1	0	1	0	1
	1	1	0	0	1	1	1	0	1	0
	1	1	0	1	0	0	0	1	0	0
	1	1	0	1	0	0	1	1	0	1
	1	1	0	1	0	1	0	1	0	1
	1	1	0	1	0	1	1	0	1	0
	1	1	0	1	1	0	0	0	1	0
	1	1	0	1	1	0	1	0	1	1
	1	1	0	1	1	1	0	0	1	1
	1	1	0	1	1	1	1	1	1	0
B - A	1	0	1	0	0	0	0	0	1	0
	1	0	1	0	0	0	1	0	1	1
	1	0	1	0	0	1	0	1	0	1
	1	0	1	0	0	1	1	0	1	0
	1	0	1	0	1	0	0	1	1	0
	1	0	1	0	1	0	1	1	1	1
	1	0	1	0	1	1	0	0	1	1
	1	0	1	0	1	1	1	1	1	0
	1	0	1	1	0	0	0	1	0	0
	1	0	1	1	0	0	1	1	0	1
	1	0	1	1	0	1	0	0	0	1
	1	0	1	1	0	1	1	1	0	0
	1	0	1	1	1	0	0	0	1	0
	1	0	1	1	1	0	1	0	1	1
	1	0	1	1	1	1	0	1	0	1
	1	0	1	1	1	1	1	0	1	0
A - B	0	1	1	0	0	0	0	0	1	0
	0	1	1	0	0	0	1	1	0	1
	0	1	1	0	0	1	0	0	1	1
	0	1	1	0	0	1	1	0	1	0
	0	1	1	0	1	0	0	1	0	0
	0	1	1	0	1	0	1	0	0	1
	0	1	1	0	1	1	0	1	0	1
	0	1	1	0	1	1	1	1	0	0
	0	1	1	1	0	0	0	1	1	0
	0	1	1	1	0	0	1	0	1	1
	0	1	1	1	0	1	0	1	1	1
	0	1	1	1	0	1	1	1	1	0
	0	1	1	1	1	0	0	0	1	0
	0	1	1	1	1	0	1	1	0	1
	0	1	1	1	1	1	0	0	1	1
	0	1	1	1	1	1	1	0	1	0

Table 5.9: The simulation test of the two bit adder circuit.

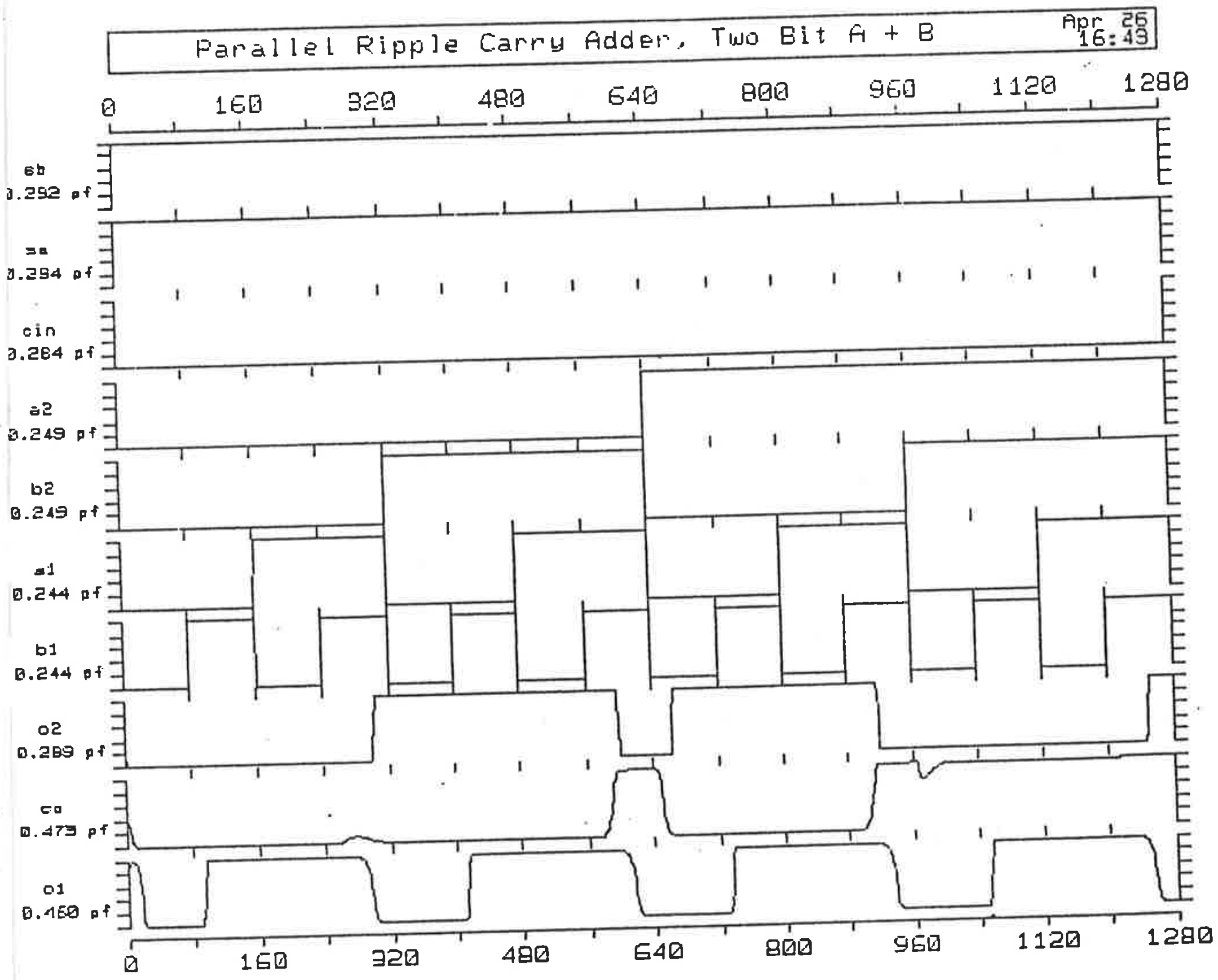


Figure 5.12 (k): The FACTS simulation of the two bit adder for the operation A + B.

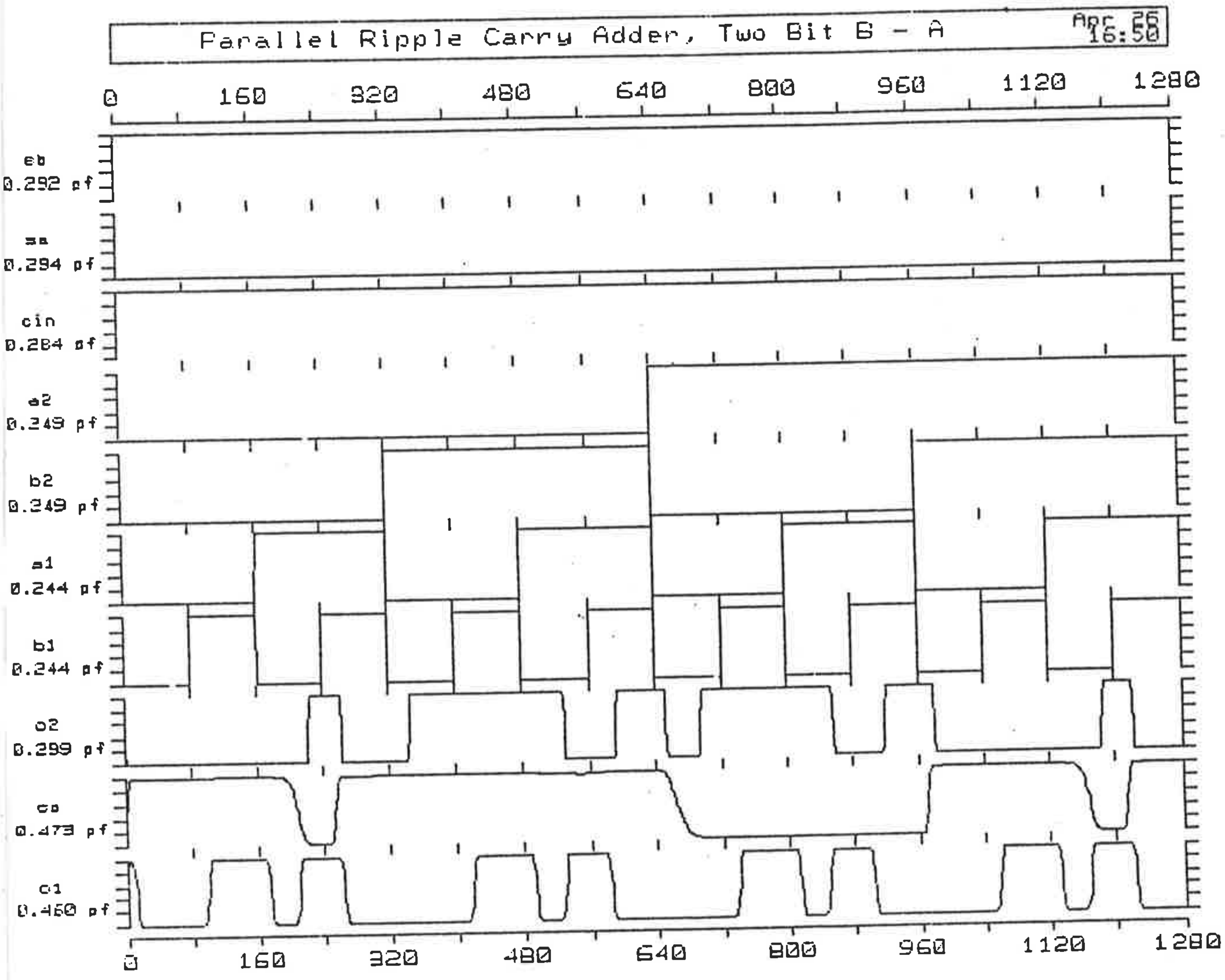


Figure 5.12 (l): The FACTS simulation of the two bit adder for the operation B - A.

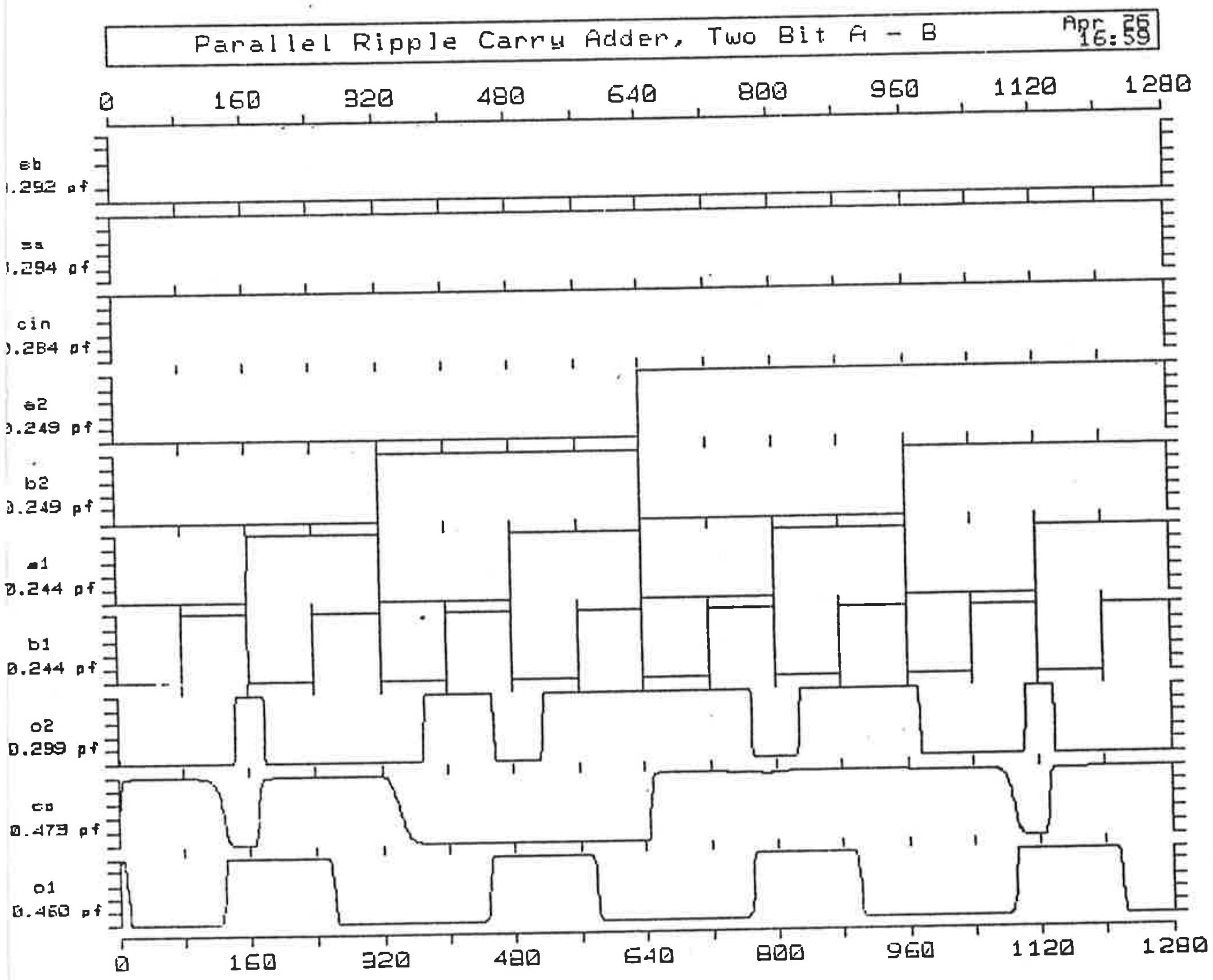


Figure 5.12 (m): The FACTS simulation of the two bit adder for the operation A - B.

Separate simulations have been run for each of the operations $A+B$, $A-B$ and $B-A$ for reasons of clarity. The important features in the simulation results shown in Figs. 5.12 (k), (l) and (m) are as follows:-

(i) The results are as expected in table 5.9.

(ii) It is important to note that the longest time taken to generate the correct output for any of the conditions indicated in any of Figs. 5.12 (k), (l) or (m) is approximately 50ns. These times can be interpreted as the quiescent time delay for the addition result to be produced, T_{QS}, or the delay T_{QC} discussed previously.

Examining a simulation of the worst case for propagation of the carry through the two bit adder block allows the designer to determine values for T_D with $n = 2$. Table 5.10 gives the input conditions for the worst case for carry propagation through the two bit adder and the expected results.

a_2	b_2	a_1	b_1	c_{in}	c_o
0	1	0	1	0	0
0	1	0	1	1	1
0	1	0	1	0	0
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	0	0	0

Table 5.10: Simulation of worst case carry propagation in the two bit adder.

The features of the simulation shown in Fig. 5.12 (n) are as follows:-

(i) The results are as expected in table 5.10.

(ii) The carry propagation through two bits of the adder indicates that the delay $T_D = 14ns$ and so $T_C = 7ns$. This indicates that the overall delay through an n-bit adder would be $T_D = n.7ns$. This result is important as it must be used in conjunction with T_{QC} and T_{QS} to determine whether the layout needs to be optimised to reduce the critical delay path. Any one of T_D, T_{QC} or T_{QS} may

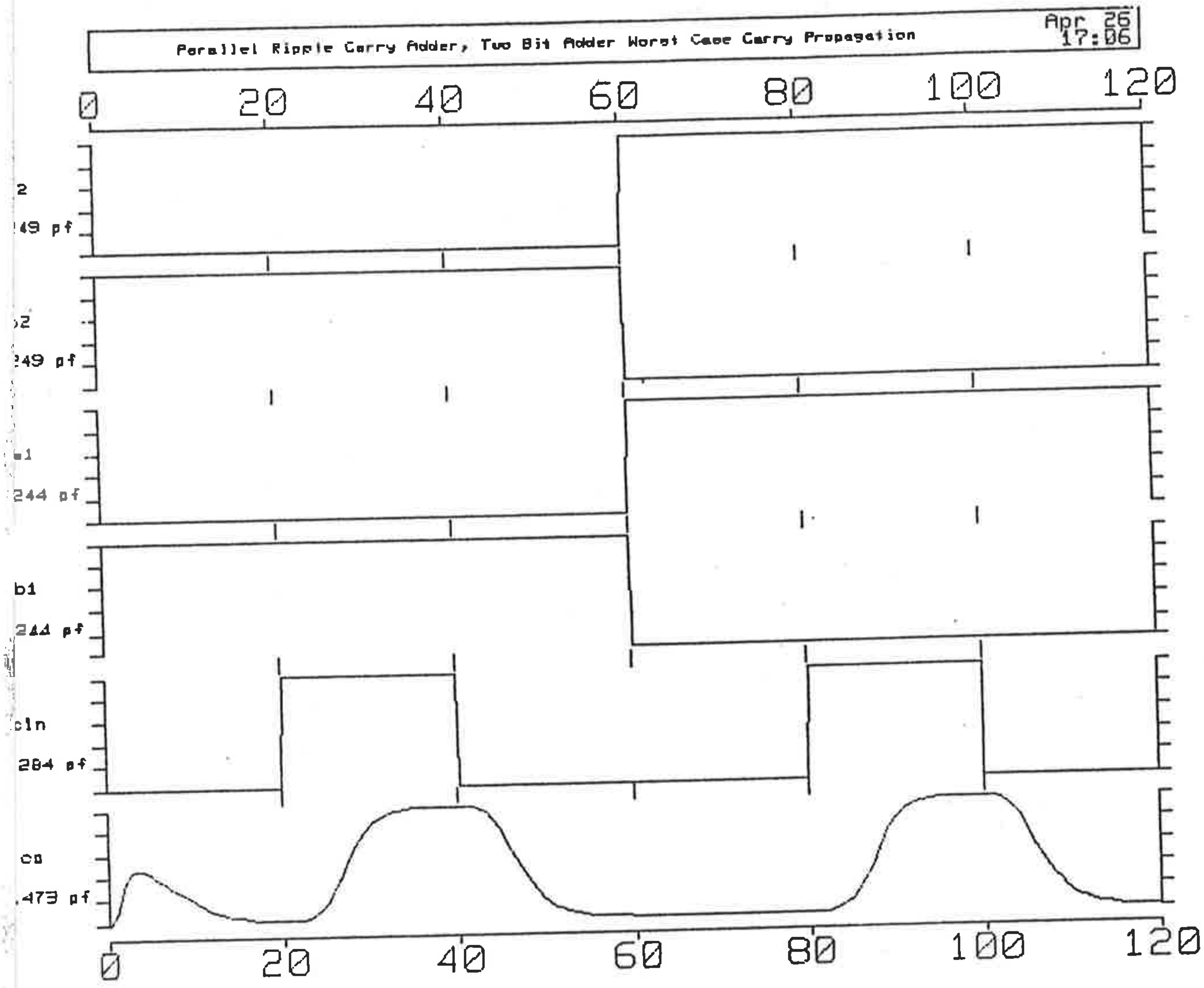


Figure 5.12 (n): The FACTS simulation of the two bit adder for the worst case carry propagation.

be the critical time delay depending upon how many stages are used in the addition circuit.

5.4.7 Simulation of an Eight Bit Adder

Considering the previous simulations allowing approximation of values for each of T_C, T_{QC} and T_{QS}, it is clear that for an adder where $n = 8$, the delays T_{QS}, T_{QC} and $T_D = 8T_C = 56ns$ are quite similar. Simulations are required that will determine which of these delays is the more critical so that the circuit may be optimised to produce an eight bit addition result in the fastest time.

The simulation required for the worst case carry propagation test will be such that all of the inputs $a_1 ... a_8$ and $b_1 ... b_8$ will be opposite and s_a and s_b forced to go through transitions to provide the necessary c_{in} transition at the first stage of the adder. The respective input patterns and expected output results are indicated in Table 5.11. Two possible operating conditions, $A - B$ and $B - A$ will cause this worst case carry propagation to occur.

The results of these simulations can be seen in Figs. 5.12 (o) and (p). The results are as expected. The output rise and fall times t_r and t_f are obviously very different. The worst values for the two delays, as determined by the FACTS results, are $t_f = 56ns$ while $t_r = 28ns$. The delay T_D for the overall circuit is determined by the greater of these two delays, $t_f = 56ns$.

s_b	s_a	$a1..a8$	$b1..b8$	c_{in}	c_o
1	1	0	1	0	0
0	1	0	1	1	1
0	1	0	1	1	1
1	1	0	1	0	0
1	1	0	1	0	0
1	0	0	1	1	1
1	0	0	1	1	1
1	1	0	1	0	0

Table 5.11: Simulation of the worst case carry propagation through the eight bit adder.

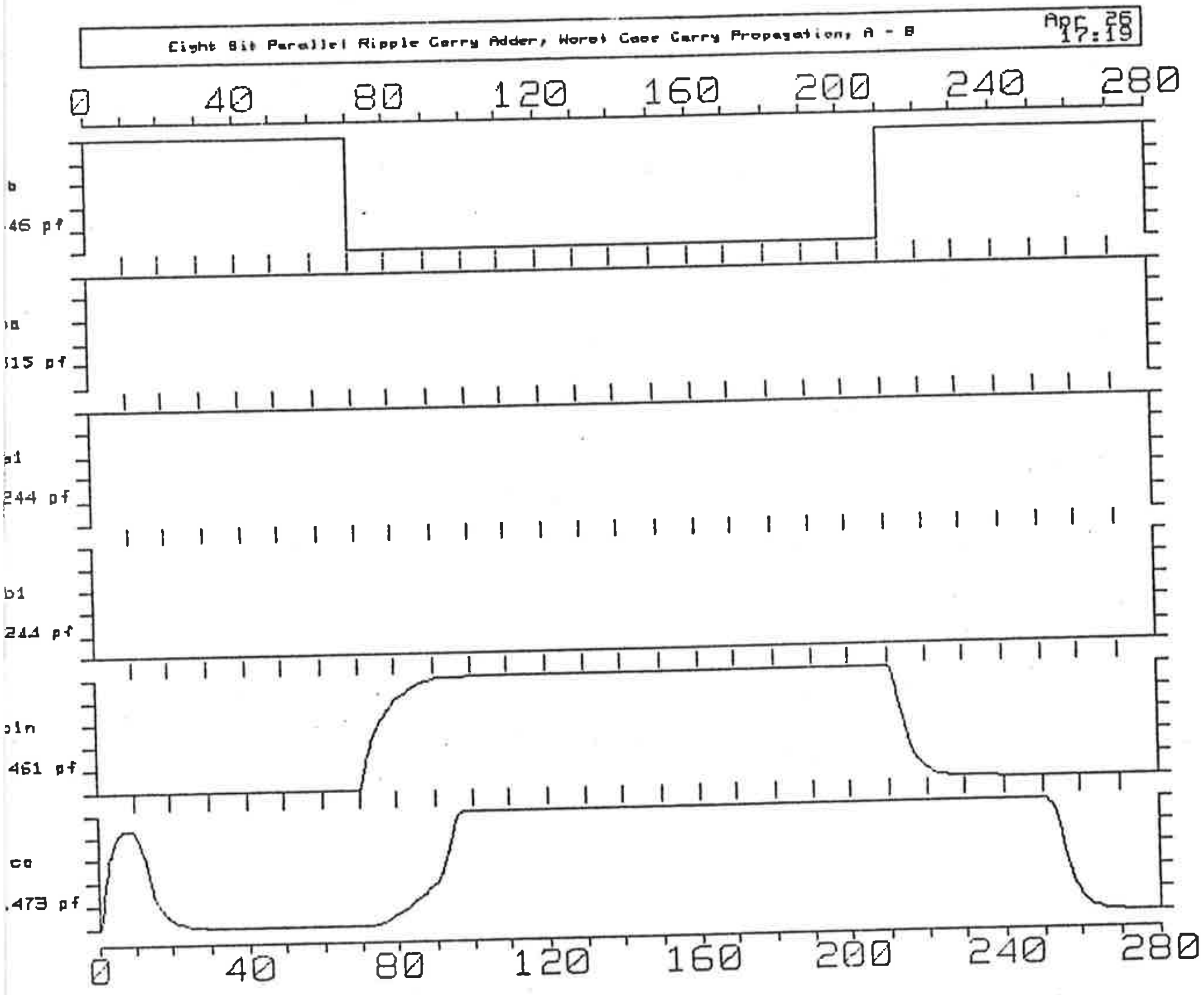


Figure 5.12 (o): Worst case carry propagation through an eight bit adder for the case A - B.

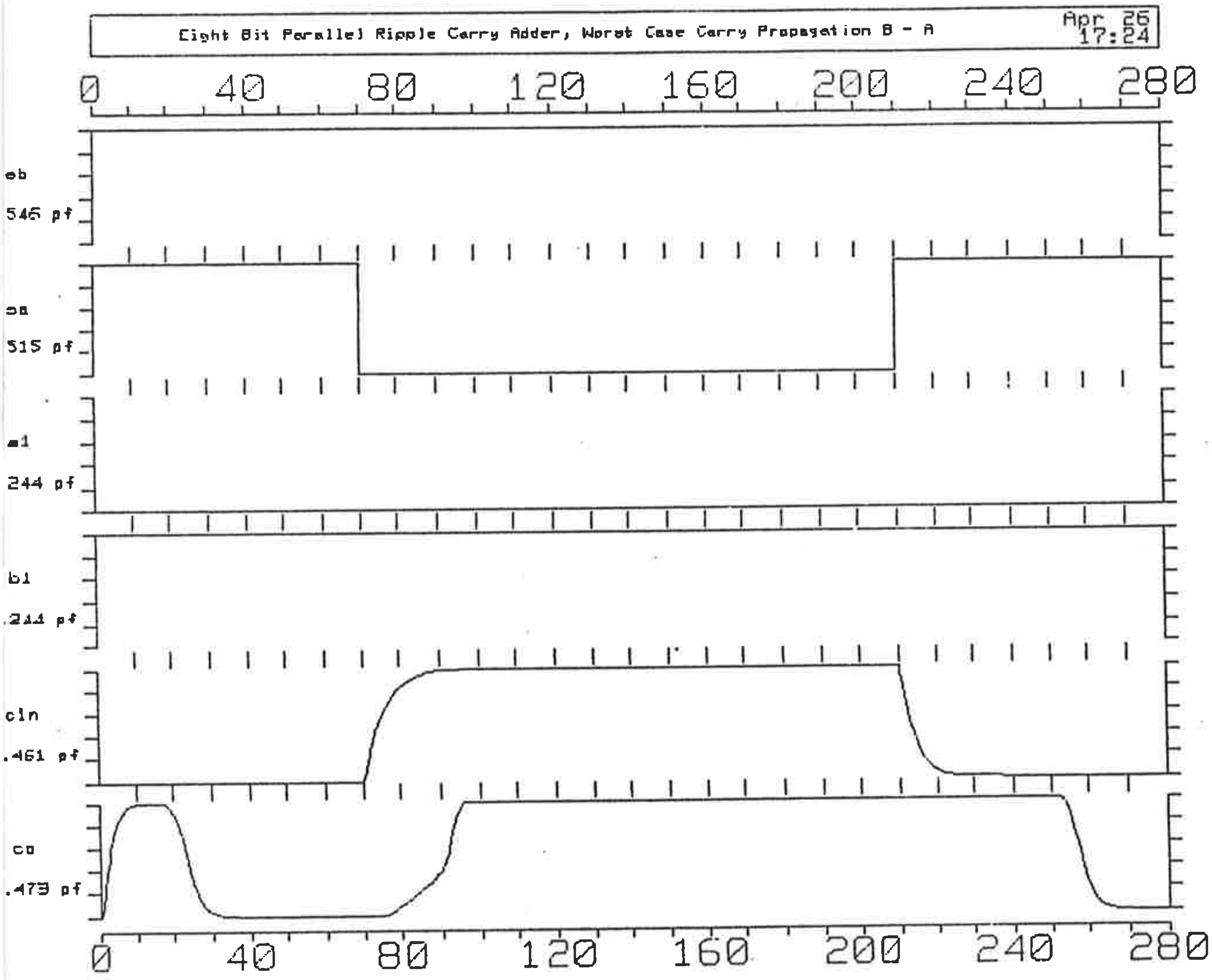


Figure 5.12 (p): Worst case carry propagation through an eight bit adder for the case B - A.

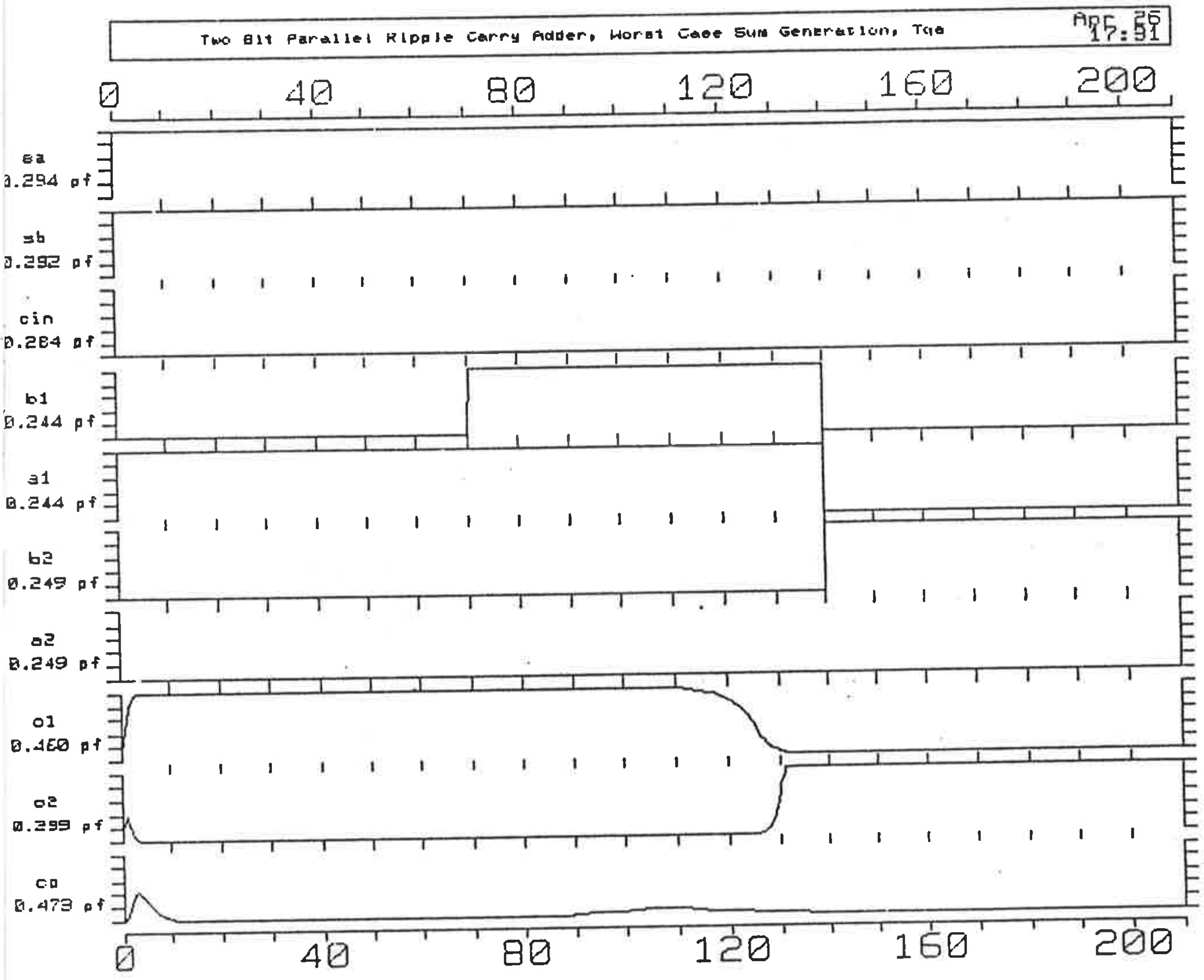


Figure 5.12 (q): Example of worst case for either T_QS or T_QC after examining Figs. 5.12 (k), (l) and (m). It is no worse than $T_D = 8.T_C$

The simulation in Fig. 5.12 (q) shows the expanded simulation of the apparent worst case for T_{QS} or T_{QC}, as determined by examining Figs. 5.12 (k), (l) and (m). The results for these delays are tested on the two bit adder. These results show that $T_{QS} = 50ns$ at worst and so the layout described in this chapter has minimised the critical timing paths to provide an eight bit addition within $T_A = T_M + T_D = 63ns$.

5.5 Compaction and Mask Level Verification

After the symbolic layout has been optimised to the designer's requirements as indicated in the previous sections of this chapter, it is only necessary to compact the design to produce an efficient mask level description of the design, translate to CIF, add pads using a mask level geometry editor such as KIC, and have the circuit fabricated using a Silicon Broker as described in previous chapters.

The conversion from ABCD representation to CIF is a two stage process. The conversion from ABCD to LLAMA is performed by the VIVID compactor. The LLAMA format provides a mask level description of the circuit. From this intermediate LLAMA format, the tool ATOLL is used to create CIF or any one of a number of other physical layout descriptions (see Chapter III) for the purpose of mask generation. The CIF layout of the two bit adder design can be seen in Fig. 5.13. Compaction and translation into CIF of the complete circuit allows comparative mask level design rule checks and simulations to be carried out.

Once in the CIF format, further extraction and simulation can be used to provide a more accurate estimate of circuit operation. The main differences in the simulation results should be introduced by the absolute layout of wires specified in CIF. An example of the translation process can be seen in Appendix F which shows the ABCD, LLAMA and CIF representations respectively for the carry generation stage of the adder along with its CIF layout which can be compared with the ABCD layout in Fig. 5.8 (b).

It should be noted at this stage that the parameters to be used in the extraction and simulation tools should all be derived from the same fabrication process. Those used here for extraction and simulation of the CIF circuits are not the same as those used for the symbolic level extraction and simulation tools. However they are similar enough to provide an approximate idea of the comparative results obtained using a currently operational CMOS process in the US, the MOSIS $3\mu m$ process to extract and simulate with at the symbolic level, as opposed to approximations for an AWA CMOS process used for extraction and simulation at the mask level.

Simulation on the CIF form of the two bit adder was carried out to verify the correct operation of the CIF version of the adder structure which had been simulated in ABCD form. Subsequent simulation was carried out on an eight bit version of the same design to verify the worst case carry propagation speeds. The CIF versions of the designs are first checked and flattened using the design rule checker CHECK. The circuits are then extracted using NET and simulated with PROBE. The simulations carried out on the designs are the same as those carried out on the ABCD versions of the designs.

Fig. 5.14 (a) shows the PROBE results of the simulation run specified by Table 5.9 as applied to the circuit extracted from the CIF description of the two bit adder by the tool NET. Fig. 5.14 (b) shows the simulated results of the test specified by Table 5.10. The results of both of these simulations show that the carry propagation delay is of the same order of magnitude as that estimated by the symbolic level simulator FACTS. The actual propagation delay can be approximated to within 10ns per stage.

The simulated results of the mask level designs and the design rule checks run on the designs indicate that the circuit shown in Fig. 5.15 will function very close to the speed predicted by FACTS. Of course, any simulation is only as good as the model parameters and process parameters given to it and allowances need to be made for different fabrication facilities. The ability to accurately predict operation of IC designs requires an intimate knowledge of the fabrication process as well as the use of detailed

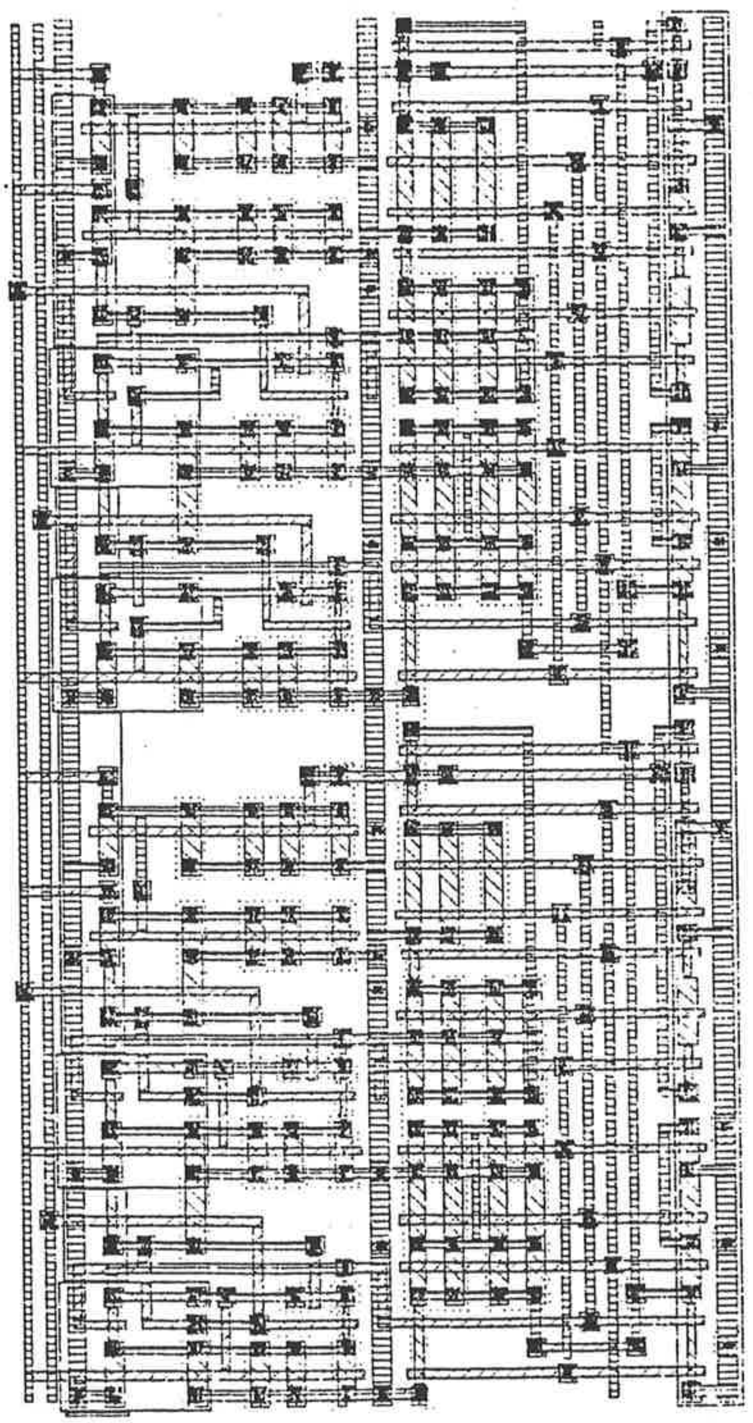


Figure 5.13: The CIF layout of the complete two bit adder.

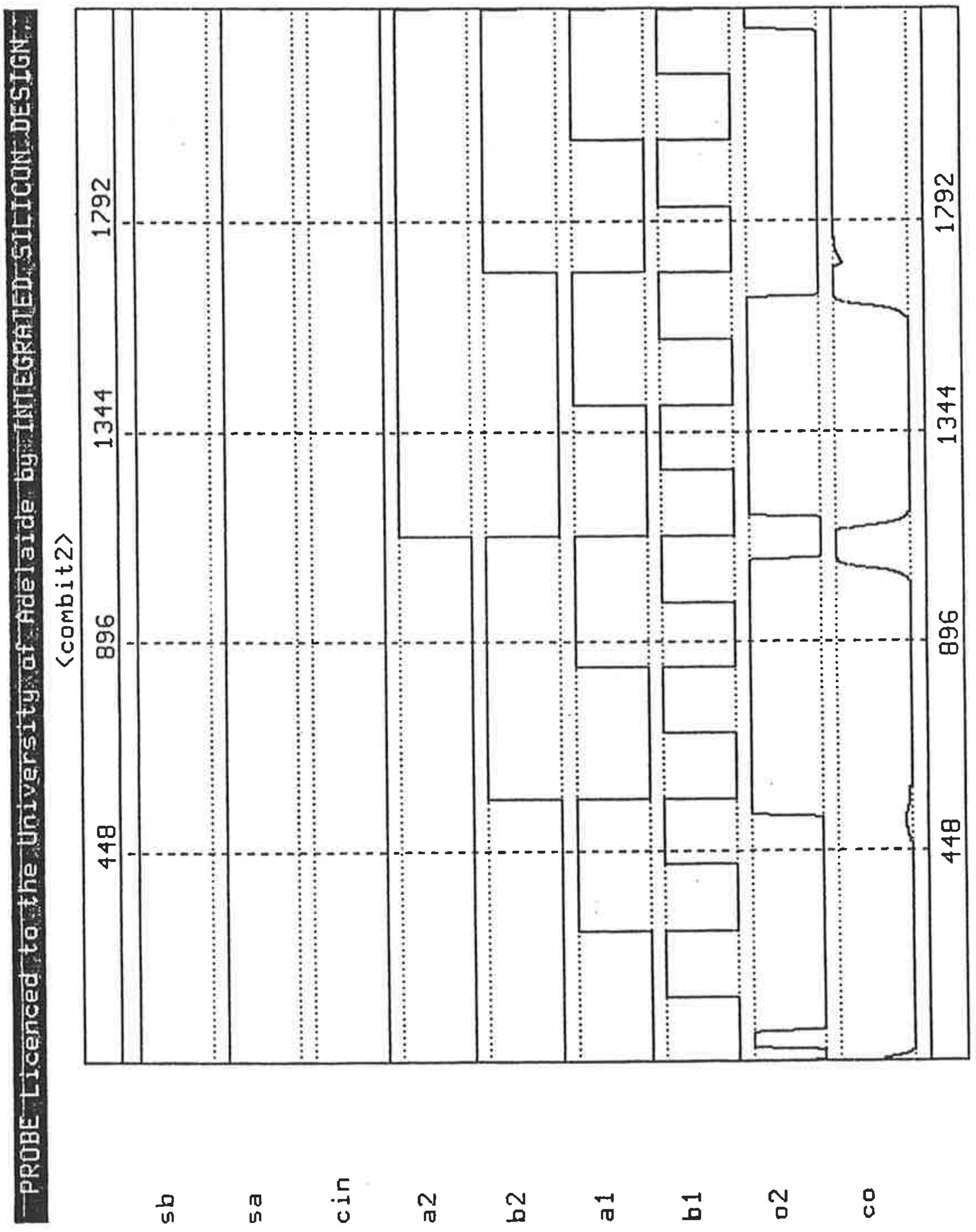


Figure 5.14 (a): The PROBE simulation of the circuit extracted from CIF according to table 5.9, A + B.

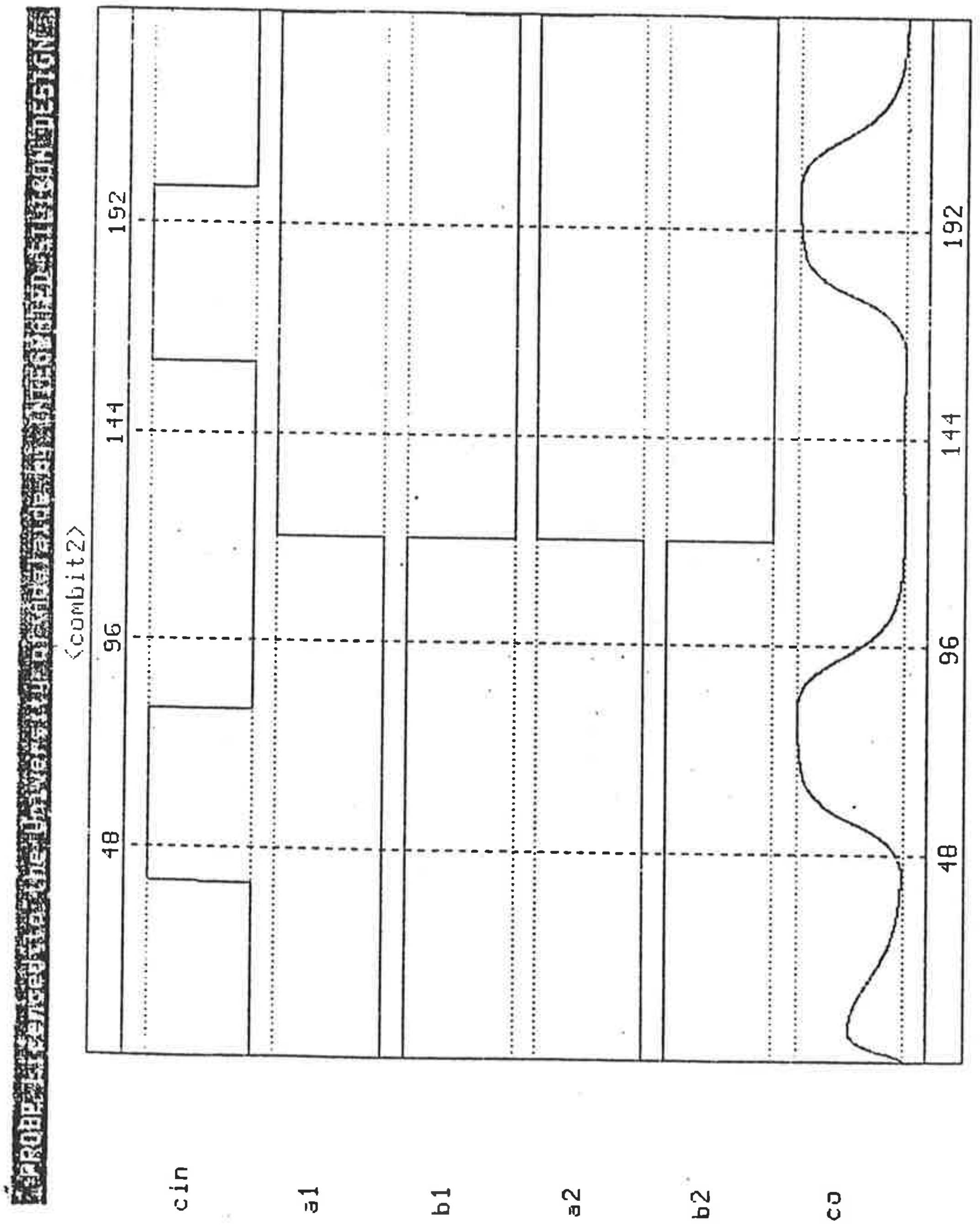


Figure 5.14 (b): The PROBE simulation for the worst case carry propagation through the two bit adder, according to table 5.10.

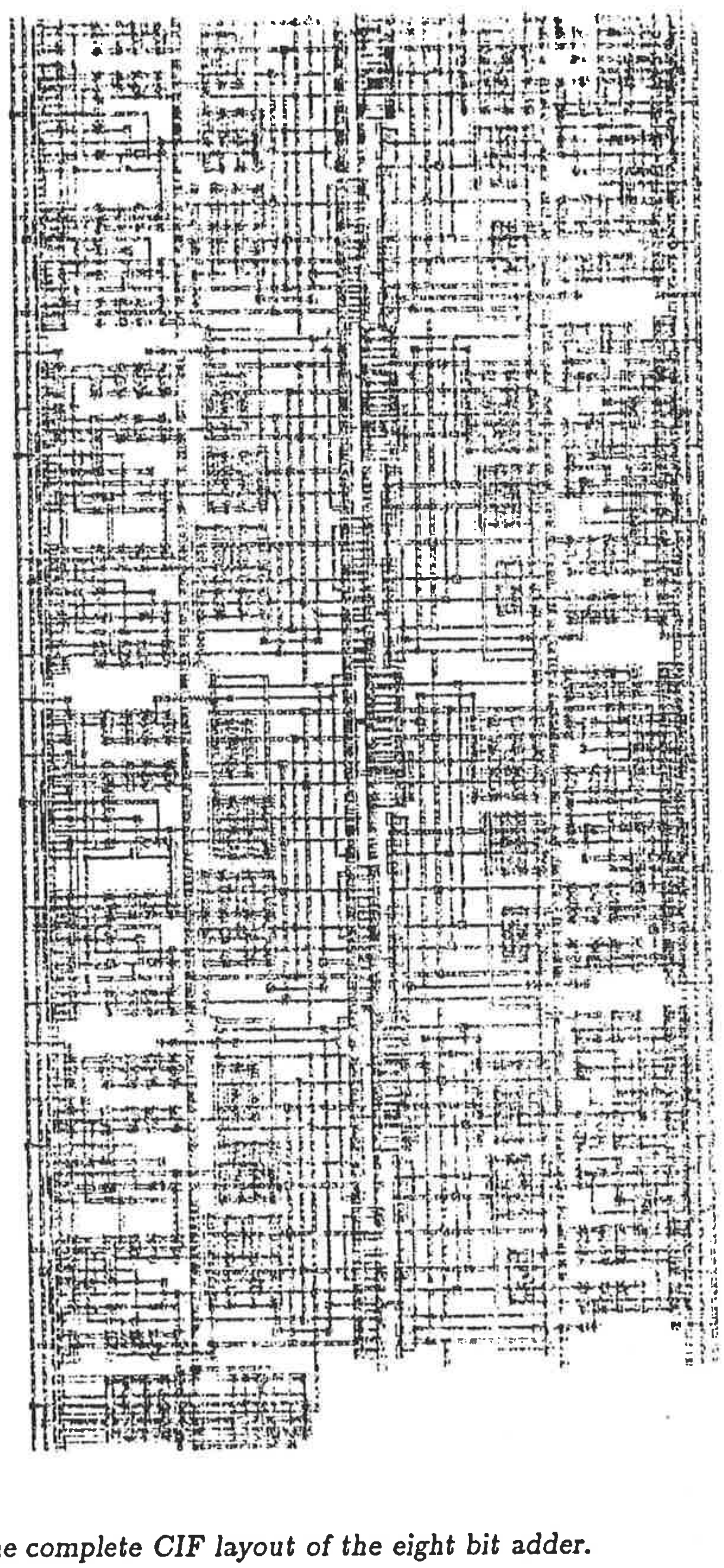


Figure 5.15: The complete CIF layout of the eight bit adder.

models within the simulator. Both PROBE and FACTS allow estimates of operation that can predict the correct orders of magnitude of operating speeds.

5.6 CMOS MPC Fabrication and Conclusions

The CMOS adder design described in this chapter is almost ready for fabrication on the AWA CMOS fabrication line. All that is needed is a set of input and output pads connected to the eight bit design. Work has been done at Adelaide University on producing a set of CMOS pads for use with MPC designs[92][93], but has not yet been completed. Once these last barriers are overcome, the mask level graphics editor KIC can be used to connect the pads to the adder design and the design will be completed, ready for fabrication. The design should allow an eight bit addition to be completed within approximately 80ns. This fundamental time delay can then be used in all subsequent calculations for operations performed in TFB if this adder design is to be implemented.

MPC runs using nMOS technology have been tried and proven using a number of sources as indicated in this thesis. The design work detailed in this chapter has been performed in anticipation of similar MPC facilities being created for the fabrication of CMOS designs[94]. This work has been completed using a version of CIF describing the design for the AWA CMOS process in Sydney. Work done by ISD should provide MPC services using a CMOS technology conforming to the Weste and Eshraghian design rules[48] some time in 1986. The flowchart for such a fabrication run would be similar to that described by Fig. 1.5 (b), the only difference being that the technology used would be CMOS.

It is expected that this design should function correctly. The accuracy of the timing estimates provided by FACTS and PROBE can only be determined by testing a fabricated circuit. As long as the device and process parameters used in these simulators are accurate, the limiting factor is the accuracy of the model used by the simulators to predict circuit operation.

CHAPTER VI

ANALYSING THE PAST AND LOOKING INTO THE FUTURE

Maintaining the analogy between various levels of sophistication of Programming Languages and of Computer Aided VLSI Circuit Design tool sets, the results of this research are partitioned into their appropriate sections. As a conclusion to the work presented in this thesis, a final summary is made of the results obtained in chapters IV and V providing statistical evidence that the advances made in the CAD VLSI Circuit Design tools at the University of Adelaide have resulted in a state of the art CAD tool set that enables fabrication of Integrated Circuits in either nMOS or CMOS technologies. Using this thesis as reference material, a final look is taken at trends in tool development and tools of the future are foreshadowed.

6.1 Mask Level Tools

While the results of chapters IV and V indicate that the use of mask level tools for VLSI circuit design is not recommended, it is still possible. The mask level CAD tools presented in this thesis allow the completion of the IC design process from layout to total verification, including design rule checking, circuit extraction and simulation at the circuit, timing and logic levels.

To try and design full custom VLSI circuits using the mask level tools is still a very time consuming, error prone process as the designer needs to know about interrelationships of mask layers as well as circuit design techniques. This added complexity magnifies the burden of an already difficult design task, and provides the main source of error in the design process. Perhaps the most substantial evidence to indicate this complexity is the great difficulty experienced by the CSIRO VLSI design team in achieving the successful completion of a full custom, 100,000 transistor nMOS chip using mask level CAD tools almost identical to those discussed in this thesis. The design could not be made fully operational after three years concentrated effort by a large, experienced and well coordinated design team and very significant investment by the Australian Government.

The most time consuming problem with mask level design of IC's is accurate layout. Circuit extractors, design rule checkers, simulators - all of the verification tools can also present problems if they do not perform their function efficiently, yet none of these tools presents as bad a design bottleneck as does the mask level layout problem.

Different methods, aimed at improving the efficiency of mask layout have been introduced in this thesis and designs using both methods have been fabricated. Comparisons of design time are not easy to make for many reasons. Variables such as the amount of time each day spent working on the design, designer experience, funding of the work which usually accounts for the sophistication of the hardware used (eg. multi-user machine versus dedicated work station), have all had an effect on the figures used in this conclusion. Nevertheless, some facts and figures are presented in an effort to allow some comparison of the tools to be made. Using an embedded layout language, BELLE, a 430 transistor nMOS circuit (the Signature Analyser) was created by a two man design team in three months with little simulated verification of the circuit operation. The design was 100% functional. Using a geometry editor, KIC, an 1860 transistor nMOS circuit (the Data Path Chip) was created by a two man design team in five months, and completely simulated to verify 100% device functionality.

Both design teams consisted of novice designers of similar backgrounds who were therefore considered equal in ability. The machine used was the same multi-user VAX 11/780 and the tools were almost identical except for the layout tool used. The amounts of time spent per day on the designs were very similar and so the conditions for comparison were quite equal. These two results show that the geometry editor certainly increases the efficiency of a design team, allowing a larger circuit design per man month as well as complete simulation of the design as final verification. The work by the author in making KIC operational at the University of Adelaide would seem to be rewarded by this result, which shows an approximate 250% improvement in design efficiency as measured by circuit complexity (number of devices) completed per man design month. It should also be understood that mask level layout is an error prone and time consuming process no matter what tool is used for the job.

The two layout tools discussed in detail in this thesis, BELLE and KIC, do not solve a fundamental problem with mask level layout tools. That is, the worst case minor modification to a cell could require a layout task just as significant, or more significant than the initial design task. The tool MAGIC has avoided this problem by approaching the mask level design task in a similar way to that in which VIVID has approached symbolic level design.

The current result of research into mask level tools for Computer Aided VLSI design has provided a complete set of original tools that are technology independent. The Technology Independence is perhaps the most important feature of these tools as it means that no matter what new technologies are introduced for IC design, as long as the technology description is based on a set of mask level geometrical design rules that can be described by CIF, very little work, in the order of tens of hours, would be required to create a totally new set of tools to handle designs using the new technology.

Mask level tools have a place in the full custom design of integrated circuits but are no longer the preferred option. Their main advantages are in meeting absolute mask level layout requirements that cannot be guaranteed by higher level tools as well as providing potentially the most precise simulation information, depending upon availability of accurate simulation parameters.

6.2 Symbolic Level Tools

The symbolic level of design has been presented in this thesis as the more efficient of the two design philosophies. The feature which underlies this increase in efficiency, virtual grid layout of circuit symbols that can be simulated at the symbolic level and compacted to mask level descriptions, has been detailed in Chapters III and V of this thesis. All of these advantages introduced by the VIVID symbolic level tool set can be seen in the comparison of the more efficient nMOS design experience, the Data Path Chip, with the CMOS design experience, the Adder.

The CMOS design, a 620 transistor design, was completed by the author alone in a period of two months. The design has been completely simulated at both the symbolic and the mask level. This is a further increase in design efficiency of approximately 50% in comparison to the time taken to design and simulate the Data Path nMOS chip using KIC and the mask level verification tools.

This design has clearly been completed in a much more efficient manner than either of the two nMOS designs and provides evidence of the significant advantages gained in using the symbolic level tools as opposed to the mask level tools. The VIVID tool set allows the design of IC's using two different technologies, nMOS and CMOS, and like the mask level tools, is technology independent. The VIVID technology independence feature is an improvement on mask level technology independence as all VIVID tools reference the Master Technology File which allows changes in technology to be completed for all tools in the one alteration process. On the other hand, changes in technology for the mask level tools must be completed on an individual basis extending the time required for creation of a tool set to handle a new technology.

Expanding and summarising the advantages of symbolic level design, VIVID allows the designer to ignore design rules and work with circuit primitives on a virtual grid rather than with geometric primitives on an absolute grid. This design feature allows great increases in layout speed due to the error free layouts produced. Once the ciruit editor ICE is understood, an experienced circuit designer would find it difficult to make an error in the circuit layout phase of the design as the designer would be dealing with primitives which are commonly known and understood. On the other hand, using geometric primitives to describe absolute mask layouts requires the designer to learn and account for geometric design rules as well as circuit layout during the layout phase of the design process. This added complexity greatly increases the time taken for layout. The virtual grid feature makes alterations to symbolic layouts a negligible problem, even in the worst case. In comparison, worst case alterations to mask layouts using an editor such as KIC can require a total redesign effort which could be greater than the initial design effort.

VIVID reduces the number of software interfaces between layout, and layout verification at the symbolic level. This feature helps improve the speed of the layout verification design phase. The major advantage that the VIVID system has over the mask level design tools in the layout verification design phase, however, results from the fact that VIVID has no design rule checking loop in this design phase. The compactor produces valid, design rule error free, CIF versions of the design.

VIVID has the added advantage of providing a software interface for designs to various design system Intermediate Formats facilitating designer interface to fabrication facilities accepting other than CIF as the design description format.

Layout needs that have yet to be met by higher level tools are the reliable automatic positioning of pads and automatic routing between sub-components of a VLSI circuit and between the circuit and pads for full custom circuits. This task is currently left to manual placement using layout editors, either layout languages or geometry editors. The VIVID system allows manual routing between sub-circuit elements at the symbolic level and will provide a solution to the pad placement problem by the use of a mask level geometry editor as well as the symbolic level editor for addition of pads and final completion of a design.

This second solution does however imply that mask level verification tools would also still be required. Mask level manipulation, no matter how trivial, can still be a source of design rule violations which could prove fatal to an IC design. Consequently a mask level design rule checker should also be provided. Because the pads could not be positioned at the symbolic level, simulated results for the circuit including the pads must also be completed at the mask level, another important verification step requiring a further mask level tool.

The VIVID system, running under VMS or UNIX, allows the most efficient completion of complex full custom VLSI circuits even though low level tools are still required to complete the design by allowing the placement and verification of pads. The pad placement problem is usually the least complex mask level routing task for an IC design and so is the least time consuming mask level layout task. The mask level simulation task can also be quite simple if a logic level simulation only is required, although timing and possibly some circuit simulation should also be attempted to provide a more detailed insight into the effects of the pads on the circuit.

6.3 CMOS MPC's

Using both the mask and symbolic level tools introduced in this thesis, the completion of VLSI circuits becomes a reality. The complexity of the VLSI design task is reduced significantly with the aid of the higher level tools and the fine details of pad placement can easily be handled by the mask level tools.

The Adelaide University mask level tools have been verified by in excess of twenty fabricated nMOS designs using the two MPC facilities mentioned in this thesis. Two of the designs used to verify these tools have been presented in Chapter IV. Using the mask level tools to verify the mask level results of the work done at the symbolic level using the VIVID system, circuits can be reliably checked to determine their level of functionality. The CMOS circuit designed in Chapter V has been shown to be 100% operational as determined by the mask level verification tools.

The verification work done in this thesis in conjunction with ISD and other researchers at Adelaide University has enabled negotiations to proceed with AWA with the intent of setting up a CMOS MPC process. This will happen in 1986 and should be one of the first if not the first commercial CMOS MPC fabrication processes in Australia.

6.4 Higher Level Tools of the Future

Two levels of CAD tools for VLSI circuit design have been defined and examined in this thesis. The first (lower) level, Mask Level CAD tools, uses mask geometries to define absolute circuit structures on silicon. The second (higher) level, Symbolic Level CAD tools, uses circuit elements to define circuits at an abstracted level which can automatically generate mask level descriptions of the circuit for fabrication purposes.

Table 3.2 in Chapter III identifies integrated circuit design levels. These levels can be directly associated with the levels of CAD tools used for IC design and so indicate the future directions that should be taken for CAD tool development for VLSI design work.

This thesis has discussed the analogy between levels of Programming Language and levels of CAD tool sets for IC design, as well as between generations of computing systems and scales of integration which allow a measure of complexity of both computing systems and IC designs. The fifth generation computing systems have been characterised by advances in software, Artificial Intelligence, rather than the hardware characteristics used for the preceeding generations. Artificial Intelligence, or Knowledge Based Systems, have begun to have their impact on IC design research. Tools using AI techniques are being developed to provide solutions to the search for CAD tool sets operating at the higher levels not examined in this thesis, in particular at the Register Transfer and Processor levels as defined by table 3.2

6.4.1 CADRE

One such tool set has already been referred to in this thesis. CADRE[57] is a system of cooperating VLSI design expert agents which allows automatic synthesis of hierarchical structural circuit descriptions into a set of full custom VLSI low level geometric mask patterns. CADRE operates by modelling the human design procedure as a collection of expert agents communicating through a central manager. The individual agents solve locally constrained problems while the manager ensures that they all work towards a good global solution.

The CADRE system is relevant to this thesis as it moves from the structural level (the input is an hierarchical structural description consisting of interconnected cells) to the symbolic layout level (the output is an hierarchical symbolic layout description). The CADRE output is passed on to a symbolic level design system for translation into physical mask layouts. This post processing of CADRE output means that detailed process design rules can be ignored by CADRE so it can concentrate on topology and interconnection problems.

The CADRE system requires a number of expert tools including floorplanning, leaf cell layout, critical path analysis, and global and local signal routing in an effort to satisfactorily model a human VLSI designer. Fig. 6.1 shows the CADRE VLSI design system exposing the individual agents used in the design process and how they are related to the manager and the user. CADRE provides flexibility in that an expert agent of CADRE may be rule based, algorithmic, or even human. This flexibility allows for the future development of the CADRE system.

CADRE is by no means complete, it is a long term research project aimed at exploring all problem areas of VLSI design and providing new solutions with the aid of Knowlege Based systems.

Taking a closer look at an individual knowledge based agent, current research at the University of Adelaide is aimed at creating one of the agents described by the CADRE system, a Knowledge Based Floor planner[95]. This floorplanner, FLOYD, is designed to translate hierarchical structural descriptions into floor plans for custom VLSI design. It uses expertise in structured top down design to create floor plans incorporating modules that may not necessarily have been completed at the time of floor plan creation. This allows global as well as internal module design constraints to be accounted for.

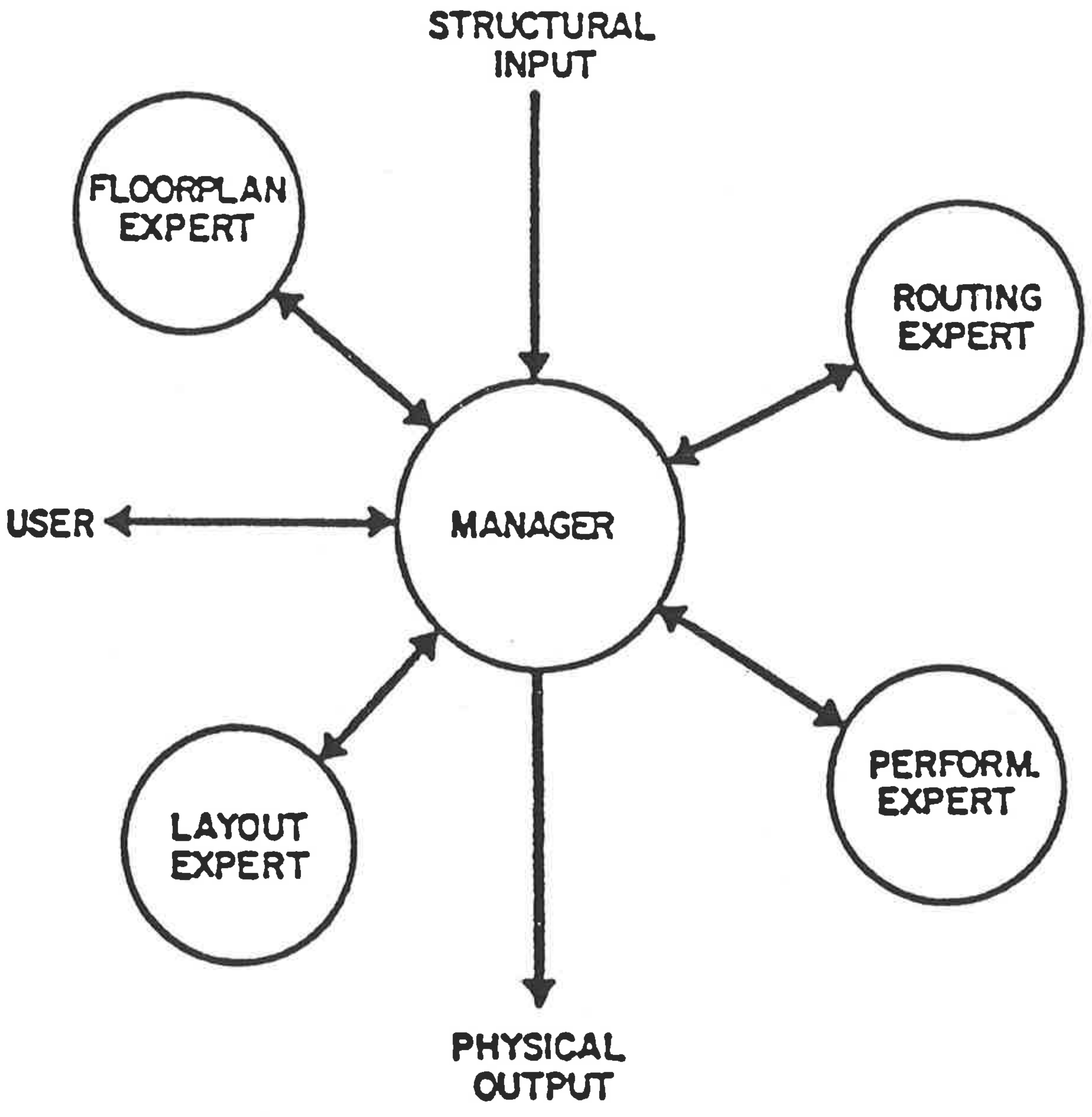


Figure 6.1: The CADRE design system, exposing the expert agents and their method of interaction with the global manager.

It is felt[95] that the top down planning approach yields floor plans of higher quality than conventional bottom up placement algorithms. FLOYD plans its activities by generating and relaxing constraints on module placements and interconnect. This planning strategy allows it to operate either without supervision or under the influence of designer specified constraints.

FLOYD provides an insight into the details of one of the CADRE expert agents. Similar research effort is ongoing into the other agents shown in Fig. 6.1. The work done in this thesis provides an introduction to the historical evolution of VLSI design tools to a level which is directly compatible with future tools proposed by the CADRE system.

6.4.2 **Silicon Compilation**

Automatic synthesis tools aid the VLSI circuit designer by automatically performing one or more levels of design translation. The *Silicon Compiler* is a term that has been used in the past to describe a number of different automatic synthesis tools. The ideal silicon compiler would be a tool which would translate all the way from a functional description to a full custom physical layout.

As yet, no such tool exists. Functional to structural translation is such a broad task that its complexity has not yet been handled. Existing tools that have been able to successfully translate functional descriptions into physical layout have at least had to assume a restricted structural domain[57].

This leads to the conclusion that that there will not be one single tool that can be described as a *Silicon compiler*. Rather, tools such as those discussed in this thesis and those foreshadowed by the CADRE system will provide a good basis from which to tackle the complexity problem involved in functional to structural translation, the main problem remaining in Silicon Compilation.

As a result, it could be reasonably expected that, using Artificial Intelligence techniques, tool sets or systems will be developed which consist of computer coordinated groups of individual expert tools which provide nearly optimal solutions to all of the individual problems involved in silicon compilation. The computer aided coordination of all of these tools will in turn provide a globally optimal solution which may be at the expense of non-optimal solutions to some local problems. It can therefore be expected that the term *Silicon Compiler* for full custom Integrated Circuits should be applied to such a system of sophisticated tools rather than used to describe a single CAD tool.

REFERENCES

[1] **J.Becker and J.Shive**, "The Transistor - A New Semiconductor Amplifier." In *Proceedings of the IEEE*, Vol. 72, No. 12, December 1984, pp 1696 - 1703.

[2] **R.Kahn**, "The Quest, A New Generation in Computing." In *IEEE Spectrum*, Nov. 1983, pp 36 - 41.

[3] **C.Tajnai**, "Fred Terman, the Father of Silicon Valley." In *IEEE Design and Test of Computers*, April 1985, pp 75 - 81.

[3.1] **G.Moore**, "VLSI: Some Fundamental Challenges". In *IEEE Spectrum*, April 1979, pp 30 - 37.

[4] **R.Colclaser**, *Microelectronics. Processing and Device Design.*

[5] **D.Moralee**, "Visions of the VLSI Future." In *Electronics & Power*, April 1982, pp 301 - 305.

[6] **R. Clarke**, "AUSMPC Designer Documentation." *CSIRO VLSI Program Report.*

[7] **L.Hon and C.Sequin**, *A Guide to LSI Implementation - Second Edition.* XEROX, *Palo Alto Research Center Report*, Jan. 1980.

[8] **C.Mead and L.Conway**, *Introduction to VLSI Design.* Addison Wesley, Reading, Mass. 1980.

[9] **I.Sutherland and C.Mead**, "Microelectronics and Computer Science". In *Scientific American*, Sept. 1977, pp 210 - 228.

[10] **C.Bell, J.Mudge and J.M^{c}Namara**, *Computer Engineering* . Digital Press, September 1978.

[11] **J.Mudge**, "VLSI Chip Design at the Crossroads." In *VLSI '81*, Edinburgh, Aug. 1981.

[12] **C.Sequin**, "Managing VLSI Complexity: An Outlook." In *Proceedings of the IEEE*, Vol. 71, No. 1, Jan 1983, pp 149 - 166.

[13] **N.Weste and B.Ackland**, "A Pragmatic Approach to Topological IC Design." In *Microelectronics '82*, Adelaide, South Australia, May 1982, pp 27 - 31.

[13.2] **D.Kollaritsch and N.Weste**, "A Rule Based Symbolic Layout Expert." In *VLSI Design*, Aug. 1984, pp 62 - 66.

[14.1] **P.Ivey**, "VLSI Design for Systems Applications." In *4th Australian Microelectronics Conference*, May 1985.

[14.2] **A.Kessler and A.Ganesan**, "Standard Cell Design: A Tutorial." In *IEEE Circuits and Devices Magazine*, Vol. 1, No. 1, Jan. 1985, pp 17 - 33.

[14.3] **VLSI Design Staff**, "Silicon Compilers Part 1: Drawing a Blank." In *VLSI Design*, Sept. 1984, pp 54 - 58.

[15] **A.Bell**, "The Implementation of VLSI Systems." In *Microelectronics '82*, Adelaide, South Australia, May 1982, pp 64 - 68.

[16] **L.Conway, A.Bell and M.Newell**, "MPC79: The Large Scale Demonstration of a New Way to Create Systems in Silicon." In *LAMBDA Magazine*, Vol. 1, No. 2., 1980.

[16.1] **J.Noonan and A.Williams**, "AUSMPC 5/82, Signature Analyser, Project C4." *AUSMPC 5/82 Design Report*, May 1982.

[16.2] **J.Noonan**, "AUSMPC 5/82 Signature Analyser Test Report." *AUSMPC 5/82 Test Report*, Sept. 1982.

[17] **J.Mudge and R.Clarke**, "Australia's First MPC Implementation System." In *Microelectronics '82*, Adelaide, South Australia, May 1982.

[18] **R.Clarke**, "A Summary of Results of Australian MPC's to Date." In *Creating Integrated Systems '83*, Adelaide, May 1983.

[19] **J.Lipman**, "VLSI Training - Enhancing the Silicon Broker/Foundry Concept." In *Microelectronics '82*, Adelaide, May 1982.

[20] **R.Lyon**, "The Optical Mouse." XEROX *Palo Alto Research Report*, Aug. 1981.

[21] **J.Ables and A.Hunt**, "A Parallel Pipelined VLSI Circuit for High Speed Digital Correlators." In *Creating Integrated Systems*, Adelaide, May 1983.

[22] **K.Eshraghian and D.Pucknell**, "Design for VLSI - An Undergraduate Teaching Program." In *Microelectronics '82*, Adelaide, May 1982.

[23] **J.Batali and A.Hartheimer**, "The Design Procedural Language Manual." In *MIT AI Laboratory VLSI Memo 80-31*, Nov. 1980.

[24] **B.Phillips**, "BELLE Documentation." CSIRO VLSI Group Documentation, Dec. 1981.

[25] **K.Eshraghian**, "A Guide to Interactive Computing for CMOS VLSI System Designers." University of Adelaide Internal Report, 1983.

[25.1] **A.Dickinson**, "RIDE." University of Adelaide Internal Report, 1982.

[26] **A.Dickinson and R.Woloszczuk**, "PLAN Documentation." University of Adelaide Internal Report, 1984.

[27] **K.Keller**, "KIC: A Graphics Editor for Integrated Circuit Design." In *Berkeley VLSI Tools*, Jul. 1982.

[28] **R.Woloszczuk and D.Pucknell**, "ET - A Friendly Interactive Editing Tool for I.C. Layout." In *The 4th Australian Microelectronics Conference, Sydney, May 1985.*

[28.1] **R.Woloszczuk**, "M.Eng.Sci. Thesis writing up, Adelaide University."

[29] **J.Ousterhout**, "Editing VLSI Circuits with CAESAR." In *Berkeley VLSI Tools*, Jul. 1982.

[29.1] **J.Ousterhout, G.Hamachi, R.Mayo, W.Scott and G.Taylor**, "The Magic VLSI Layout System." In *IEEE Design and Test of Computers*, Feb. 1985, pp 19 - 30.

[30] *Visual 500 Reference Manual*, Visual Technology Inc., Tewksbury MA., Sept. 1982.

[31] *The Summagraphics Bitpad Manual.*

[32] *AED 512 Colour Graphics/Imaging Terminal, Users Manual*, Advanced Electronics Design Inc., Sunnyvale, California, 1981.

[33] *Vectrix VX Series Graphics System User's Manual*, Vectrix Corp., Greensboro N.C., July 1983.

[34] *Mouse Systems Corp. M-1 Mouse Technical Reference Manual*, Mouse Systems, Santa Clara Ca., March 1981.

[35] **B.Kernighan and D.Ritchie**, *The C Programming Language*. Prentice-Hall, 1978.

[36] **J.Noonan**, *Research Seminar*, University of Adelaide, Sept. 1983.

[37] **C.Baker**, "Artwork Analysis Tools for VLSI Circuits." *Master's Thesis*, MIT 1980.

[38] **D.Noice, R.Mathews and J.Newkirk**, "A Polygon Package for Analysing Integrated Circuit Designs." In *VLSI Design*, third quarter, 1981.

[39] **A.Noble**, "ADRIC - A Polygon Based Design Rule Checker." University of Adelaide Internal Report, 1982.

[40] **T.Whitney**, "A Hierarchical Design Rule Checking Algorithm." In *Lambda Magazine*, first quarter, 1980.

[41] **M.Arnold and J.Ousterhout**, "LYRA:- A New Approach to Geometric Layout Rule Checking." In *Proceedings of the 19th Design Automation Conference*, 1982.

[42] **D.Hartley**, "ROWAN - An Hierarchical, Corner Based Design Rule Checker for VLSI Designs." University of Adelaide Internal Report, 1983.

[42.1] *CHECK*, Integrated Silicon Design Documentation, Adelaide, S.A., 1985.

[43] **D.Fitzpatrick**, "CIFPLOT." In *Berkeley VLSI Tools*, Jul. 1982.

[44] **D.Fitzpatrick**, "MEXTRA: A Manhattan Circuit Extractor." In *Berkeley VLSI Tools*, Jul. 1982.

[45] **A.Gupta and R.Hon**, "HEXT: An Hierarchical Circuit Extractor." In *Journal of VLSI and Computer Systems*, Vol. 1, No. 1, Spring 1983.

[46] *NET: Network Circuit Extractor*, Integrated Silicon Design Documentation, Adelaide, 1985.

[47] **A.Gupta**, "ACE: A Circuit Extractor." In *Proceedings of the 20th Design Automation Conference*, 1983.

[48] **N.Weste and K.Eshraghian**,*Principles of CMOS VLSI Design. A Systems Perspective.*, Addison Wesley, 1985.

[49] **M.Shand**, "Hierarchical VLSI Artwork Analysis." In *VLSI '85* Japan, 1985.

[50] **D.Fitzpatrick**, "Circuit Analysis from CIF Layouts." In *Berkeley VLSI Tools*, Jul. 1982.

[51] *ELEC: Electrical Rules Checker Users Manual*, Integrated Silicon Design Documentation, Adelaide, 1985.

[52] **A.Vladimirescu and S.Liu**, "The Simulation of MOS Integrated Circuits Using SPICE 2." In *ERL Memorandum*, Feb. 1980.

[53] **D.Fitzpatrick**, "MOSSIM." In *Berkeley VLSI Tools*, Jul. 1982.

[54] **M.Pope**, "PROBE Users Manual." University of Adelaide Internal Report, 1985.

[55] **A.Dickinson**, "TICTOC: A VLSI System Language and Simulator." University of Adelaide Internal Report, Mar. 1984.

[56] *VIVID User's Manual, Version 1.0*, Documentation Supplied by the Microelectronics Center of North Carolina.

[57] **B.Ackland, A.Dickinson, et al.**, "CADRE - A system of Cooperating VLSI Design Experts." In *IEEE International Conference on Computer Design*, Oct. 1985.

[58] **J.Rosenberg and N.Weste**, "ABCD - A Better Circuit Description." In *MCNC Technical Report No. 82-01*.

[59] **Weste N. H. E.**, "MULGA. An Interactive Symbolic Layout System for the Design of Integrated Circuits." In *BSTJ 60(G) pp 823 -857, Jul-Aug 1981*.

[60] **Weste N. H. E.**, "Virtual Grid Symbolic Layout." In *Proc. 18th D.A.C. pp 225-233 June 1981.*

[61] **Trimberger S., Rowson J.**, "RIOT - A Simple Graphical Assembly Tool." In *Proc. 19th D.A.C. pp 23-29 June 1982.*

[62] "VIVID System Manual - ICE Interactive Circuit Editor Designer Reference Version 1.0." *An MCNC manual.*

[63] "VIVID System Manual - VIVID System Overview." *An MCNC Manual.*

[64] **Buchanan I.** "Modelling and Verification in Structured IC Design." *Phd. Thesis, University of Edinburgh.*

[65] **Ackland and Weste.** "An Automatic Assembly Tool for Virtual Grid Layout." In *Proc. VLSI '83, Aug. 1983, Norway.*

[66] "The SEE User's Manual." *An ISD Manual.*

[67] **Hayes J.** "Computer Architecture and Organisation." *M^{c}Graw Hill, 1978.*

[68] **Johannsen D.** "Bristle Blocks: A Silicon Compiler." In *Proc. 16th D.A.C. June 1978, pp 310-313.*

[69] **Denyer P., Renshaw D., Bergmann N.** "A Silicon Compiler for VLSI Signal Processors." In *Proc. ESSCC 1982.*

[70] **Siskind J., Southand J., Crouch K.** "Generating High Performance VLSI Designs from Succinct Algorithmic Descriptions." In *Proc. of Conf. on Advanced Research in VLSI, June 1982, MIT pp 28-40.*

[71] **Hoenielsen B. & Mead C.** "Fundamental Limitations in Microelectronics - I. MOS Technology." In *Solid State Electronics, 1972, Vol. 15, pp 819-829.*

[72] "CSIRONET Services." In *Services Note 2, Mar. 1982, CSIRO Division of Computing Research.*

[73] **Lauder P., Kummerfeld R. Els R.** "ACSNET - The Australian Alternative to UUCP." In *AUUGN Vol. 5 No. 4, pp 15 - 20.*

[74] **Maxwell P.** "Design for Testability." In *CSIRO VLSI Program Technical Report, VLSI-TR-83-1-2.*

[75] **Knuth D.** "The TEXBook." *Addison Wesley, 1984.*

[76] **Webber D. Dunis V.** "Design of a Control Unit for a 4-Bit Microprocessor." *University of Adelaide EEE Dept. Final Year Report.*

[77] "Test of JMRC Fabricated Control Unit." To come.

[78] **Lee and Loo.** "Data Path Chip for a Four Bit Microprocessor." *University of Adelaide EEE Dept. Final Year Report, 1982.*

[79] **Loja A.** "Testing of a Data Path Chip." *University of Adelaide EEE Dept. Final Year Report, 1982.*

[80] **Andrerwartha L. Walsh A.** "Design of an NMOS Control Unit." *University of Adelaide EEE Dept. Final Year Report, 1983.*

[81] **Liebelt M. J.** "The Testability of Microprocessor Systems." *M.Eng.Sci. Thesis, Adelaide University, 1981.*

[82] **Eshraghian K.** "Vehicle Traffic Monitoring." *M.Eng.Sci. Thesis, Adelaide University, 1977.*

[83] **Eshraghian K.** "Electromagnetic Traffic Sensing and Surveillance." *Phd. Thesis, Adelaide University, 1980.*

[84] **Rockliff J.** "The Implementation of Testability Strategies in a VLSI Circuit." *M.Eng.Sci. Thesis, Adelaide University, 1986.*

[85] **Barter C.** "The Screen Editor Ludwig - User's Manual and User Guide." *University of Adelaide CS Dept.*

[86] **Eshraghian K. et al.** "The Transform and Filter Brick." In *VLSI '85, Tokyo Japan 1985.*

[87] **Eshraghian K. et al.** "A New CMOS VLSI Architecture for Signal Processing." In *VLSI PARC, Melbourne 1984.*

[88] **Oppenheim A., Schaefer.** "Digital Signal Processing. " *Prentice Hall, New Jersey, 1975.*

[89] **Eshraghian K., Zyner G.** "Multiplier/Divider Designs for a VLSI Signal Processing System." In *IREECON, Sydney 1985.*

[90] "The AWA MPC Process." *An ISD/Adelaide University EEE dept. CMOS Design Experience.*

[91] "GPLOT.DOC - An online Users Guide" *Adelaide University Plotting Software Manual.*

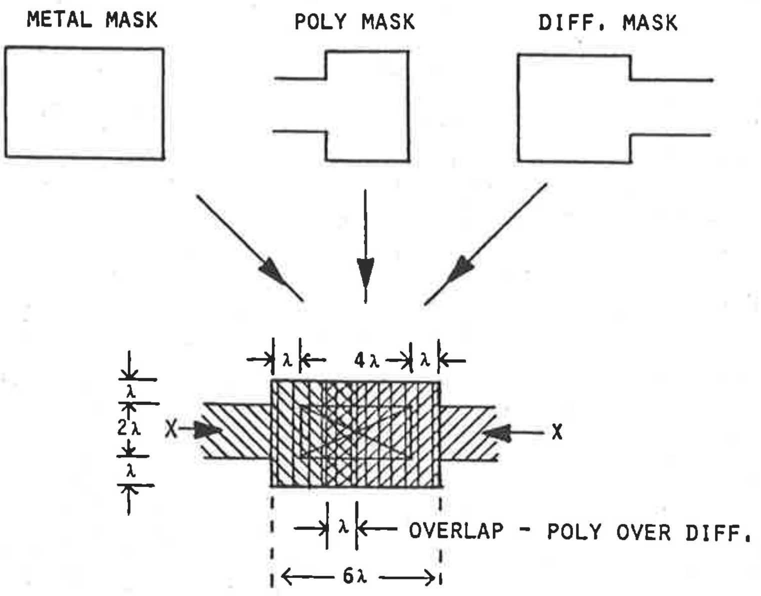
[92] **Noonan J.** "CMOS Pad Design." *A report sent to MCNC Dec. 1984.*

[95] **Dickinson A.** "Floyd: A Knowledge Based Floor Plan Designer." *A Summary for ICCD '86.*

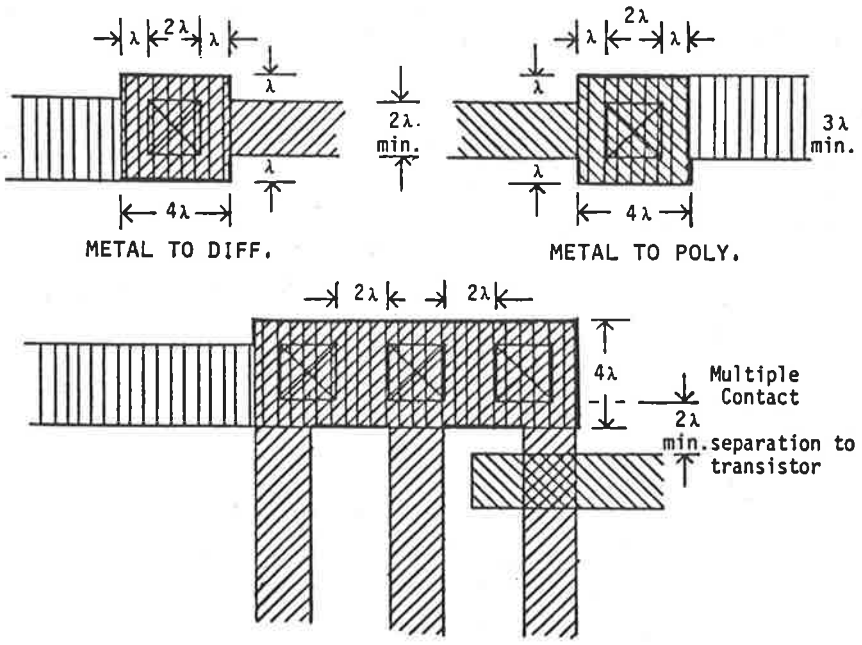
APPENDIX A

THE MEAD AND CONWAY nMOS DESIGN RULES

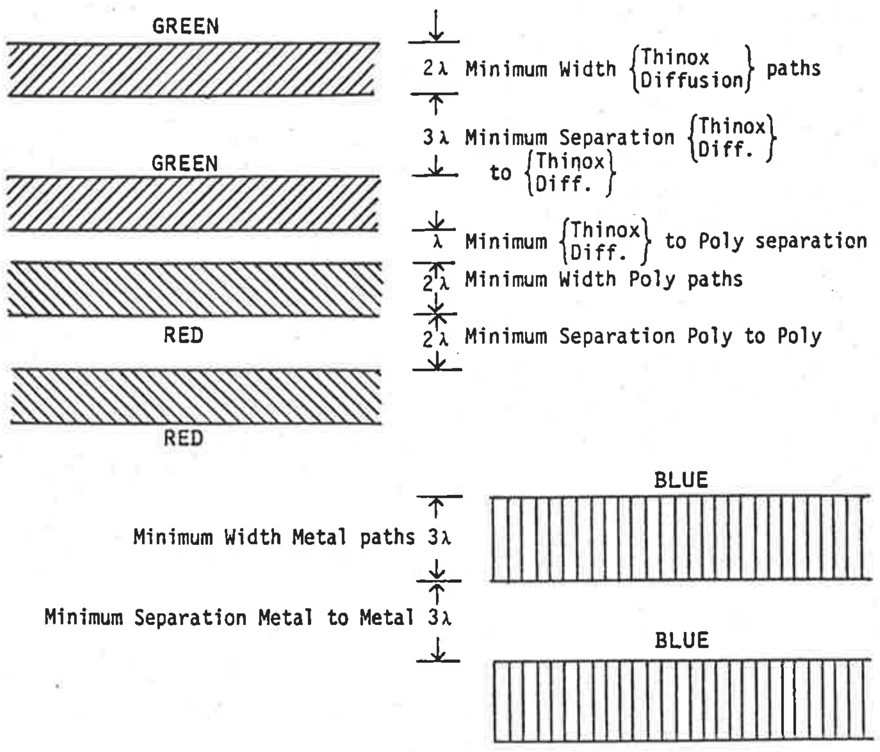
These rules use the same relationships as those presented in the book by Mead and Conway[8].



Butting Contact (Polysilicon to Diffusion via Metal)

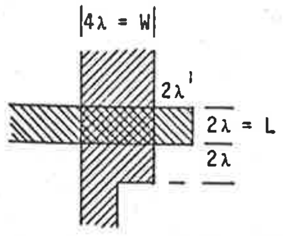


Design Rules for Contact Cuts (Metal to Polysilicon or Diffusion)

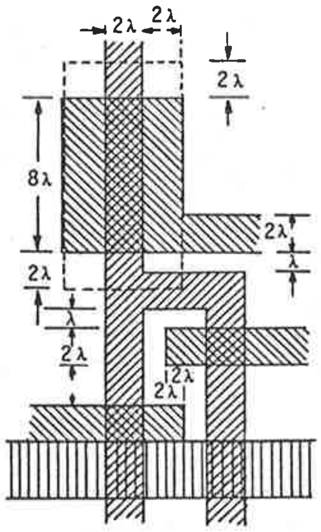

GREEN
2λ Minimum Width {Thinox Diffusion} paths
3λ Minimum Separation {Thinox Diff.} to {Thinox Diff.}
GREEN
λ Minimum {Thinox Diff.} to Poly separation
2λ Minimum Width Poly paths
RED
2λ Minimum Separation Poly to Poly
RED
BLUE
Minimum Width Metal paths 3λ
Minimum Separation Metal to Metal 3λ
BLUE

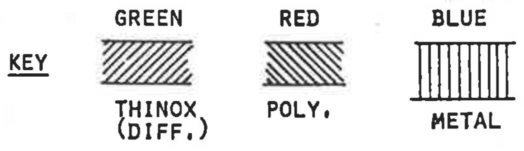
Conducting Path Width and Separation Design Rules

(*Note*: The Thinoxide (Thinox.) mask defines the regions where Diffusion (Diff.) will take place in areas *not* covered by Polysilicon (Poly).



Pass transistor L:W ratio = 1:2





APPENDIX B

THE BELLE1 MANUAL

1.0 PREFACE

The purpose of this document is to introduce you to using the VLSI design tool BELLE1. BELLE1 is a Pascal embedded design tool for the creation of Nmos and Cmos designs in CIF code, but makes use of the high level Pascal language to create the geometry.

2.0 BELLE1 A USERS VIEWPOINT

2.1 INTRODUCTION

Belle1 is a procedural integrated circuit and mask layout language embedded in the Pascal programming language. Embedding allows the power of a high level programming language to be used to aid in the description of integrated circuit mask layouts, without the need to develop an entirely new programming language.

Belle1 is composed of a set of pascal procedures which can be used to describe the various structures in a layout. Belle1 generates its output in CIF (Caltech Intermediate Form) [Hon and Sequin, 1980], which is a low level description of the circuit. CIF is the standard data format chosen for communication of designs during the MPC (Multiple Project Chip).

All dimensions used in Belle1 refer to the basic dimensions Lambda. Only integer values of lambda are allowed when calling Belle1 procedures.

Most work can be done using orthogonal geometry. However, Belle1 does allow 45 degree wires to be used after the procedure set45 has been called.

Belle1 may be used to generate either nMOS or P-well cMOS layouts. Only one technology may be used in any given layout.

2.2 BELLE1 FUNCTIONS AVAILABLE

The built in functions of Belle1 are :

2.2.1 THOSE COMMON TO NMOS AND CMOS

SETTECH

Syntax:	settech(newtech);
Description:	Changes the technology base for the current layout to newtech, where newtech is one of CMOS or NMOS. NMOS is the default technology.
Example:	settech (CMOS);

or

settech (NMOS);

SETSYMNO

Syntax:	setsymno(nextsym);
Description:	Nextsym is an integer that is used to label the next symbol in the CIF program generated by BELLE1.
Example:	setsymno(5000);

DEFINE

Syntax:	define('newname');
Description:	defines the start of a symbol definition and gives it the name 'newname'.
Example:	define('transistor');

```
define('transistor');
{ define the symbol 'transistor' }
{ proceed with the definition i.e. boxes
and wires that form 'transistor' }
enddef; { end the definition }
```

ENDDEF

Syntax:	enddef;
Description:	defines the end of a symbol definition
Example:	define('transistor');

```
define('transistor');
{ define the symbol 'transistor' }
enddef; { end the definition }
```

DRAW

Syntax:	draw('newname',x,y);

Description: draws the symbol previously defined as 'newname' at a position (x,y)

Example: draw('transistor',5,10);

{draws the geometry held in the symbol definition 'transistor' at the point 5,10}

```
     _^

   Y |

     |

     |

     |     +-------+

     |     |       |

     |     | tran  |

     |     |       |

  10 |     *-------+

     |     _^

     |   (5,10)

     |

   0 +------------------>

     0     5         X
```

MX

Syntax: draw('newname',x,y);mx;

Description: mirrors the image of the symbol being drawn about the vertical line passing through the point specified in the draw statement.

Example: draw('transistor',5,10);mx;

```
     _^
```

```
        Y |

          |

          |

          +------+ - - -+

          |      |

          | tran |      |

          |      |

       10 +------X - - -+

          |      _^

          |    (5,10)

          |

        0 +------------------>

          0      5         X
```

MY

Syntax: draw('newname',x,y);my;

Description: mirrors the image of the symbol being drawn about the horizontal line passing through the point specified in the draw statement.

Example: draw('transistor',5,10);my;

```
          _^

        Y |

          |

          |

          |      +- - - +

          |

          |      |      |
     |
 10  |(5,10)*------+
     |      |      |
     |      | tran |
     |      |      |
  0  +------+------+----->
     0      5          X
```

ROT

Syntax:	draw('newname',x,y),rot(angle);
Description:	rotates the x-axis of the symbol being drawn to the specified angle (measured anticlockwise)
Example:	draw('transistor',5,10);rot(180);
NOTE:	angle must be an integer multiple of 90 degrees unless set45 procedure has been called.

```
     -^
   Y |
     |      + - - -+
     |     .   .180 degrees
     |   .  |   .  |
     |  V        .
 10  +------* - - -+
     |  n   |(5,10)
     |  a   |
     |  r   |
  0  +--t---+----------->
     0      5          X
```

LAYER

Syntax:

Description: defines the layer on which the succeeding geometry is to be defined. The layers available are:

Diffusion	NMOS
Poly	NMOS _& CMOS
Implant	NMOS
Contact	NMOS _& CMOS
Metal	NMOS _& CMOS
Glass	NMOS _& CMOS
Buried	NMOS
Pwell	CMOS
Thinox	CMOS
Pplus	CMOS

Example:

```
layer(thinox);
{ geometry written after this point is on
the thinoxide layer }
```

BOX

Syntax: box(x1,y1,x2,y2);

Description: draw a rectangular box with corners (x1,y1) and (x2,y2)

Example:

```
box(2,1,6,4);
{ draws a box with bottom left corner at
(1,2) and top right corner at (6,4) }
```

```
_^
Y|
4|   +-------+
 |  |   |
3|  |   |
 |  |   |
2|  |   |
 |  |   |
1|  +---+
 |
 +-------->
    2 4 6  X
```

WIRE

Syntax: wire(width,x,y);

Description: defines a wire of width "width", starting at (x,y). The wire call must be followed by one or more of the following procedures:

```
X(xposn);       { continues the wire to the
                  point (xposn,y) }
Y(yposn);       { continues the wire to the
                  point (x,yposn) }
dx(xlength);    { continues the wire to
                  the point (x+xlength,y) }
dy(ylength);    { continues the wire to the
                  point (x,y+ylength) }
xy(xposn,yposn); { continues the wire to
                   the point (xposn,yposn)
                   at an angle of 45 degrees }
dxy(xlength,ylength); { continues the wire
          to the point (x+xlength,y+ylength)
                   at an angle of 45 degrees }
```

If a wire is to have an odd number of lambda units width, then the centre coordinates must be specified between grid points, so as to enable the wire edges to run along lambda grid lines and not between them.

Example: Wire(2,2,3);x(5);dy(5);

{ starts a wire of width 2 at (2,3), extends it to the point (5,3) then extends it differentially to 5,8) }

```
Y _^
  |               /-----\
  |              /       \
8 |              |   X   |
  |              | (5,8) |
7 |              |       |
  |              |       |
6 |              |       |
  |              |       |
5 |              |       |
  |              |       |
4 |              |       |
  |              |       |
3 |   /----------+       |
  |   |            (5,3) |
2 |   |  X(3,2)       X  |
  |   |                  /
1 |    \----------------/
  |
0+---------------------------->

 0  1   2   3   4   5   6   7  X
```

NODELABEL

Syntax:

Description: names the electrical node at the position and layer specified. Used for simulation programs, most of which at least require Vdd and GND to be labelled.

Layers that can have legal nodes are:

Diffusion	NMOS
Poly	NMOS _& CMOS
Metal	NMOS _& CMOS
Thinox	CMOS

Example:

```
nodelabel('vdd',275,18,metal);
{ labels the node at (275,18) on the metal
layer with the name VDD }
```

COMMENT

Syntax: comment('comment string');

Description: inserts a comment into the CIF file.

Example:

```
comment('start a new definition');
{ produces
        start a new definition
  in the CIF file }
```

SETNOEND

Syntax: setnoend;

Description: suppresses generation of the end statement in the CIF output files. For use when different CIF files are to be appended.

Example: setnoend;

SET45

Syntax: set45;

Description: allows the use of 45 degree lines. However the CIF generated is incompatible with some service programs. (DRC etc.).

Example: set45;

EXISTING

Syntax: existing('symbolname');

Description: the function returns true if the symbol named has already been defined.

Example:

```
if existing('transistor') then
    begin { if }
    .
    .
    end; { if }
{ the compound statement bounded by begin
and end; will be executed only if transistor
has been defined previously }
```

BOUNDINGBOX

Syntax: boundingbox('symbolname',x1,y1,x2,y2);

Description: when boundingbox is called, the boundingbox of the symbol named is returned in the form (x1,y1) lower left coordinates (x2,y2) upper right coordinates

Example:

```
boundingbox('transistor',xlow,ylow,xhigh,yhigh);
                {would return the bounding box coordinates
                 in the variables xlow, ylow, xhigh, yhigh }
```

IMPORTSYMBOLS

Syntax:

Description: the filew must contain the bounding boxes of the geometry which is external to the BELLE1 program. The file will usually be the .SLB file generated by GETSYMBOL. Importsymbols must be called before any geometry is defined in the program.

Example:

```
importsymbols('pads.slb');
{ allows the BELLE1 compiler to use the
external symbols outlined in PADS.SLB }
```

ABORT

Syntax: `abort('errormessage');`

Description: prints the string 'errormessage' then ceases to execute, returning control to the user.

Example:

```
abort('symbol does not exist');
{ writes to SYS$OUTPUT
      symbol does not exist
then ceases execution }
```

2.2.2 THOSE EXCLUSIVELY NMOS

DIFFCUT

Syntax: diffcut(x,y);

Description: provides all the geometry for a cut between the metal and diffusion layers, the cut centre being at (x,y)

Example: diffcut(0,0);

```
+---------------+
|               |
|   +-------+   |
|   | \   / |   |
|   |   X (0,0) |
|   | /   \ |   |
|   +-------+   |
|               |
+---------------+
```

POLYCUT

Syntax: polycut(x,y);

Description: provides all the geometry for a cut between the metal and the poly layers, the cut centre being at (x,y)

Example: polycut(0,0);

```
+---------------+
|               |
|   +-------+   |
|   | \   / |   |
|   |   X (0,0) |
|   | /    \ |  |
|   +--------+  |
|               |
+---------------+
```

BUTTCONTACT

Syntax: buttcontact(x,y,angle);

Description: supplies the geometry for a cut between poly, diffusion and metal. The cut is centred at (x,y) and the angle of the poly section to the positive X-axis must be supplied.

Example: buttcontact(0,0,90);

```
+---------------+
|  poly         |<.
|   +--------+  |  .
|   |\      /|  |    .
|   | \    / |  | 90 degrees
|   |  \  /  |  |      .
+---+  X (0,0)-+  -----V--
|   |  / \   |  |
|...| /   \  |...|
|   |/     \|   |
|   +--------+  |
|  diffusion    |
+---------------+
```

BURIEDCONTACT

Syntax: buriedcontact(x,y,angle);

Description: supplies the geometry for a buried contact between diffusion and poly. The cut centre is positioned at (x,y) and the angle of the poly section to the positive X-axis must be specified.

Example: buriedcontact(0,0,90);

```
+-------------------+
| poly _& buried    |
|                   |
|    +========+     |<.
|    H  poly  H     |     . 90 degrees
|    H  diff  H     |      .
|    H buried H     |       .
+----H----*---H----+      --V-->
.    H  (0,0) H    .          X
.    H diff   H  buried layer all over
.    H        H    .
.....+========+.....
```

PULLUP

Syntax: pullup(length,x,y,angle);

Description: places a depletion transistor with length to width ratio of "length" at the position (x,y) which is the centre of the boundary between source and the gate. The angle refers to the drain direction.

Example: pullup(8,0,0,0);

2.2.3 THOSE EXCLUSIVELY CMOS

MP (metal poly contact)

Syntax: mp(x,y);

Description: provides all the geometry for a cut between metal and polysilicon layers, the centre of the cut being at (x,y).

Example: mp(0,0);

```
+---------------+
|               |
|   +-------+   |
|   | \   / |   |
|   |   X (0,0) |
|   | /   \ |   |
|   +-------+   |
|               |
+---------------+
```

MD (metal thinox contact)

Syntax: md(x,y);

Description: provides all the geometry for a cut between metal and thinoxide layers, the centre of the cut being at (x,y).

Example: mp(0,0);

```
+---------------+
|               |
|   +-------+   |
|    | \   / |    |
|    |   X (0,0) |
|    | /   \ |    |
|    +-------+    |
|                 |
+-----------------+
```

MDP (metal P+ diffusion contact)

This is one of two types of metal thinoxide contacts
Here, thinox is coincident with pplus layer and hence
is p+ diffusion, representing the source or drain
region of a p-device.

Syntax: mdp(x,y);

Description: describes all the geometry for a cut between
metal and thinox (p+ diffusion) layers for
connection of a p-device source or drain
region to a metal strap, the cut centre being at
(x,y).

Example: mdp(0,0);

```
+ - - - - - - - - - - - - +
|          p+ thinox      |
     +------------+
|    | metal      |       |
     | +-------+  |
|    | | \   / | |        |
     | |  (0,0)  |
|    | | /   \ | |        |
     | +-------+  |
|    |            |     |

     +------------+

|                       |


+ - - - - - - - - - - - - +
```

MDN (metal/n+ diffusion contact)

This is one of the two types of metal/thinox contact. Here thinox is not coincident with pplus region hence is n+ diffusion. The thinox however is in a pwell region and hence is a source or drain region for a n-device.

Syntax: mdn(x,y);

Description: describes all the geometry for a cut between metal and thinox (n+ diffusion) layers for connection of an n-device source or drain region to a metal strap, the cut centre being at (x,y).

Example: mdn(0,0);

```
. . . . . . . . . . . . . .

.                          .

.          n+ thinox       .

.    +------------+        .

.    | metal      |        .

.    | +-------+  |        .

.    | | \   / | |        .

.    | |  (0,0)  |        .

.    | | /   \ | |        .

.    | +-------+  |        .
.    |            |     .
     +------------+
.                       .
.                       .
.                       .
. . . . . . . . . . . . . .
```

VSS (metal/p+ diffusion/pwell contact)

Here the metal will be the Vss rail and will contact the p+ region in the pwell. The p+/p contact ensures a good ohmic contact to tie the pwell to Vss (ground). Tying the Pwell to Vss and the n substrate to Vdd helps prevent latch up problems.

Syntax: Vss(x,y);

Description: provides all the geometry for an ohmic connection of the pwell to Vss. The cut is centred at (x,y).

Example: vss(0,0);

```
. . . . . . . . . . . . . . .
.        p well               .
.   + - - - - - - - - - - +   .
.   |  p+ mask            |   .
.   |  +------------+     |   .
.   |  |  +-----+   |     |   .
.   |  |  |\   /|   |     |   .
.   |  |  |  X(0,0)|  |   |   .
.   |  |  |/   \|   |     |   .
.   |  |  +-----+   |     |   .
.   |  +-diffusion-+      |   .
.   |                     |   .
.  + - - - - - - - - - - +  .

.                           .

. . . . . . . . . . . . . . .
```

VDD (metal/n+ diffusion/n- substrate contact)

Here the metal will be the Vdd rail and will contact the n+ region in the n- substrate. This n+/n contact ensures a good ohmic contact to tie the n- substrate to Vdd.

Syntax: Vdd(x,y);

Description: describes all the geometry for an ohmic connection of the n- substrate to Vdd. The cut is centred at (x,y).

Example: Vdd(0,0);

```
+------------+
|  metal     |
|  +------+  |
|  | \  / |  |
|  |  (0,0)  |
|  | /  \ |  |
|  +------+  |
|            |
+------------+
```

VDDSPLIT (ohmic and rectifying contacts)

This is one of the two possible types of split contact. This contact connects the n- substrate (via n+ region) and the p-type source region to Vdd at the same time. It is a very compact contact and finds wide practical use (e.g. for

Pullup device of inverter pair).

Syntax: Vddsplit(x,y,angle);

Description: provides all the geometry necessary to tie the n- substrate and p+ diffusion (source) to Vdd at one time. The cut "centre" is positioned at (x,y) and the angle (integer multiple of 90) of the ohmic (n+ diffusion) end to the x axis must be specified;

Example: vddsplit(0,0,90);

VSSSPLIT (ohmic and rectifying contacts)

This is one of two possible types of split contact. This contacts the p-well (via p+ region) and an n-type source region to vss at the same time. It is a very compact contact and finds wide practical use (e.g. for pull-down device of inverter pair).

Syntax: vsssplit(x,y,angle);

Description: provides all the geometry necessary to tie the p-well and n+ diffusion (source) to Vss at the one time. The cut "centre" is positioned at (x,y) and the angle of the rectifying (n+ diffusion) end to the x axis must be specified.

Example: vsssplit(0,0,0);

TRAN (minimum size gate transistor)

Syntax: tran(x,y,angle);

Description: provides the geometry for a transistor with a minimum size gate, the centre of which is at (x,y) and oriented such that the angle of the thin oxide section to the x axis is the specified angle.

Example: tran(0,0,90);

TRANX (transistor with variable thinox width)

Syntax: tranx(x,y,width,angle);

Description: provides the geometry for a transistor with centre at (x,y), thinox width of "width" and angle specified is angle between the thinoxide section and the x-axis. Width must be even!!

Example tranx(0,0,6,90);

TRANC (transistor with contacts)

Syntax: tranc(x,y,angle);

Description: provides the geometry for a minimum size transistor with contacts at both ends. Transistor centre is at (x,y) and angle specified is angle of thinoxide relative to x-axis.

Example: tranc(0,0,90);

2.2.4 THE GETSYMBOL PROGRAM

An auxillary program to Belle1 is the GETSYMBOL program. This program is for use in conjunction with the IMPORTSYMBOLS function within Belle1. It is used to generate a header file of type required by IMPORTSYMBOLS. GETSYMBOL operates on a CIF file and produces two files. One file contains the names of all symbols, their symbol numbers and their bounding boxes. The other file contains information on CIF node-labels , their name, location and layer type.

An example output of the symbol library file, FILE.SLB looks like:

```
901 MDN -500, -500, 500, 500
902 MP -500, -500, 500, 500
```

Records have the format: Sym-No. Sym-Name. Boundary Box (lower left, upper right corners).

All coordinates are given in hundredths of a micron. The Output file of GETSYMBOL is of the type ".slb". This file, when operated on by IMPORTSYMBOLS in the BELLE1 Program, enables the CIF file to be produced containing all the geometry defined in the BELLE1 program. It also contains calls to the external definitions whose bounding boxes are supplied by the .slb file. When the CIF file has been generated the file of external definitions must be appended to it to give a complete CIF description of the geometry.

In this way it is possible to use predefined geometries (Pads, PLA's etc.) from the relevant library.

2.3 USE OF PASCAL WITH REFERENCE TO BELLE1

When using the design language BELLE1, it is possible to use the basic Pascal programming structures (some of which are introduced in this section).

2.3.1 STRUCTURE

A simplified structure of a Belle1 Pascal program could be:

```
 [inherit('belle1_dir:belle1.pen')]          {include belle1
definitions}
 program <program name>(input,output);
 const <constant declarations>;
 var <variable declarations>;
 <procedure declarations>;
 begin
    open_ciffile('<file_name>');
    settech(nmos);     {or cmos}
    <MAIN PROGRAM>;
    .
    .
    close_ciffile;
 end.
```

2.3.2 GENERAL

Identifier (variable names, procedure names etc.) must start with a letter and may be any length of letter and numbers. All Pascal statements must be separated by a semicolon, as lines have no separating significance.

A variable can be assigned a value by

```
<variable identifier> := <value>;
```

e.g.

```
                A:=10;
                B:=5;
```

Comments can be entered into the Pascal program by enclosing the comments by { }

e.g.

```
        {Start of Definition}
```

2.3.3 CONSTANTS

If an identifier is to hold the same value throughout the program, it can be declared in the constant section at the start of the program:

e.g.

```
              length = 57;
                 K = 8;
```

2.3.4 VARIABLES

If an identifier will be required to change its value during a program, if must be declared in the VAR section and its type (integer, real, char etc) be declared:

e.g.

```
          VAR WIDTH: integer;
                H: real;
```

At a later stage these variables may be assigned values, e.g.

```
                WIDTH:= 10;

                 H:= 5.76;
```

And at a later stage these can be assigned different values:

e.g.

```
             WIDTH:= WIDTH + 1;

                  H:= 6.2;
```

2.3.5 PROCEDURES

Procedures must be of the same structure as that described in "structure" ,except that they end in a semicolon. They are declared using the PROCEDURE statement followed by the parameter list in brackets.

Procedures can be called from other procedures, or from the main program.

e.g.

```
VAR LENGTH: integer;           { main program declarations }

PROCEDURE TRANS(SIDE:integer);  { procedure declarations }

CONST B = 5;

begin

    .

    .                                 { body of procedure }

    .

end; {trans}                          { note the semicolon }

    .
```

```
   .                    { other procedure declarations }

   .

begin {main}                 { start of main program }

   .

   .

   .

   trans(LENGTH);{ call procedure with length as parameter }

   .

   .

   .

end. {main}                    { end of program }
```

2.3.6 DESIGN USING BELLE1

Typically, designs are performed using Belle1 in the following manner:

(1) Beginning with an initial sketch or stick diagram of the circuit being designed. The circuit must be laid out on graph paper to aid in digitisation. This process usually takes a couple of attempts to generate an acceptable layout.

Hints for Layout

(a) Use coloured pens to help distinguish between masks.

(b) Graph paper with 2mm grid spacing is a good size. (2mm represents 1 lambda produces quite legible layouts).

(c) Liquid paper or equivalent is almost a necessity to remove those occasional mistakes.

(d) Always check and recheck your layout for design rule violations, circuit correctness etc. Some violations are often very difficult to detect. It is wise to have at least one other person check your layout since a fresh outlook may spot elusive errors.

(e) Some layouts, such as address decoders, are highly repetitive with only slight variations from section to section of the layout. In these cases, it is not necessary to draw the entire layout on paper, but to design the basic cell common to all sections and determine how to program the cells to achieve the desired function.

(2) Once the layout has been completed on paper, programmability and parameterisation must be determined. That is:

(a) Is there a need to program the cells by conditional placement of transistors or contact cuts ?

(b) Should a variable number of inputs or outputs be provided to enable the same BELLE1 routine to be used for several different sections of the layout ?

(c) Should the width of power supply busses be variable

Strategy for implementing these parameters should be decided upon.

(3) At this stage, coding the circuit in BELLE1 can commence. Coordinates which are to be parameterised should be expressed in terms of variables during digitisation. By expressing coordinate positions relative to reference lines, (ie the lower left corner of the cell, a vertical line etc), stretching the cell is made easy. Coding the circuit in

Belle1 is effectively the same as writing a pascal procedure, with the added benefit that many of the required procedures and functions have been pre-written.

(4) The BELLE1 program together with the BELLE1 procedure describing your circuit, must be compiled and linked before it can be run. When the resulting program is run, a file is generated containing the CIF description of your layout. Error messages may occur either when the program is being compiled or as the program is run. The Pascal compilation errors pick up syntax and static semantic errors, while both Pascal or the BELLE1 program itself may detect errors during run-time. BELLE1 detects errors such as missing or duplicate symbol names, illegal geometries, etc.

(5) The generated CIF file should then be checked visually by plotting the layout on whatever facilities are available to the user. Errors can be corrected and modifications can be made by repeating the design and digitisation loop.

(6) Other factors:

(a) When working in multi-person design teams, it is necessary to take care in the specification of symbol names and symbol numbers to avoid clashes. When CIF files are combined or BELLE1 programs are merged, symbol number and name clashes cause BELLE1 to report errors. Designers should therefore assign themselves a range of CIF symbol numbers which no-one else in the design team may use. For example:

designer A	symbols 10000 - 10999
designer B	symbols 11000 - 11999
designer C	symbols 12000 - 12999

The starting CIF symbol number can be set using the SETSYMNO function in BELLE1.

(b) Symbols which are commonly used in many different sections of a design (eg a pullup transistor or a nor gate) should be placed into a separate CIF library, which can then be used in BELLE1 programs by several designers. Reference to library elements may be made in a BELLE1 program once an IMPORTSYMBOLS command has been issued to read the header file for those symbols.

(c) The IMPORTSYMBOLS command in BELLE1 does not read the CIF file, but only reads the header information to determine size and symbol numbers so that the library symbols can be called. It is necessary to include the CIF library at the beginning of the final BELLE1 generated CIF file for your circuit. When generating a library using BELLE1, the SETNOEND command should be used. This enables the CIF library to be inserted into other CIF files without modification.

(d) The symbol numbers 1 - 1000 are reserved for library cells such as pads, drivers etc. No two symbols can have the same symbol number nor the same symbol name within the overall design.

2.3.7 OTHER INSTRUCTIONS ON PROCEDURES

2.3.7.1 USAGE OF NODELABELS

Nodelabels in defined procedures must be drawn:

e.g.

```
  Procedure Nodes;
  begin {nodes}
  Define ('nodes');
     nodelabel('ai',60,60,metal);
     nodelabel('bi',120,60,metal);
     .
     . { must contain geometry (boxes etc. ) as
     . well as node labels }
     nodelabel('out',60,900,poly);
   enddef;
   end;
begin {main}
setsymno(10000);
nodes;
draw('nodes',0,0);
setnoend;
end. {main}
```

NOTE, however that if 'nodes' is defined as shown, it must contain some geometry. That is, it must consist of more than a series of "nodelabel" statements. This restriction is to allow the design to be processed by other design aids at CSIRO.

2.3.7.2 RUNNING BELLE1

To run BELLE1, enter the following command line:

```
$ bel1 <filename>
```

where <filename> is the name of your BELLE1 input file, which you will have created using an editor. The <filename> will, in general , be of the form:

```
FILENAME.EXT.
```

If you do not specify an extension to your filename (.ext), BELLE1 assumes that your input file has an extension of ".pas".

The BELLE1 output file (ie the CIF code) will be written into your directory with the name "filename.cif" where "filename" is the name of your input file.

An executable file "filename.exe" is also produced. This is the file that is actually run, and since it is large, BELLE1 asks whether you wish to delete it, after it has been run.

For debugging the Pascal program, it is often useful to have a listing file. This can be achieved by using the command:

```
$ bel1 <filename>/lis
```

2.3.7.3 EXTRACTION OF EXISTING DESIGNS, PADS ETC

Pads and other universal symbols are stored in the VMS Text library CIFLIB. This section will demonstrate how to extract a "pad" from the library and include it in a design.

The pad chosen is "padone" which for example may require one other symbol to be extracted from the library: padblank.

To extract the symbols required, type:

```
$ library/extract=(padblank,padone)/output=pads.cif   ciflib
```

This will generate a file "pads.cif", containing the CIF for "padone".

The BELLE1 program requires not the CIF file but instead a file containing the bounding boxes of the pad. This is done by typing:

```
getsymbol pads <CR>
```

which produces a file "pads.slb". It is now possible to write a short BELLE1 program to use this pad, making use of the BELLE1 importsymbols procedure.This program follows:

```
[inherit('belle1__dir:belle1.pen'),check(all)]
program testpad(input,output);
   procedure designproc;
   begin {designproc}
      importsymbols('pads.slb');  {inform BELLE1 of the
      existence of extra definitions in file "pads.slb"}
      draw('padone',0,0);
             { draw "padone" from "pads.slb" }
   end; {designproc}
begin {testpad}
   open__ciffile('CIFFILE');
   settech(nmos);
   designproc;
   close__ciffile;
end {testpad}
```

This produces a file containing a call to padone (testpad.cif) but the actual CIF for the symbol is not contained in the file. The file containing both the call (from testpad.cif) and the pad (from pads.cif) can be created by typing:

```
       $ copy pads.cif,testpad.cif  temp.cif
```

which will append testpad.cif to pads.cif in the file temp.cif.

by typing:

```
          $ rename temp.cif testpad.cif
```

the complete file will be in testpad.cif.

2.3.8 GENERAL STRUCTURE OF BELLE1 DESIGN PROGRAMS

The general structure of design programs in BELLE1 differs slightly from the approach used in previous versions of the procedural layout language BELLE. The difference lies in the fact that BELLE1 users write a full Pascal program, whereas BELLE users were writing only procedures. The Structure of a BELLE1 design program may be seen by considering the file represented below. This file, TEMPLATE_.PAS, represents the basis for any design undertaken using BELLE1.

```
[inherit('belle1__dir:belle1.pen'),check(all)]
program template(input,output);


     procedure designproc;
     begin { designproc }

        { INSERT DESIGN DEFINITION HERE }

     end; { designproc }
begin { template }
     open__ciffile('CIFFILE');
     settech(nMOS);   { or "settech(cMOS);" }
     designproc;
     close__ciffile;
end. { template }
```

NOTES:

(1) The first line must be included, but the user need not understand what it is doing. (in fact, it allows the general BELLE1 environment required by the BELLE1 programs to be inherited and used by the user.

(2) The name of the file and the program name should be the same.

(3) The design may be placed where indicated, using correct Pascal syntax. Using this approach, any design that has been developed using BELLE may also be used with BELLE1 , by inserting the design at this point.

(4) The technology type must be selected as nMOS or cMOS.

(5) The file type is .PAS (not .BEL).

3.0 BELLE1 SYSTEM DEFINITIONS

To configure BELLE1 for normal operation, two global symbols need to be defined for all BELLE1 users.

(1) Set up a subdirectory (preferably in a group manager directory) and place all the BELLE1 files therein. A global symbol "belle1_dir" will need to be defined to reference that particular directory.
e.g. $ assign <directory_spec> belle1_dir

(2) Define the symbol BEL1 to invoke the BELLE1 command procedure

```
$ bel1 :== @belle_dir:belle1.com
```

in a suitable group LOGIN.COM file would be suitable.

(3) Set up a library file CIFLIB containing all current cells (CIF code) as a VAX/VMS Text Library, for the users to extract as they wish. A list of available cells for the

users would be desirable.

APPENDIX C

A SAMPLE ROWAN TECHNOLOGY FILE

```
{ These rules are taken from "Portable Design Rules for Bulk
  CMOS", VLSI Design, pp. 62-67, Sep/Oct 1982, by Thomas W.
  Griswold. The rules are listed under headings from figure 2
  of that paper. }

lambda(250); { lambda = 2.5 microns }

layers;

 layer('pwell','CW');
 layer('thinox','CD');
 layer('poly','CP');
 layer('p+','CS');
 layer('contact','CC');
 layer('metal','CM');
 layer('glass','CG');

rules;

  { THIN OXIDE AND P-WELL }

  separation('thinox.not  pwell','pwell', 5,'External thinox to pwell spacing');
  separation('thinox.pwell','not  pwell',3,'Internal thinox to pwell spacing');
```

```
spacing('pwell',2,'P-well  spacing');

width('pwell',4,'P-well  width');


{ THIN OXIDE AND P+ }


width('p+',2,'P+  width');

spacing('p+',2,'P+  spacing');

extension('p+','thinox',1.5,'Internal  p+ to thinox spacing');

extension('not  p+','thinox',1.5,'External  p+ to thinox spacing');

width('thinox',2,'Thinox  width');

spacing('thinox',3,'Thinox  spacing');


{ METAL-POLY CONTACT }


width('contact',2,'Contact  width');

separation('contact.poly','not  poly',1,'Poly width around contact');

separation('contact.metal','not  metal',1,'Metal  width  around  contact');


{ CONTACTS AND GATES }


separation('contact','poly.thinox',2,'Contact  to gate spacing');


{ THIN OXIDE AND POLY }


width('poly',2,'Poly  width');

spacing('poly',2,'Poly  spacing');

extension('not  thinox','poly',1,'Poly  to thinox spacing');

extension('thinox','poly',2,'Thinox  overhang  on  gate');

extension('poly','thinox',2,'Poly  overhang  on  gate');
```

```
define('Malformed transistor');
 context('   not poly','      X',
        'poly.thinox','  not thinox');
 constraint(-1,2,'thinox');
 constraint(1,1,'not poly.not thinox');
 constraint(2,-1,'poly');
endrule;

{ METAL }

width('metal',3,'Metal width');
spacing('metal',3,'Metal spacing');

{ P+ AND GATE EDGES }

separation('poly.thinox.p+','not p+',1.5,'P+ width around gate');
separation('poly.thinox.not p+','p+',1.5,'P+ to gate spacing');

{ METAL-THIN OXIDE CONTACTS }

spacing('contact',2,'Contact spacing');
separation('contact.thinox','not thinox',1,'Thinox width around contact');

{ SPLIT CONTACTS }

define('Malformed split contact');
 context('     not contact       ','   X',
        'contact.metal.thinox.p+',' not p+');
 constraint(3,-1,'contact');
```

```
endrule;

define('Malformed split contact');

 context('     not contact         ','   X',
        'contact.metal.thinox.not  p+',' p+');

 constraint(3,-1,'contact');

endrule;


{ some miscellaneous contact rules }


define('Contact  without metal');

 context('       X','           X',
        'contact.not metal',' X');

 constraint(-1,-1,'not  contact');   { fails automatically }

endrule;


define('Contact  without poly or thinox');

 context('       X','           X',
        'contact.not poly.not thinox',' X');

 constraint(-1,-1,'not  contact');   { fails automatically }

endrule;
```

APPENDIX D

THE BELLE AND CIF FILES FOR THE SIGNATURE ANALYSER

The listing that appears first in this appendix is the BELLE description of the complete Signature Analyser.

```
PROCEDURE dynshift;


CONST
      Diff_width       =  2;
      Diff_spacing     =  3;
      Poly_width       =  2;
      Metal_width      =  3;
      Pol_Dif_spacing  =  1;
      Pol_Dif_overlap  =  2;
      Impl_over_channel=  2;
      Impl_to_channel  =  2;
      Cut_size         =  2;
      Cut_spacing      =  2;
      Cut_to_edge      =  1;
      Pwr_width        =  4;

VAR
      cell_start: coordtype;
      cell_width,cell_length: integer;
```

```
begin
   setsymno(9040);
   define('dyn_shift');
   cell_width:=33;
   cell_length:=48;
      Layer(Metal);
         box(0,0,cell_width,pwr_width);
         box(0,22,cell_width,22+pwr_width);
         box(0,44,cell_width,44+pwr_width);
     Layer(diffusion);
         box(14,14,20,20);
         box(14,6,17,15);
         box(16,6,30,8);
         box(14,28,20,34);
         box(14,33,17,42);
         box(16,40,30,42);
         wire(2,19,0); y(6);
         wire(2,0,35); x(6); y(10); x(14);
         box(15,22,19,26);

     Layer(poly);
         box(22,4,28,11);
         box(17,9,20,13);
         box(19,9,23,11);
         box(17,35,20,39);
         box(22,37,28,44);
         box(19,37,23,39);
         wire(2,13,17); x(26); y(37);
         wire(2,12,12); y(-11);
      wire(2,3,33); y(59);
      wire(2,11,31); x(21);

   Layer(Implant);
      box(20,4,30,10);
      box(20,38,30,44);

   md(-1,34);
   mp(28,34);
   md(31,42);
   md(17,27);
   md(17,21);
   md(31,6);
   dp(17,11,0);
   dp(8,32,0);
   dp(17,37,0);

enddef;
end;
begin
   dynshift;
   setnoend;
end;


Procedure Slice;

begin
   importsymbols('plafile.slb');
importsymbols('dynshift.slb');

setsymno(9050);

define('contact_wire');

   layer(metal);

   box(0,0,118,4);

enddef;

define('slice');

layer(green);

      box(71,125,75,129);

      box(39,161,43,165);

      box(55,285,59,289);

layer(red);

      box(71,128,75,131);

      box(39,164,43,167);

      box(55,283,59,286);

layer(cuts);

      box(72,126,74,130);

      box(40,162,42,166);

      box(56,284,58,288);

layer(metal);

      box(71,125,75,131);

      box(39,161,43,167);

      box(55,283,59,289);

draw('dyn_shift',0,65);      my;

draw('dyn_shift',54,65);     my;

draw('dyn_shift',22,144);    my;

draw('dyn_shift',83,144);    my;

draw('dyn_shift',38,321);

draw('dyn_shift',83,321);
```

```
  draw('pla9000',0,195);


{ Place the contacts lines to the leaf cells}


  draw('contact_wire',0,3);

  draw('contact_wire',0,10);

  draw('contact_wire',0,68);

  draw('contact_wire',0,75);

  draw('contact_wire',0,82);

  draw('contact_wire',0,89);

  draw('contact_wire',0,147);

  draw('contact_wire',0,154);

  draw('contact_wire',0,307);

  draw('contact_wire',0,314);


{ Conect the power to the leaf cells}


  draw('contact_wire',0,372);

  draw('contact_wire',0,379);

  draw('contact_wire',0,17);

  draw('contact_wire',0,39);

  draw('contact_wire',0,61);

  draw('contact_wire',0,96);

  draw('contact_wire',0,118);

  draw('contact_wire',0,140);

  draw('contact_wire',0,321);

  draw('contact_wire',0,343);

  draw('contact_wire',0,365);
```

```
{ Connect the leaf cells to the connect lines}


   mp(12,77);

   mp(66,70);

   mp(3,12);

   mp(57,5);

   mp(25,91);

   mp(86,84);

   mp(34,156);

   mp(95,149);

   mp(50,316);

   mp(95,309);

   mp(41,381);

   mp(86,374);


{ Add connections for power mods to the PLA}

{ Also make connections between SR's}

   layer(metal);


   wire(3,84,31); x(115);

   wire(3,0,110); x(19);

   wire(3,113,110); x(118);

   wire(3,0,355); x(35);

   wire(3,112,355); x(118);


   wire(3,30,31); x(51);

   wire(3,51,110); x(80);

   wire(3,67,355); x(80);
```

```
  layer(diffusion);


  wire(2,73,65);  y(126);

  wire(2,41,144);  y(162);

  wire(2,57,321);  y(286);


  layer(poly);


  wire(2,0,301);  x(25);  y(285);

  wire(2,95,295);  y(301);  x(118);


  wire(2,73,130);  y(165);

  enddef;


  end;
begin

  slice;

  draw('slice',185,83);

  draw('slice',303,83);

  draw('slice',421,83);

  draw('slice',539,83);

  setnoend;

end;



procedure layout;


   begin

       importsymbols('pads.slb');
```

```
      setsymno(9070);
      define('layout');
      draw('newpadin',-13,107);  rot(-90);
      draw('newpadin',-13,213);  rot(-90);
      draw('newpadin',-13,319);  rot(-90);
      draw('newpadin',-13,425);  rot(-90);
      draw('newpadin',-13,531);  rot(-90);
      draw('newpadin',-13,637);  rot(-90);
      draw('newpadin',222,649);  rot(180);
      draw('padvdd',328,649);  rot(180);
      draw('newpadin',434,649);  rot(180);
      draw('newpadin',540,649);  rot(180);
      draw('newpadin',646,649);  rot(180);
      draw('newpadin',752,649);  rot(180);
      draw('padground',848,424);  rot(90);
      draw('padout',848,318);  rot(90);
      draw('padout',848,212);  rot(90);
      draw('padout',848,106);  rot(90);
      draw('padout',848,0);  rot(90);
      enddef;
   end;
procedure connect;
     begin
       define ('connect');
         layer(blue);
              comment('this is the interconnect for gnd between pads');
              wire(8,122,547);  x(111);  y(530);  x(89);  y(4);  x(746);  y(547);  x(400);
              comment('this is the interconnect for vdd between pads');
              wire(8,-9,106);  y(645);  x(844);  y(106);
     comment('this is pad in no. 6 interconnect');
layer(red);
     mp(111,594);
     wire(2,111,594); y(515); x(130);
     mp(130,515);
layer(blue);
     wire(3,105,594); x(111);
     wire(3,130,515); y(193);
     mp(130,193);
layer(red);
     wire(2,130,193); x(185);
     mp(185,193);
     comment('this is pad in no. 7 interconnect');
layer(blue);
     wire(3,179,531); y(524); x(138); y(438);
     mp(138,438);
layer(red);
     wire(2,138,438); x(185);
     mp(185,438);
     comment('this is pad in no. 5 interconnect');
layer(blue);
     wire(3,105,488); x(122); y(135); x(138); y(113);
     md(138,113);
layer(green);
     wire(2,138,113); x(185);
     comment('this is pad in no. 4 interconnect');
layer(red);
     mp(115,382);
     wire(2,115,382); x(138);
     mp(138,382);
layer(blue);
     wire(3,105,382); x(115);
     wire(3,138,382); y(160);
     mp(138,239);
     mp(138,160);
layer(red);
     wire(2,138,160); x(185);
     wire(2,138,239); x(185);
     mp(185,239);
     mp(185,160);
     comment('this is pad in no. 3 interconnect');
layer(red);
     mp(115,276);
     wire(2,115,276); x(146);
     mp(146,276);
layer(blue);
     wire(3,105,276); x(115);
     wire(3,146,276); y(153);
     mp(146,232);
     mp(146,153);
layer(red);
     wire(2,146,232); x(185);
     wire(2,146,153); x(185);
     mp(185,153);
     mp(185,232);
     comment('this is pad in no. 2 interconnect');
layer(blue);
     wire(3,105,170); x(115);
     wire(3,154,174); y(95); x(185);
     mp(115,170);
     mp(154,174);
layer(red);
     wire(2,115,170); y(174); x(185);
     mp(185,174);
     comment('this is pad in no. 1 interconnect');
layer(blue);
     wire(3,105,64); x(130); y(88); x(185);
     wire(3,130,88); y(108);
     mp(130,108);
layer(red);
     wire(2,130,108); y(143);
     mp(130,143);
layer(blue);
     wire(3,130,143); y(167);
     mp(130,167);
layer(red);
     wire(2,130,167); x(185);
     mp(185,167);
     comment('this is pad in no. 8 interconnect');
layer(blue);
     wire(3,220,524); x(212); y(516);  x(146); y(392);
     wire(3,391,531); y(524); x(240);
     mp(240,524);
     mp(220,524);
     mp(146,392);
layer(red);
     wire(2,220,524); x(240);
```

```
          wire(2,146,392); x(185);
          mp(185,392);
          comment('this is pad in no. 9 interconnect');
     layer(blue);
          wire(3,220,516); y(508); x(154); y(399);
          wire(3,497,531); y(516); x(240);
          mp(240,516);
          mp(220,516);
          mp(154,399);
     layer(red);
          wire(2,220,516); x(240);
          wire(2,154,399); x(185);
          mp(185,399);
          comment('this is pad in no. 10 interconnect');
     layer(blue);
          wire(3,603,531); y(464);
          comment('this is pad in no. 11 interconnect');
     layer(blue);
          wire(3,709,531); y(524); x(667); y(457); x(657);
          comment('this is pad out no. 12 interconnect');
     layer(red);
          wire(2,704,371); x(697);
          mp(697,371);
     layer(blue);
          wire(3,697,371); y(438);
          mp(697,438);
     layer(red);
          wire(2,697,438); x(657);
          mp(657,438);
     comment('this is pad out no. 13 interconnect');
     wire(2,704,265); x(689);
     mp(689,265);
layer(blue);
     wire(3,689,265); y(384);
     mp(689,384);
layer(red);
     wire(2,689,384); x(657);
     comment('this is pad out no. 14 interconnect');
layer(red);
     wire(2,704,159); x(697);
     mp(697,159);
layer(blue);
     wire(3,697,159); y(193);
     mp(697,193);
layer(red);
     wire(2,697,193); x(657);
     mp(657,193);
     comment('this is pad out no. 15 interconnect');
layer(red);
     wire(2,704,53); x(697);
     mp(697,53);
layer(blue);
     wire(3,697,53); y(114); x(680);
     mp(680,114);
layer(red);
     wire(2,680,114); x(655);
     mp(655,114);
     comment('this is pad gnd interconnect');
```

```
layer(blue);
     wire(8,805,477); x(684); y(428); x(672); y(128);
     wire(4,672,128); y(124); x(657);
     wire(4,672,428); x(657);
     wire(4,672,203); x(657);
     wire(4,672,286); x(657);
     comment('this is pad vdd interconnect');
layer(blue);
     wire(8,275,596); x(230); y(490); x(165); y(105);
     wire(4,165,105); y(102); x(185);
     wire(4,165,225); x(185);
     wire(4,165,450); x(185);
     wire(4,165,269); x(185);
     wire(4,165,336); x(185);
     wire(4,165,146); x(185);
     wire(4,165,181); x(185);
     wire(4,165,406); x(185);
     comment('these are the node labels for all the pads');
     nodelabel('vdd',275,596,metal);
     nodelabel('gnd',795,477,metal);
     nodelabel('PHI2SL',40,53,metal);
     nodelabel('PHI1SL',40,159,metal);
     nodelabel('PHI2NSL',40,265,metal);
     nodelabel('PHI1NSL',40,371,metal);
     nodelabel('MSKSRIN',40,477,metal);
     nodelabel('SIGSRIN',40,583,metal);
     nodelabel('DATSRIN',169,596,metal);
     nodelabel('PHI2NDL',381,596,metal);
     nodelabel('PHI1NDL',487,596,metal);
```

```
                nodelabel('PHI1DL',593,596,metal);

                nodelabel('PHI2DL',699,596,metal);

                nodelabel('DATOUT',795,371,metal);

                nodelabel('COMPOUT',795,265,metal);

                nodelabel('SIGOUT',795,159,metal);

                nodelabel('MSKOUT',795,53,metal);

                mp(165,384);

                layer(red);

                wire(2,165,384);  x(185);

                enddef;

     end;

procedure logo(height,width:real);(*height  and width of the logo*)

label 33;

var here,line,w,h:integer;




procedure posn(space:integer);  (*calculates position of next letter*)

begin

    here:=here+space+w+w div 2;

end;




 begin
```

```
    if (height<60) or (width<255) then

    begin writeln('Invalid size, try again');goto 33;end;

    height:=height/4.56666;

    h:=trunc(height);

    width:=width/18.2;

    w:=trunc(width);



(*start definitions of characters*);

    define('U_logo');

    layer(metal);

    wire(4,0,h);y(0);x(w);y(h);

    enddef;


    define('O_logo');

    layer(metal);

    draw('U_logo',0,0);

    wire(4,0,h);x(w);

    enddef;


    define('F_logo');

    layer(metal);

    wire(4,0,0);y(h);x(w);

    wire(4,0,h div 2);x(w div 2);

    enddef;


    define('A_logo');

    layer(metal);

    wire(4,0,0);y(h);x(w);y(0);x(w);
```

```
 wire(4,0,h div 2);x(w);

 enddef;


define('d_logo');

 layer(metal);

 wire(4,w,h);y(0);x(0);y(h  div 2);x(w);

 enddef;


 define('E_logo');

 layer(metal);

 draw('F_logo',0,0);

 wire(4,0,0);x(w);

 enddef;


 define('L_logo');

 layer(metal);

 wire(4,0,h);y(0);x(w);

 enddef;


 define('p_logo');

 layer(metal);

 wire(4,0,0);y(h);x(w);y(h  div 2);x(0);

 enddef;


 define('b_logo');

 layer(metal);

 wire(4,0,h);y(0);x(w);y(h  div 2);x(0);

 enddef;
```

```
define('1_logo');

layer(metal);

wire(4,w div 2,h);y(0);

enddef;


define('i_logo');

layer(metal);

wire(4,w div 2,h);y(0);

wire(4,0,h);x(w);

wire(4,0,0);x(w);

enddef;


define('C_logo');

layer(metal);

wire(4,w,h);x(0);y(0);x(w);

enddef;


define('g_logo');

layer(metal);

draw('c_logo',0,0);

wire(4,w,0);y(h div 2);

enddef;


define('eight_logo');

layer(metal);

draw('O_logo',0,0);

wire(4,0,h div 2);x(w);

enddef;
```

```
define('two_logo');

layer(metal);

wire(4,0,h);x(w);y(h  div 2);x(0);y(0);x(w);

enddef;


define('dash_logo');

layer(metal);

wire(4,w div 3,h div 2);x(2*w div 3);

enddef;


define('J_logo');

layer(metal);

wire(4,0,h); x(w);

wire(4,w div 2,h); y(0); x(0); y(h div 2);

enddef;


define('n_logo');

layer(metal);

wire(4,0,0); y(h); x(w); y(0);

enddef;


define('W_logo');

layer(metal);

wire(4,0,h); y(0); x(w); y(h);

wire(4,w div 2,0); y(h);

enddef;


define('k_logo');

layer(metal);
```

```
   wire(4,0,0); y(h);

   wire(4,0,h div 2); x(w); y(h);

   wire(4,w*3 div 4,h div 2); y(0);

   enddef;


   define('four_logo');

   layer(metal);

   wire(4,0,h); y(h div 2); x(w);

   wire(4,w,h); y(0);

   enddef;


(*start logo here, only need to alter draw statements *)

   define('logo');

   here:=2  ;

   line:=14*h div 4;

   draw('U_logo',here,line);

   posn(w);

   draw('O_logo',here,line);

   posn(0);

   draw('F_logo',here,line);

   posn(w);

   draw('A_logo',here,line);

   posn(0);

   draw('d_logo',here,line);

   posn(0);

   draw('E_logo',here,line);

   posn(0);

   draw('L_logo',here,line);

   posn(0);
```

```
  draw('A_logo',here,line);
  posn(0);
  draw('I_logo',here,line);
  posn(0);
  draw('D_logo',here,line);
  posn(0);
  draw('E_logo',here,line);
  line:=7*h div 4;
  here:=3*w div 2;
  draw('E_logo',here,line);
  posn(0);
  draw('E_logo',here,line);
  posn(0);
  draw('eight_logo',here,line);
  posn(0);
  draw('two_logo',here,line);
  posn(0);
  draw('dash_logo',here,line);
  posn(0);

(*put project number here*)
    draw('O_logo',here,line);
    posn(0);
    draw('four_logo',here,line);
    posn(0);


    draw('dash_logo',here,line);
    posn(0);
```

```
(*demonstrators initials*)

    draw('k_logo',here,line);

    posn(0);

    draw('E_logo',here,line);

    line:=2;

    here:=7*w div 2;


(*designers initials*)

    draw('A_logo',here,line);

    posn(0);

    draw('W_logo',here,line);

    posn(0);


    draw('dash_logo',here,line);

    posn(0);


    draw('J_logo',here,line);

    posn(0);

    draw('n_logo',here,line);

    posn(0);


    enddef;

 33:;

 end;


 begin

     connect;

     draw('connect',0,0);

     logo(66,255);
    draw('logo',290,12);

    layout;

    draw('layout',0,0);

end;
```

The second listing is the CIF description of the complete Signature Analyser.

```
25 Lambda = 250;

DS 9041;

9 diffcut;

42 -500,-500 500,500;

L ND;

B 1000 1000 0,0;

L NC;

B 500 500 0,0;

L NM;

B 1000 1000 0,0;

DF;


DS 9042;

9 polycut;

42 -500,-500 500,500;

L NP;

B 1000 1000 0,0;

L NC;

B 500 500 0,0;
```

```
L NM;
B 1000 1000 0,0;
DF;

DS 9043;
9 buttcont;
42 -750,-500 750,500;
L ND;
B 1000 1000 -250,0;
L NP;
B 750 1000 375,0;
L NC;
B 1000 500 0,0;
L NM;
B 1500 1000 0,0;
DF;

DS 9040;
9 dyn_shift;
42 -750,-3000 8250,15000;
L NM;
B 8250 1000 4125,500;
B 8250 1000 4125,6000;
B 8250 1000 4125,11500;
L ND;
B 1500 1500 4250,4250;
B 750 2250 3875,2625;
B 3500 500 5750,1750;
B 1500 1500 4250,7750;
```

```
B 750 2250 3875,9375;

B 3500 500 5750,10250;

W 500  4750,0 4750,1500;

W 500  0,8750 1500,8750 1500,2500 3500,2500;

B 1000 1000 4250,6000;

L NP;

B 1500 1750 6250,1875;

B 750 1000 4625,2750;

B 1000 500 5250,2500;

B 750 1000 4625,9250;

B 1500 1750 6250,10125;

B 1000 500 5250,9500;

W 500  3250,4250 6500,4250 6500,9250;

W 500  3000,3000 3000,-2750;

W 500  750,8250 750,14750;

W 500  2750,7750 5250,7750;

L NI;

B 2500 1500 6250,1750;

B 2500 1500 6250,10250;

C 9041 T -250,8500;

C 9042 T 7000,8500;

C 9041 T 7750,10500;

C 9041 T 4250,6750;

C 9041 T 4250,5250;

C 9041 T 7750,1500;

C 9043 T 4250,2750;

C 9043 T 2000,8000;

C 9043 T 4250,9250;

DF;
```

```
25 Lambda = 250;

DS 9001;      9 FsmGround  ;
( 4 Items. );
L ND;
B 1000 1500 1500,1000;
L NP;
B 500 1500 500,1000;
B 500 1500 2500,1000;
L NC;
B 500 500 1500,750;
DF;

DS 9002;    9 FsmPullps  ;
( 16 Items. );
L ND;
B 1000 1000 500,750;
B 1000 1000 500,2750;
B 2250 500 1875,750;
B 2250 500 1875,2750;
B 1000 1000 3250,750;
B 1000 1000 3250,2750;
L NI;
B 2250 1500 2375,750;
B 2250 1500 2375,2750;
L NP;
B 1250 1500 2375,750;
B 1250 1500 2375,2750;
```

```
L NC;

B 500 500 500,750;

B 500 500 500,2750;

B 1000 500 3000,750;

B 1000 500 3000,2750;

L NM;

B 1500 1000 3000,750;

B 1500 1000 3000,2750;

DF;


DS 9003;    9 FsmConnct  ;

( 10 Items. );

L ND;

B 1250 1000 2625,1500;

L NP;

B 1000 1000 750,750;

B 1000 1000 750,2750;

B 2250 500 2125,500;

B 2250 500 2125,2500;

L NC;

B 500 500 750,750;

B 500 500 750,2750;

B 500 500 2500,1500;

L NM;

B 1250 1000 625,750;

B 1250 1000 625,2750;

B 1000 4000 2500,2000;

DF;
```

```
DS 9004;     9 FsmConnSp ;

( 5 Items. );

L NP;

B 1000 1000 2500,750;

B 500 500 3000,500;

L NC;

B 500 500 2500,750;

L NM;

B 2250 1000 1125,750;

B 1000 2000 2500,1000;

DF;


DS 9006;     9 FsmGndSp ;

( 2 Items. );

L NP;

B 1000 1000 1250,500;

L NC;

B 500 500 1250,500;

DF;


DS 9007;     9 FsmIn ;

( 32 Items. );

L ND;

B 1000 1000 1000,5500;

B 1000 1500 1250,7750;

B 1000 1500 1250,3250;

B 500 1500 1000,6500;

B 500 250 1000,7125;

B 500 1500 1000,4750;
```

```
B 500 250 1000,3875;

B 1000 1250 1500,625;

B 1500 2000 2500,3000;

B 1500 2000 2500,8750;

B 1000 1000 3500,750;

B 750 7750 3375,4875;

L NI;

B 1500 1500 1000,6500;

B 1500 1500 1000,4500;

L NP;

B 1000 1000 500,8000;

B 1000 1000 500,3000;

B 500 1500 250,7000;

B 500 1500 250,4000;

B 2250 500 1375,6500;

B 1500 500 1000,4500;

B 500 2750 500,1375;

B 500 6750 2500,3375;

B 500 3750 2500,9125;

L NC;

B 1000 500 1000,8000;

B 1000 500 1000,3000;

B 500 500 1000,5500;

B 500 500 1500,750;

B 500 500 3500,750;

L NM;

B 4000 1000 2000,750;

B 4000 1000 2000,5500;

B 1500 1000 1000,8000;
```

```
B 1500 1000 1000,3000;
DF;

DS 9008;    9 FsmCkdIn ;
( 9 Items. );
C       9007;
L ND;
B 1500 2000 2500,10750;
B 1000 1000 2500,13000;
B 500 1500 2500,14000;
L NP;
B 4000 500 2000,14000;
B 1000 750 2500,12375;
B 500 2500 2500,11000;
L NC;
B 500 1000 2500,12750;
L NM;
B 1000 1500 2500,12750;
DF;

DS 9021;    9 FsmOut ;
( 41 Items. );
L ND;
B 500 3250 750,6875;
B 1000 1000 1000,8500;
B 1500 2000 1500,4750;
B 1500 500 1500,8250;
B 1000 1000 1250,10750;
B 750 500 1625,10250;
```

```
B 2250 500 2375,11500;
B 500 2250 2000,9375;
B 1000 1000 2500,3750;
B 1500 2000 2750,6250;
B 1000 1000 3250,12500;
B 500 5500 3500,9500;
L NI;
B 1500 1500 2000,9000;
B 1500 1500 2750,11500;
L NP;
B 500 3000 250,10500;
B 1000 750 1000,9125;
B 500 2000 750,12500;
B 1000 1000 1250,2000;
B 500 4000 1500,4250;
B 1500 500 2000,9000;
B 750 1000 2625,12500;
B 500 3250 2750,6125;
B 500 750 2750,13125;
B 500 1500 2750,11500;
B 750 500 3125,4750;
B 1000 1000 3250,2000;
B 500 2750 3500,3625;
L NC;
B 500 1000 1000,8750;
B 500 500 1250,2000;
B 500 500 1250,10750;
B 500 500 2500,3750;
B 1000 500 3000,12500;
```

```
B 500 500 3250,2000;

L NM;

B 4000 1000 2000,3750;

B 4000 1000 2000,10750;

B 1000 1500 1000,8750;

B 1000 3500 1250,750;

B 1500 1000 3000,12500;

B 1000 3500 3250,750;

DF;


DS 9022;    9 FsmCkdOut ;

( 41 Items. );

L ND;

B 1000 1000 1250,-500;

B 1000 1000 3250,-500;

B 500 1500 1250,500;

B 500 1500 3250,500;

B 1000 1000 1250,1500;

B 1000 1000 3250,1500;

B 500 3250 750,6875;

B 1000 1000 1000,8500;

B 1500 2000 1500,4750;

B 1500 500 1500,8250;

B 1000 1000 1250,10750;

B 750 500 1625,10250;

B 2250 500 2375,11500;

B 500 2250 2000,9375;

B 1000 1000 2500,3750;

B 1500 2000 2750,6250;
```

```
B 1000 1000 3250,12500;

B 500 5500 3500,9500;

L NI;

B 1500 2000 2000,9250;

B 2000 1500 2500,11500;

L NP;

B 4000 500 2000,500;

B 500 3000 250,10500;

B 1000 750 1000,9125;

B 500 2000 750,12500;

B 1000 750 1250,2125;

B 500 4000 1500,4250;

B 1500 1000 2000,9250;

B 750 1000 2625,12500;

B 500 3250 2750,6125;

B 500 750 2750,13125;

B 1000 1500 2500,11500;

B 750 500 3125,4750;

B 1000 750 3250,2125;

B 500 2750 3500,3625;

L NC;

B 500 500 1250,-500;

B 500 500 3250,-500;

B 500 1000 1000,8750;

B 500 1000 1250,1750;

B 500 500 1250,10750;

B 500 500 2500,3750;

B 1000 500 3000,12500;

B 500 1000 3250,1750;
```

```
L NM;

B 4000 1000 2000,3750;

B 4000 1000 2000,10750;

B 1000 1500 1000,8750;

B 1000 1500 1250,1750;

B 1500 1000 3000,12500;

B 1000 1000 1250,-500;

B 1000 1000 3250,-500;

B 1000 1500 3250,1750;

DF;


DS 9019;    9 FsmProgrm ;

( 2 Items. );

L ND;

B 2500 1000 1250,0;

L NC;

B 500 500 2000,0;

DF;


DS 9000 250 1;

9 PLA9000;

C 9002 T 0,15;

L NM;

B 66 4 47,18;

B 66 4 47,26;

C 9003 T 79,15;

L NP;

B 20 2 101,17;

B 20 2 101,25;
```

```
L ND;

B 20 4 101,21;

C 9001 R 0,1 T 116,15;

C 9002 T 0,31;

L NM;

B 66 4 47,34;

B 66 4 47,42;

C 9003 T 79,31;

L NP;

B 20 2 101,33;

B 20 2 101,41;

L ND;

B 20 4 101,37;

C 9001 R 0,1 T 116,31;

C 9001 T 15,7;

L NP;

B 2 36 17,30;

B 2 36 25,30;

L ND;

B 4 36 21,30;

C 9007 T 15,47;

C 9001 M Y T 31,53;

L NP;

B 2 36 33,30;

B 2 36 41,30;

L ND;

B 4 36 37,30;

C 9007 M Y T 31,13;

C 9001 T 47,7;
```

```
L NP;
B 2 36 49,30;
B 2 36 57,30;
L ND;
B 4 36 53,30;
C 9007 T 47,47;
C 9001 M Y T 63,53;
L NP;
B 2 36 65,30;
B 2 36 73,30;
L ND;
B 4 36 69,30;
C 9007 M Y T 63,13;
C 9002 R 0,1 T 108,0;
L NM;
B 4 34 97,31;
B 4 34 105,31;
C 9021 T 92,47;
L NM;
94 vdd 2,10 NM;
B 4 58 2,42;
B 84 4 42,69;
B 84 4 42,-9;
B 4 15 82,-3;
B 27 4 93,2;
B 4 25 82,79;
B 42 4 101,90;
B 4 15 105,-3;
B 19 4 112,-9;
```

```
B 4 25 120,79;

B 78 4 52,10;

B 78 4 52,50;

B 4 56 89,36;

B 28 4 101,62;

B 4 57 113,35;

B 19 4 9,8;

B 11 4 116,8;

94 gnd 113,47 NM;

C 9019 T 21,18;

C 9019 T 69,18;

C 9019 T 21,26;

C 9019 T 37,26;

C 9019 T 53,26;

C 9019 R -1,0 T 69,26;

C 9019 T 21,34;

C 9019 R -1,0 T 37,34;

C 9019 R -1,0 T 53,34;

C 9019 R -1,0 T 69,34;

C 9019 R 0,-1 T 97,21;

C 9019 R 0,1 T 97,21;

C 9019 R 0,-1 T 97,37;

DF;


25 Lambda = 250;

DS 9050;

9 contact_wire;

42 0,0 29500,1000;

L NM;
```

```
B 29500 1000 14750,500;

DF;


DS 9052;

9 polycut;

42 -500,-500 500,500;

L NP;

B 1000 1000 0,0;

L NC;

B 500 500 0,0;

L NM;

B 1000 1000 0,0;

DF;


DS 9051;

9 slice;

42 -750,750 29875,95750;

L ND;

B 1000 1000 18250,31750;

B 1000 1000 10250,40750;

B 1000 1000 14250,71750;

L NP;

B 1000 750 18250,32375;

B 1000 750 10250,41375;

B 1000 750 14250,71125;

L NC;

B 500 1000 18250,32000;

B 500 1000 10250,41000;

B 500 1000 14250,71500;
```

```
L NM;
B 1000 1500 18250,32000;
B 1000 1500 10250,41000;
B 1000 1500 14250,71500;
C 9040 MY T 0,16250;
C 9040 MY T 13500,16250;
C 9040 MY T 5500,36000;
C 9040 MY T 20750,36000;
C 9040 T 9500,80250;
C 9040 T 20750,80250;
C 9000 T 0,48750;
C 9050 T 0,750;
C 9050 T 0,2500;
C 9050 T 0,17000;
C 9050 T 0,18750;
C 9050 T 0,20500;
C 9050 T 0,22250;
C 9050 T 0,36750;
C 9050 T 0,38500;
C 9050 T 0,76750;
C 9050 T 0,78500;
C 9050 T 0,93000;
C 9050 T 0,94750;
C 9050 T 0,4250;
C 9050 T 0,9750;
C 9050 T 0,15250;
C 9050 T 0,24000;
C 9050 T 0,29500;
C 9050 T 0,35000;
```

```
C 9050 T 0,80250;

C 9050 T 0,85750;

C 9050 T 0,91250;

C 9052 T 3000,19250;

C 9052 T 16500,17500;

C 9052 T 750,3000;

C 9052 T 14250,1250;

C 9052 T 6250,22750;

C 9052 T 21500,21000;

C 9052 T 8500,39000;

C 9052 T 23750,37250;

C 9052 T 12500,79000;

C 9052 T 23750,77250;

C 9052 T 10250,95250;

C 9052 T 21500,93500;

L NM;

W 750  21000,7750  28750,7750;

W 750  0,27500  4750,27500;

W 750  28250,27500  29500,27500;

W 750  0,88750  8750,88750;

W 750  28000,88750  29500,88750;

W 750  7500,7750  12750,7750;

W 750  12750,27500  20000,27500;

W 750  16750,88750  20000,88750;

L ND;

W 500  18250,16250  18250,31500;

W 500  10250,36000  10250,40500;

W 500  14250,80250  14250,71500;

L NP;
```

```
W 500  0,75250 6250,75250 6250,71250;

W 500  23750,73750 23750,75250 29500,75250;

W 500  18250,32500 18250,41250;

DF;


C 9051 T 46250,20750;

C 9051 T 75750,20750;

C 9051 T 105250,20750;

C 9051 T 134750,20750;

DS 2;

9 PADBLANK;

L NM;   B 26500 2000 13250,1000;

        B 20500 2000 13250,25500;

        B 13500 13500 13250,13250;

L NG;   B 11500 11500 13250,13250;

DF;

DS 3;

9 PADDRIVER;

C 2;

L ND;   B 26500 7000 13250,3500;

        B 7000 19500 3500,16750;

L NC;   B 1000 500 2000,1000;

L NP;   B 500 23000 2250,14000;

        B 4250 500 4125,25250;

        B 22500 500 13250,2500;

L NM;   B 1000 1500 3250,7000;

        B 1000 1500 3250,13250;

        B 1000 1500 3250,19500;

L NC;   B 500 1000 3250,7000;
```

```
        B 500 1000 3250,13250;
        B 500 1000 3250,19500;
L NM;   B 3250 1000 4875,7000;
        B 3250 1000 4875,13250;
        B 3250 1000 4875,19500;
L NC;   B 1000 500 4500,1000;
L NP;   B 500 19500 4250,14250;
        B 4750 500 6375,23750;
        B 18500 500 13250,4500;
        B 500 1500 6000,25750;
L NM;   B 1500 1000 7000,3500;
        B 1000 2750 7000,5125;
L NC;   B 1000 500 7000,3500;
        B 1000 500 7000,1000;
L ND;   B 12500 7000 13250,23000;
L NP;   B 500 3000 8500,25000;
L NC;   B 1000 500 9500,1000;
        B 1000 500 10250,26000;
        B 1000 500 10250,25000;
        B 1000 500 12000,1000;
        B 1000 500 12250,26000;
        B 1000 500 12250,25000;
L NM;   B 1500 1000 13250,3500;
L NC;   B 1000 500 13250,3500;
L NM;   B 1000 2750 13250,5125;
L NC;   B 1000 500 14250,26000;
        B 1000 500 14250,25000;
        B 1000 500 14500,1000;
        B 1000 500 16250,26000;
        B 1000 500 16250,25000;
        B 1000 500 17000,1000;
L NP;   B 4750 500 20125,23750;
        B 500 3000 18000,25000;
L NM;   B 1500 1000 19500,3500;
L NC;   B 1000 500 19500,3500;
L NM;   B 1000 2750 19500,5125;
L NC;   B 1000 500 19500,1000;
L ND;   B 7000 19500 23000,16750;
L NM;   B 3250 1000 21625,19500;
        B 3250 1000 21625,13250;
        B 3250 1000 21625,7000;
L NP;   B 4250 500 22375,25250;
        B 500 1500 20500,25750;
L NC;   B 1000 500 22000,1000;
L NP;   B 500 19500 22250,14250;
L NM;   B 1000 1500 23250,7000;
        B 1000 1500 23250,13250;
        B 1000 1500 23250,19500;
L NC;   B 500 1000 23250,7000;
        B 500 1000 23250,13250;
        B 500 1000 23250,19500;
        B 1000 500 24500,1000;
L NP;   B 500 23000 24250,14000;
DF;
(CIF file of symbol newpadin
created by the CD package for user Rob
on Wed Apr  7 08:14:50 1982
.);
```

```
DS 502 1 1;

9 buses;

( VDD and Ground Bus Wires );

L NM;

B 26500 2000 13250 25500;

B 26500 2000 13250 1000;

DF;


DS 504 1 1;

9 mypadblnk;

( PadBlank );

L NM;

B 26500 2000 13250 25500;

B 20500 2000 13250 1000;

B 13500 13500 13250 13250;

L NG;

B 11500 11500 13250 13250;

DF;


DS 503 1 1;

9 mypadin;

( Updated Padin );

C 504 T 0 3250;

L NI;

B 1500 2000 16500 1250;

B 1500 2000 21000 1250;

B 2000 1500 13000 1500;

L ND;

B 500 24000 24000 15250;
```

```
B 2000 3250 23000 6625;

B 16750 500 15625 27000;

B 1000 2000 7000 26250;

B 19250 4500 12625 6500;

B 26500 1000 13250 29250;

B 1000 28750 500 14375;

B 1000 28750 26000 14375;

B 1000 500 15750 2250;

B 500 1500 24000 2500;

B 1000 1000 23500 1750;

B 1000 1000 14750 2000;

B 500 2000 16500 1250;

B 500 2000 21000 1250;

B 1000 1000 19250 2000;

B 1000 1500 13000 1500;

B 1500 1000 10750 2000;

B 3000 1500 20750 3000;

B 4000 1500 11500 3000;

B 1000 1500 16250 3000;

B 12750 500 15875 4000;

B 12000 500 19500 250;

B 12500 750 7250 375;

B 1500 1000 2000 29250;

B 1500 1000 4500 29250;

B 1500 1000 7000 29250;

B 1500 1000 9500 29250;

B 1500 1000 12000 29250;

B 1500 1000 14500 29250;

B 1500 1000 17000 29250;
```

```
B 1500 1000 19500 29250;
B 1500 1000 22000 29250;
B 1500 1000 24500 29250;
L NP;
B 500 2000 23250 4750;
B 500 1250 23250 8125;
B 1000 1000 23000 4250;
B 18750 500 14125 5750;
B 18750 500 14125 7500;
B 500 1750 5000 6625;
B 750 1000 22875 1750;
B 500 1000 22750 2750;
B 1000 750 14750 1375;
B 500 750 15500 1375;
B 1500 1000 16500 1250;
B 1000 500 18250 1500;
B 500 1000 18000 2250;
B 1500 1000 21000 1250;
B 500 750 20000 1375;
B 1000 750 19250 1375;
B 2000 500 13000 1500;
B 250 500 14125 1500;
B 4000 500 20750 3000;
B 5000 500 11500 3000;
B 2000 500 16250 3000;
B 1000 500 17750 3000;
B 1250 500 14625 3000;
L NM;
B 1000 4250 7000 24625;
```

```
B 1500 1000 23250 1750;
B 1000 1500 14750 1750;
B 1000 1500 19250 1750;
B 1500 1000 10750 2000;
B 1000 1500 10750 750;
B 1500 1000 2000 29250;
B 1500 1000 4500 29250;
B 1500 1000 7000 29250;
B 1500 1000 9500 29250;
B 1500 1000 12000 29250;
B 1500 1000 14500 29250;
B 1500 1000 17000 29250;
B 1500 1000 19500 29250;
B 1500 1000 22000 29250;
B 1500 1000 24500 29250;
L NC;
B 500 500 23000 4250;
B 1000 500 21250 4750;
B 1000 500 18750 4750;
B 1000 500 16250 4750;
B 1000 500 13750 4750;
B 1000 500 11250 4750;
B 1000 500 8750 4750;
B 500 1000 7000 26000;
B 1000 500 6250 4750;
B 1000 500 3750 4750;
B 1000 500 23250 1750;
B 500 1000 14750 1750;
B 500 1000 19250 1750;
```

```
B 1000 500 10750 2000;

B 1000 500 2000 29250;

B 1000 500 4500 29250;

B 1000 500 7000 29250;

B 1000 500 9500 29250;

B 1000 500 12000 29250;

B 1000 500 14500 29250;

B 1000 500 17000 29250;

B 1000 500 19500 29250;

B 1000 500 22000 29250;

B 1000 500 24500 29250;

DF;

DS 501 1 1;

9 newpadin;

( the new PadIn with metal buses overlaid );

C 502 T 0 0;

C 503 R -1 0 MX T 0 29750;

DF;

DS 6;

9 PADGROUND;

C 2;

L NM;   B 2000 4500 7500,22250;

        B 2000 4500 19000,22250;

DF;

DS 7;

9 PADVDD;

L NM;   B 26500 2000 13250,1000;

        B 13500 13500 13250,13250;

        B 2000 4500 7500,4250;
```

```
L NG;   B 11500 11500 13250,13250;

L NM;   B 2000 4500 19000,4250;

DF;

DS 4;

9 PADOUT;

C 3;

L ND;   B 1000 6750 2250,29875;

        B 2500 1000 3500,32750;

        B 2000 3000 3500,28750;

L NI;   B 1500 4000 3500,28750;

        B 1500 2000 3500,32750;

L NM;   B 1000 1500 3500,34250;

L NP;   B 1000 750 3500,33875;

L ND;   B 1500 1000 3750,34500;

L NP;   B 500 7000 3500,30250;

L NC;   B 500 1000 3500,34250;

L NP;   B 3500 500 5250,30750;

L ND;   B 500 3750 4500,34375;

        B 2000 1000 5250,29250;

        B 3000 500 6000,36000;

L NM;   B 1500 2250 5500,28625;

L NP;   B 1500 1000 5500,28000;

L NC;   B 1000 500 5500,28000;

        B 1000 500 5500,29250;

L NM;   B 1000 3750 5750,30625;

        B 3500 1000 7000,32750;

L NP;   B 500 2250 6000,27125;

        B 500 3500 6750,33500;

        B 16500 500 14750,31750;
        B 5250 500 9125,35250;

        B 13250 500 13125,29750;

        B 500 1000 6750,30250;

L ND;   B 6000 4000 10250,30750;

        B 4000 1750 9250,35375;

        B 1500 1000 8000,32750;

L NM;   B 3250 1000 8875,28000;

L NP;   B 1500 1000 8000,28000;

L NC;   B 1000 500 8000,32750;

        B 1000 500 8000,28000;

L NP;   B 500 2250 8500,27125;

L ND;   B 1500 1250 9750,28125;

L NC;   B 1000 500 9750,28000;

L NM;   B 6500 1000 13250,30750;

        B 6500 1000 13250,34250;

L ND;   B 6500 1000 13250,34250;

L NC;   B 1000 500 10750,30750;

        B 1000 500 10750,34250;

L NM;   B 2000 8500 13250,30250;

L NC;   B 1000 500 13250,30750;

        B 1000 500 13250,34250;

L NP;   B 500 1250 13250,35625;

L ND;   B 6000 4000 16250,30750;

L NP;   B 6500 500 16500,35250;

L NC;   B 1000 500 15750,30750;

L ND;   B 4000 1750 17250,35375;

L NC;   B 1000 500 15750,34250;

L ND;   B 1500 1250 16750,28125;

L NM;   B 3250 1000 17625,28000;
```

```
L NC;   B 1000 500 16750,28000;

L NP;   B 500 2250 18000,27125;

L ND;   B 1500 1000 18500,32750;

L NM;   B 3500 1000 19500,32750;

L NP;   B 1500 1000 18500,28000;

L NC;   B 1000 500 18500,32750;

        B 1000 500 18500,28000;

L ND;   B 3000 500 20500,29000;

        B 3000 500 20500,36000;

L NP;   B 500 2250 20500,27125;

L NM;   B 1000 5000 20750,30000;

L NP;   B 1500 1000 21000,28000;

L NM;   B 1500 1000 21000,28000;

L NC;   B 1000 500 21000,28000;

L ND;   B 2500 1000 23000,32750;

        B 500 3750 22000,34375;

        B 2000 3000 23000,28750;

        B 1500 1000 22750,34500;

L NI;   B 1500 4000 23000,28750;

        B 1500 2000 23000,32750;

L NM;   B 1000 1500 23000,34250;

L NP;   B 1000 750 23000,33875;

        B 500 7000 23000,30250;

L NC;   B 500 1000 23000,34250;

L ND;   B 1000 6750 24250,29875;

DF;

25 Lambda = 250;

DS 1002;

9 polycut;
```

```
42 -500,-500 500,500;

L NP;

B 1000 1000 0,0;

L NC;

B 500 500 0,0;

L NM;

B 1000 1000 0,0;

DF;


DS 1003;

9 diffcut;

42 -500,-500 500,500;

L ND;

B 1000 1000 0,0;

L NC;

B 500 500 0,0;

L NM;

B 1000 1000 0,0;

DF;


DS 1001;

9 connect;

42 -3250,0 212000,162250;

L NM;

(this is the interconnect for gnd between pads);

W 2000  30500,136750 27750,136750 27750,132500 22250,132500 22250,1000

 186500,1000 186500,136750 100000,136750;

(this is the interconnect for vdd between pads);

W 2000  -2250,26500 -2250,161250 211000,161250 211000,26500;
```

```
(this is pad in no. 6 interconnect);

L NP;

C 1002 T 27750,148500;

W 500  27750,148500 27750,128750 32500,128750;

C 1002 T 32500,128750;

L NM;

W 750  26250,148500 27750,148500;

W 750  32500,128750 32500,48250;

C 1002 T 32500,48250;

L NP;

W 500  32500,48250 46250,48250;

C 1002 T 46250,48250;

(this is pad in no. 7 interconnect);

L NM;

W 750  44750,132750 44750,131000 34500,131000 34500,109500;

C 1002 T 34500,109500;

L NP;

W 500  34500,109500 46250,109500;

C 1002 T 46250,109500;

(this is pad in no. 5 interconnect);

L NM;

W 750  26250,122000 30500,122000 30500,33750 34500,33750 34500,28250;

C 1003 T 34500,28250;

L ND;

W 500  34500,28250 46250,28250;

(this is pad in no. 4 interconnect);

L NP;

C 1002 T 28750,95500;

W 500  28750,95500 34500,95500;
```

```
C 1002 T 34500,95500;

L NM;

W 750  26250,95500 28750,95500;

W 750  34500,95500 34500,40000;

C 1002 T 34500,59750;

C 1002 T 34500,40000;

L NP;

W 500  34500,40000 46250,40000;

W 500  34500,59750 46250,59750;

C 1002 T 46250,59750;

C 1002 T 46250,40000;

(this is pad in no. 3 interconnect);

L NP;

C 1002 T 28750,69000;

W 500  28750,69000 36500,69000;

C 1002 T 36500,69000;

L NM;

W 750  26250,69000 28750,69000;

W 750  36500,69000 36500,38250;

C 1002 T 36500,58000;

C 1002 T 36500,38250;

L NP;

W 500  36500,58000 46250,58000;

W 500  36500,38250 46250,38250;

C 1002 T 46250,38250;

C 1002 T 46250,58000;

(this is pad in no. 2 interconnect);

L NM;

W 750  26250,42500 28750,42500;
```

```
W 750  38500,43500 38500,23750 46250,23750;

C 1002 T 28750,42500;

C 1002 T 38500,43500;

L NP;

W 500  28750,42500 28750,43500 46250,43500;

C 1002 T 46250,43500;

(this is pad in no. 1 interconnect);

L NM;

W 750  26250,16000 32500,16000 32500,22000 46250,22000;

W 750  32500,22000 32500,27000;

C 1002 T 32500,27000;

L NP;

W 500  32500,27000 32500,35750;

G 1002 T 32500,35750;

L NM;

W 750  32500,35750 32500,41750;

C 1002 T 32500,41750;

L NP;

W 500  32500,41750 46250,41750;

C 1002 T 46250,41750;

(this is pad in no. 8 interconnect);

L NM;

W 750  55000,131000 53000,131000 53000,129000 36500,129000 36500,98000;

W 750  97750,132750 97750,131000 60000,131000;

C 1002 T 60000,131000;

C 1002 T 55000,131000;

C 1002 T 36500,98000;

L NP;

W 500  55000,131000 60000,131000;
```

```
W 500  36500,98000 46250,98000;

C 1002 T 46250,98000;

(this is pad in no. 9 interconnect);

L NM;

W 750  55000,129000 55000,127000 38500,127000 38500,99750;

W 750  124250,132750 124250,129000 60000,129000;

C 1002 T 60000,129000;

C 1002 T 55000,129000;

C 1002 T 38500,99750;

L NP;

W 500  55000,129000 60000,129000;

W 500  38500,99750 46250,99750;

C 1002 T 46250,99750;

(this is pad in no. 10 interconnect);

L NM;

W 750  150750,132750 150750,116000;

(this is pad in no. 11 interconnect);

L NM;

W 750  177250,132750 177250,131000 166750,131000 166750,114250 164250,114250;

(this is pad out no. 12 interconnect);

L NP;

W 500  176000,92750 174250,92750;

C 1002 T 174250,92750;

L NM;

W 750  174250,92750 174250,109500;

C 1002 T 174250,109500;

L NP;

W 500  174250,109500 164250,109500;

C 1002 T 164250,109500;
```

```
(this is pad out no. 13 interconnect);
W 500  176000,66250 172250,66250;
C 1002 T 172250,66250;
L NM;
W 750  172250,66250 172250,96000;
C 1002 T 172250,96000;
L NP;
W 500  172250,96000 164250,96000;
(this is pad out no. 14 interconnect);
L NP;
W 500  176000,39750 174250,39750;
C 1002 T 174250,39750;
L NM;
W 750  174250,39750 174250,48250;
C 1002 T 174250,48250;
L NP;
W 500  174250,48250 164250,48250;
C 1002 T 164250,48250;
(this is pad out no. 15 interconnect);
L NP;
W 500  176000,13250 174250,13250;
C 1002 T 174250,13250;
L NM;
W 750  174250,13250 174250,28500 170000,28500;
C 1002 T 170000,28500;
L NP;
W 500  170000,28500 163750,28500;
C 1002 T 163750,28500;
(this is pad gnd interconnect);
```

```
L NM;
W 2000  201250,119250  171000,119250  171000,107000  168000,107000  168000,32000;
W 1000  168000,32000  168000,31000  164250,31000;
W 1000  168000,107000  164250,107000;
W 1000  168000,50750  164250,50750;
W 1000  168000,71500  164250,71500;
(this is pad vdd interconnect);
L NM;
W 2000  68750,149000  57500,149000  57500,122500  41250,122500  41250,26250;
W 1000  41250,26250  41250,25500  46250,25500;
W 1000  41250,56250  46250,56250;
W 1000  41250,112500  46250,112500;
W 1000  41250,67250  46250,67250;
W 1000  41250,84000  46250,84000;
W 1000  41250,36500  46250,36500;
W 1000  41250,45250  46250,45250;
W 1000  41250,101500  46250,101500;
(these are the node labels for all the pads);
94 vdd 68750  149000  NM;
94 gnd 198750  119250  NM;
94 PHI2SL  10000  13250  NM;
94 PHI1SL  10000  39750  NM;
94 PHI2NSL  10000  66250  NM;
94 PHI1NSL  10000  92750  NM;
94 MSKSRIN  10000  119250  NM;
94 SIGSRIN  10000  145750  NM;
94 DATSRIN  42250  149000  NM;
94 PHI2NDL  95250  149000  NM;
94 PHI1NDL  121750  149000  NM;
```

```
94 PHI1DL 148250 149000 NM;

94 PHI2DL 174750 149000 NM;

94 DATOUT 198750 92750 NM;

94 COMPOUT 198750 66250 NM;

94 SIGOUT 198750 39750 NM;

94 MSKOUT 198750 13250 NM;

C 1002 T 41250,96000;

L NP;

W 500  41250,96000 46250,96000;

DF;


C 1001 T 0,0;

DS 1004;

9 u_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,3500 0,0 3500,0 3500,3500;

DF;


DS 1005;

9 o_logo;

42 -500,-500 4000,4000;

L NM;

C 1004 T 0,0;

W 1000  0,3500 3500,3500;

DF;


DS 1006;

9 f_logo;
```

```
42 -500,-500 4000,4000;

L NM;

W 1000  0,0 0,3500 3500,3500;

W 1000  0,1750 1750,1750;

DF;


DS 1007;

9 a_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,0 0,3500 3500,3500 3500,0 3500,0;

W 1000  0,1750 3500,1750;

DF;


DS 1008;

9 d_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  3500,3500 3500,0 0,0 0,1750 3500,1750;

DF;


DS 1009;

9 e_logo;

42 -500,-500 4000,4000;

L NM;

C 1006 T 0,0;

W 1000  0,0 3500,0;

DF;
```

```
DS 1010;

9 l_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,3500 0,0 3500,0;

DF;


DS 1011;

9 p_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,0 0,3500 3500,3500 3500,1750 0,1750;

DF;


DS 1012;

9 b_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,3500 0,0 3500,0 3500,1750 0,1750;

DF;


DS 1013;

9 1_logo;

42 1250,-500 2250,4000;

L NM;

W 1000  1750,3500 1750,0;

DF;


DS 1014;
```

```
9 i_logo;
42 -500,-500 4000,4000;
L NM;
W 1000  1750,3500 1750,0;
W 1000  0,3500 3500,3500;
W 1000  0,0 3500,0;
DF;

DS 1015;
9 c_logo;
42 -500,-500 4000,4000;
L NM;
W 1000  3500,3500 0,3500 0,0 3500,0;
DF;

DS 1016;
9 g_logo;
42 -500,-500 4000,4000;
L NM;
C 1015 T 0,0;
W 1000  3500,0 3500,1750;
DF;

DS 1017;
9 eight_logo;
42 -500,-500 4000,4000;
L NM;
C 1005 T 0,0;
W 1000  0,1750 3500,1750;
```

```
DF;


DS 1018;

9 two_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,3500 3500,3500 3500,1750 0,1750 0,0

 3500,0;

DF;


DS 1019;

9 dash_logo;

42 500,1250 2750,2250;

L NM;

W 1000  1000,1750 2250,1750;

DF;


DS 1020;

9 j_logo;

42 -500,-500 4000,4000;

L NM;

W 1000  0,3500 3500,3500;

W 1000  1750,3500 1750,0 0,0 0,1750;

DF;


DS 1021;

9 n_logo;

42 -500,-500 4000,4000;

L NM;
```

```
W 1000  0,0 0,3500 3500,3500 3500,0;
DF;


DS 1022;
9 w_logo;
42 -500,-500 4000,4000;
L NM;
W 1000  0,3500 0,0 3500,0 3500,3500;
W 1000  1750,0 1750,3500;
DF;


DS 1023;
9 k_logo;
42 -500,-500 4000,4000;
L NM;
W 1000  0,0 0,3500;
W 1000  0,1750 3500,1750 3500,3500;
W 1000  2500,1750 2500,0;
DF;


DS 1024;
9 four_logo;
42 -500,-500 4000,4000;
L NM;
W 1000  0,3500 0,1750 3500,1750;
W 1000  3500,3500 3500,0;
DF;


DS 1025;
```

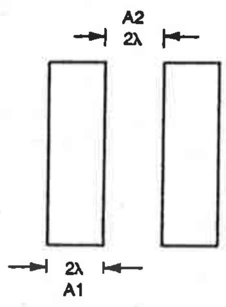
```
9 logo;

42 0,0 64000,16250;

C 1004 T 500,12250;

C 1005 T 9250,12250;

C 1006 T 14500,12250;

C 1007 T 23250,12250;

C 1008 T 28500,12250;

C 1009 T 33750,12250;

C 1010 T 39000,12250;

C 1007 T 44250,12250;

C 1014 T 49500,12250;

C 1008 T 54750,12250;

C 1009 T 60000,12250;

C 1009 T 5250,6000;

C 1009 T 10500,6000;

C 1017 T 15750,6000;

C 1018 T 21000,6000;

C 1019 T 26250,6000;

C 1005 T 31500,6000;

C 1024 T 36750,6000;

C 1019 T 42000,6000;

C 1023 T 47250,6000;

C 1009 T 52500,6000;

C 1007 T 12250,500;

C 1022 T 17500,500;

C 1019 T 22750,500;

C 1020 T 28000,500;

C 1021 T 33250,500;

DF;
```

```
C 1025 T 72500,3000;

DS 9070;

9 layout;

42 -3250,0 212000,162250;

C 501 R 0,-1 T -3250,26750;

C 501 R 0,-1 T -3250,53250;

C 501 R 0,-1 T -3250,79750;

C 501 R 0,-1 T -3250,106250;

C 501 R 0,-1 T -3250,132750;

C 501 R 0,-1 T -3250,159250;

C 501 R -1,0 T 55500,162250;

C 7 R -1,0 T 82000,162250;

C 501 R -1,0 T 108500,162250;

C 501 R -1,0 T 135000,162250;

C 501 R -1,0 T 161500,162250;

C 501 R -1,0 T 188000,162250;

C 6 R 0,1 T 212000,106000;

C 4 R 0,1 T 212000,79500;

C 4 R 0,1 T 212000,53000;

C 4 R 0,1 T 212000,26500;

C 4 R 0,1 T 212000,0;

DF;


C 9070 T 0,0;


End
```

APPENDIX E

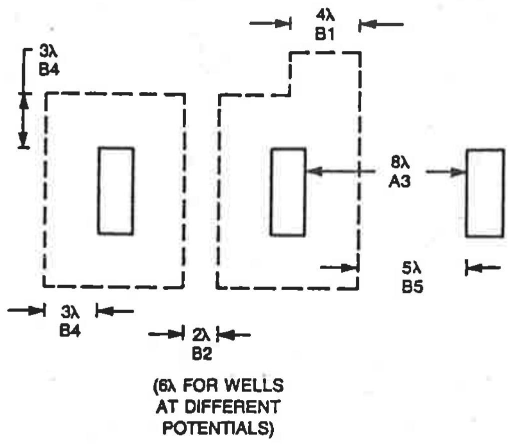
THE WESTE AND ESHRAGHIAN CMOS DESIGN RULES

These rules are copied from pages 104 - 106 of the book by Weste and Eshraghian[48].



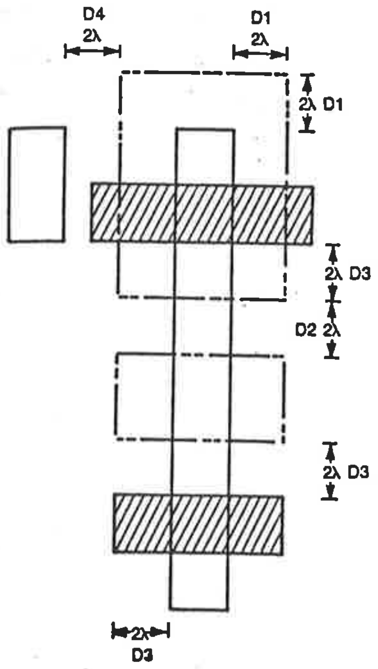
MASK 1: THINOX

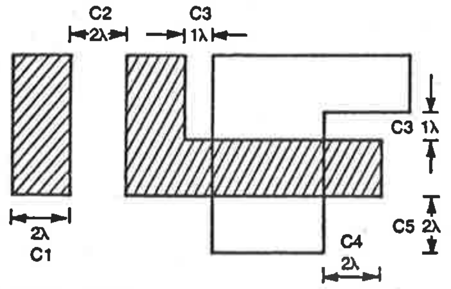
A1. MINIMUM THINOX WIDTH	2λ
A2. THINOX SPACING (n^+ to n^+ or p^+ to p^+)	2λ
A3. p^+ to n^+ SPACING	8λ



MASK 2: p-WELL

B1. MINIMUM p-WELL WIDTH	4λ
B2. MINIMUM p-WELL SPACING (SAME POTENTIAL)	2λ
B3. MINIMUM p-WELL SPACING (DIFFERENT POTENTIAL)	6λ
B4. MINIMUM OVERLAP OF INTERNAL THINOX	3λ
B5. MINIMUM SPACING TO EXTERNAL THINOX	5λ





MASK 3: POLYSILICON

C1. MINIMUM POLY WIDTH	2λ
C2. MINIMUM POLY SPACING	2λ
C3. MINIMUM POLY-THINOX SPACING	λ
C4. MINIMUM POLY GATE EXTENSION	2λ
C5. MINIMUM THINOX SOURCE/DRAIN EXTENSION	2λ

MASK 4: p - PLUS

D1. MINIMUM OVERLAP OF THINOX	2λ
D2. MINIMUM p - PLUS SPACING	2λ
D3. MINIMUM GATE OVERLAP OR DISTANCE TO GATE EDGE	2λ
D4. MINIMUM SPACING TO UNRELATED THINOX	2λ

MASK 5: CONTACT

E1. MINIMUM CONTACT AREA $2\lambda \times 2\lambda$

E2. MINIMUM CONTACT SPACING 2λ

E3. MINIMUM OVERLAP OF POLY OR THINOX OVER CONTACT λ

E4. MINIMUM SPACING TO GATE POLY 2λ

E5. n^+ SOURCE/DRAIN CONTACT

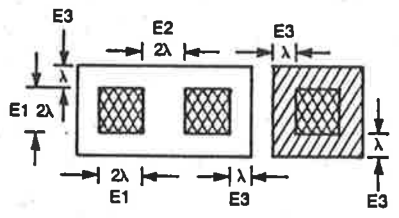
E6. p^+ SOURCE/DRAIN CONTACT

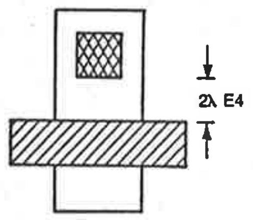
E7. V_{SS} CONTACT

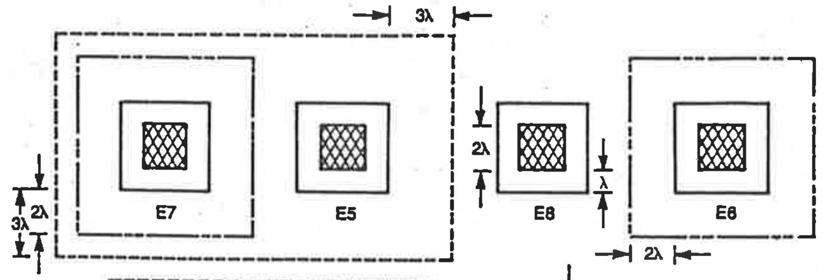
E8. V_{DD} CONTACT

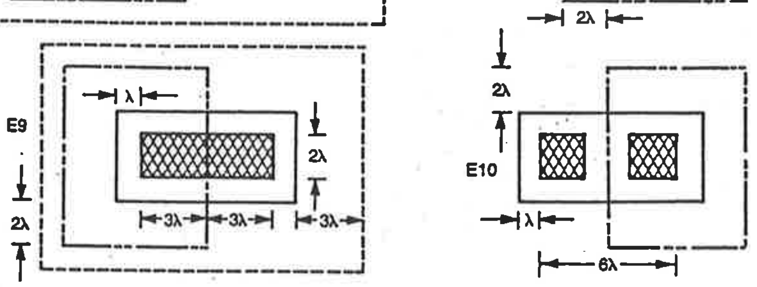
E9. V_{SS} SPLIT (OR MERGED) CONTACT (ELONGATED CONTACT SHOWN)

E10. V_{DD} SPLIT CONTACT ($2\lambda \times 2\lambda$ CONTACTS SHOWN)







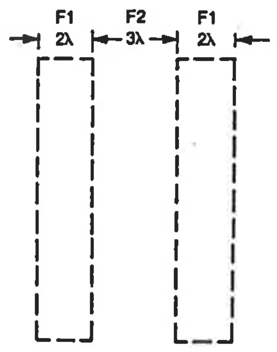


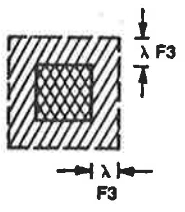
MASK 6: METAL

F1. MINIMUM METAL WIDTH 2λ

F2. MINIMUM METAL SPACING 3λ

F3. MINIMUM METAL OVERLAP λ OF CONTACT





APPENDIX F

THE ABCD, LLAMA AND CIF FILES FOR A CMOS CIRCUIT.

The listing that appears first in this appendix is the ABCD description of the CMOS carry input generation circuit to the eight bit adder discussed in Chapter IV.

```
#
# $Header$
#
# Created by ICE 1.1 on Tue May  6 17:32:17 1986
#
begin carin bbox=(18,5,24,32)
    sa_nw:      pin       metal  (18,32)
    sb_w:       pin       metal  (18,31)
    co_se:      pin       metal  (24,9)
    vss_e:      pin       metal  (24,30)
    vdd_e:      pin       metal  (24,19)
    d4:         device    p_type  (20,26)  w=2
    d5:         device    p_type  (22,26)  w=2
    d6:         device    p_type  (22,24)  w=2
    d7:         device    p_type  (20,24)  w=2
    d8:         device    p_type  (20,22)  w=2
    d9:         device    p_type  (22,22)  w=2
    d10:        device    p_type  (22,20)  w=2
    d11:        device    p_type  (20,20)  w=2
    d12:        device    n_type  (20,28)  w=2
```

```
d13:        device    n_type (22,28) w=2

d14:        device    n_type (22,13)

d15:        device    n_type (20,13)

d16:        device    p_type (20,16)

d17:        device    p_type (22,16)

            wire      metal w=2 (18,30) (24,30)

            wire      metal w=2 (18,19) (24,19)

            wire      metal (cd12D) (21,30)

cd12D:      contact   autocontact d12.d

            wire      metal (cd4D) (cd6S)

cd4D:       contact   autocontact d4.d

cd6S:       contact   autocontact d6.s

            wire      metal (cd6S1) (cd8D)

cd6S1:      contact   autocontact d6.s

cd8D:       contact   autocontact d8.d

            wire      metal (cd8D1) (cd10S)

cd8D1:      contact   autocontact d8.d

cd10S:      contact   autocontact d10.s

            wire      metal (cd10S1) (21,19)

cd10S1:     contact   autocontact d10.s

            wire      metal (cd12S) (cd4S)

cd12S:      contact   autocontact d12.s

cd4S:       contact   autocontact d4.s

            wire      metal (19,26) (cd7S)

cd7S:       contact   autocontact d7.s

            wire      metal (cd8S) (19,24)

cd8S:       contact   autocontact d8.s

            wire      metal (19,22) (cd11S)

cd11S:      contact   autocontact d11.s
```

```
                wire       metal  (cd10D)  (cd9D)

cd10D:          contact    autocontact  d10.d

cd9D:           contact    autocontact  d9.d

                wire       metal  (23,22)  (cd6D)

cd6D:           contact    autocontact  d6.d

                wire       metal  (23,24)  (cd5D)

cd5D:           contact    autocontact  d5.d

                wire       metal  (23,26)  (cd13D)

cd13D:          contact    autocontact  d13.d

                wire       poly  (22,32)  (22,16)

                wire       poly  (20,31)  (20,18)

                wire       metal  (20,18)  (23,18)  (cd17D)

cd17D:          contact    autocontact  d17.d

                wire       metal  (23,16)  (cd14D)

cd14D:          contact    autocontact  d14.d

                wire       metal  (23,20)  (24,20)

                wire       poly  (24,20)  (24,15)  (20,15)  (20,16)

                wire       poly  (22,15)  (22,13)

                wire       poly  (18,20)  (18,15)  (19,15)

                wire       metal  (cd16S)  (cd15S)

cd16S:          contact    autocontact  d16.s

cd15S:          contact    autocontact  d15.s

                wire       metal  (19,20)  (18,20)

                wire       metal  (18,31)  (24,31)

                wire       metal  (24,32)  (18,32)

                wire       metal  (22,17)  (18,17)  (18,14)

                wire       poly  (18,14)  (20,14)  (20,13)

c1:             contact    autocontact  (20,31)

c2:             contact    autocontact  (22,32)
```

```
    c3:         contact    autocontact  (18,20)

    c4:         contact    autocontact  (24,20)

    c5:         contact    autocontact  (20,18)

    c6:         contact    autocontact  (22,17)

    c7:         contact    autocontact  (19,15)

    c8:         contact    autocontact  (18,14)

    c9:         contact    vss  (21,30)

    c10:        contact    vdd  (21,19)

                wire       metal  (cd16D)  (cd14S)

    cd16D:      contact    autocontact  d16.d

    cd14S:      contact    autocontact  d14.s

                wire       metal  (cd14S1)  (21,9)  (24,9)

    cd14S1:     contact    autocontact  d14.s

                wire       metal  w=2  (18,5)  (24,5)

    c11:        contact    vss  (21,5)  or=n

end carin
```

The second listing is the LLAMA description of the carry in circuit.

```
begin name=carin tech=cmos units=2000000 elements=159 depth=0 bbox=(-7,-15,108,439)

pin metal (94,161,108,175) vdd_e

pin metal (94,379,108,393) vss_e

pin metal (98,15,104,21) co_e

pin metal (-3,407,3,413) sb_w

pin metal (-3,427,3,433) sa_nw

layer thinox
```

```
rect (43,379,57,393)

rect (16,343,86,371)

rect (13,350,16,364)

rect (86,350,89,364)

rect (16,291,86,319)

rect (13,298,16,312)

rect (86,298,89,312)

rect (16,255,86,283)

rect (13,262,16,276)

rect (86,262,89,276)

rect (16,219,86,247)

rect (13,226,16,240)

rect (86,226,89,240)

rect (16,183,86,211)

rect (13,190,16,204)

rect (86,190,89,204)

rect (43,161,57,175)

rect (13,99,89,113)

rect (13,29,89,43)

rect (43,-7,57,7)

layer poly

rect (56,422,61,438)

rect (61,93,67,438)

rect (67,422,72,438)

rect (28,402,33,418)

rect (33,139,39,418)

rect (39,402,44,418)

rect (-7,190,-3,204)

rect (-3,81,3,204)
rect (3,190,8,204)

rect (93,190,98,204)

rect (98,81,104,204)

rect (104,190,108,204)

rect (29,139,33,153)

rect (39,139,44,153)

rect (56,121,61,135)

rect (67,121,71,135)

rect (33,81,39,119)

rect (13,76,27,92)

rect (3,81,13,87)

rect (39,81,61,87)

rect (61,23,67,87)

rect (67,81,98,87)

rect (-7,51,7,66)

rect (7,55,33,61)

rect (33,23,39,61)

layer p_well

rect (5,335,97,408)

rect (5,-15,97,51)

layer p_plus

rect (39,375,61,397)

rect (12,179,90,323)

rect (9,294,12,316)

rect (90,294,93,316)

rect (9,258,12,280)

rect (90,258,93,280)

rect (9,222,12,244)

rect (90,222,93,244)
rect (9,186,12,208)

rect (90,186,93,208)

rect (9,95,93,117)

rect (39,-11,61,11)

layer metal

rect (55,421,73,439)

rect (-3,427,55,433)

rect (73,427,104,433)

rect (27,401,45,419)

rect (-3,407,27,413)

rect (45,407,104,413)

rect (-7,379,47,393)

rect (47,350,53,393)

rect (53,379,108,393)

rect (13,350,17,364)

rect (17,190,23,364)

rect (23,350,27,364)

rect (43,350,47,364)

rect (53,350,57,364)

rect (75,350,79,364)

rect (79,190,85,364)

rect (85,350,89,364)

rect (13,298,17,312)

rect (23,298,27,312)

rect (43,298,47,312)

rect (47,161,53,312)

rect (53,298,57,312)

rect (75,298,79,312)

rect (85,298,89,312)
rect (13,262,17,276)

rect (23,262,27,276)

rect (43,262,47,276)

rect (53,262,57,276)

rect (75,262,79,276)

rect (85,262,89,276)

rect (13,226,17,240)

rect (23,226,27,240)

rect (43,226,47,240)

rect (53,226,57,240)

rect (75,226,79,240)

rect (85,226,89,240)

rect (-7,190,17,204)

rect (23,190,27,204)

rect (43,190,47,204)

rect (53,190,57,204)

rect (75,190,79,204)

rect (85,190,108,204)

rect (-7,161,47,175)

rect (53,161,108,175)

rect (29,139,45,153)

rect (45,143,79,149)

rect (79,29,85,149)

rect (55,121,71,135)

rect (-3,51,3,131)

rect (3,125,55,131)

rect (13,75,17,113)

rect (17,29,23,113)

rect (23,75,27,113)
rect (43,99,47,113)
rect (47,15,53,113)
rect (53,99,57,113)
rect (75,99,79,113)
rect (85,99,89,113)
rect (-7,51,-3,67)
rect (3,51,7,67)
rect (13,29,17,43)
rect (23,29,27,43)
rect (43,29,47,43)
rect (53,29,57,43)
rect (75,29,79,43)
rect (85,29,89,43)
rect (53,15,104,21)
rect (-7,-7,108,7)
layer cut
rect (61,427,67,433)
rect (33,407,39,413)
rect (47,383,53,389)
rect (17,354,23,360)
rect (47,354,53,360)
rect (79,354,85,360)
rect (17,302,23,308)
rect (47,302,53,308)
rect (79,302,85,308)
rect (17,266,23,272)
rect (47,266,53,272)
rect (79,266,85,272)
rect (17,230,23,236)
rect (47,230,53,236)

rect (79,230,85,236)

rect (-3,194,3,200)

rect (17,194,23,200)

rect (47,194,53,200)

rect (79,194,85,200)

rect (98,194,104,200)

rect (47,165,53,171)

rect (33,143,39,149)

rect (61,125,67,131)

rect (17,103,23,109)

rect (47,103,53,109)

rect (79,103,85,109)

rect (17,81,23,87)

rect (-3,55,3,61)

rect (17,33,23,39)

rect (47,33,53,39)

rect (79,33,85,39)

rect (47,-3,53,3)

end
```

The third listing is the CIF description of the carry in circuit.

```
DS  500    50 2;

( CIF symbol 500 generated from carin.ll );

94 vdd_e 326 396 CM;

94 vss_e 326 812 CM;
```

```
94 co_se 326 0 CM;

94 sb_w 0 872 CM;

94 sa_nw 0 922 CM;

L CM ;

B  40 40 180 922 ;

B  170 20 75 922 ;

B  136 20 268 922 ;

B  40 40 100 872 ;

B  90 20 35 872 ;

B  216 20 228 872 ;

B  150 40 55 812 ;

B  20 100 140 782 ;

B  196 40 248 812 ;

B  10 40 35 752 ;

B  20 336 50 604 ;

B  10 40 65 752 ;

B  10 40 125 752 ;

B  10 40 155 752 ;

B  10 40 261 752 ;

B  20 336 276 604 ;

B  10 40 291 752 ;

B  10 40 35 666 ;

B  10 40 65 666 ;

B  10 40 125 666 ;

B  20 310 140 531 ;

B  10 40 155 666 ;

B  10 40 261 666 ;

B  10 40 291 666 ;

B  10 40 35 596 ;
```

```
B  10 40 65 596 ;
B  10 40 125 596 ;
B  10 40 155 596 ;
B  10 40 261 596 ;
B  10 40 291 596 ;
B  10 40 35 526 ;
B  10 40 65 526 ;
B  10 40 125 526 ;
B  10 40 155 526 ;
B  10 40 261 526 ;
B  10 40 291 526 ;
B  60 40 10 456 ;
B  10 40 65 456 ;
B  10 40 125 456 ;
B  10 40 155 456 ;
B  10 40 261 456 ;
B  60 40 316 456 ;
B  150 40 55 396 ;
B  196 40 248 396 ;
B  40 40 100 336 ;
B  146 20 193 336 ;
B  20 316 276 188 ;
B  40 40 180 286 ;
B  20 206 0 193 ;
B  150 20 85 286 ;
B  10 96 35 198 ;
B  20 216 50 138 ;
B  10 96 65 198 ;
B  10 40 125 226 ;
```

```
B  20 256 140 118 ;

B  10 40 155 226 ;

B  10 40 261 226 ;

B  10 40 291 226 ;

B  10 40 -15 110 ;

B  10 40 15 110 ;

B  10 40 35 50 ;

B  10 40 65 50 ;

B  10 40 125 50 ;

B  10 40 155 50 ;

B  10 40 261 50 ;

B  10 40 291 50 ;

B  186 20 243 0 ;

L CN ;

B  40 40 140 312 ;

B  266 40 163 752 ;

B  266 40 163 666 ;

B  266 40 163 596 ;

B  266 40 163 526 ;

B  266 40 163 456 ;

B  40 40 140 396 ;

B  40 40 50 226 ;

B  40 40 140 226 ;

B  40 40 276 226 ;

B  50 20 95 226 ;

B  96 20 208 226 ;

B  40 40 50 50 ;

B  40 40 140 50 ;

B  40 40 276 50 ;
```

```
B  50 20 95 50 ;

B  96 20 208 50 ;

L CP ;

B  10 40 165 922 ;

B  20 742 180 571 ;

B  10 40 195 922 ;

B  10 40 85 872 ;

B  20 576 100 604 ;

B  10 40 115 872 ;

B  10 40 -15 456 ;

B  20 316 0 318 ;

B  10 40 15 456 ;

B  10 40 311 456 ;

B  20 316 326 318 ;

B  10 40 341 456 ;

B  10 40 85 336 ;

B  10 40 115 336 ;

B  10 40 165 286 ;

B  10 40 195 286 ;

B  20 92 100 206 ;

B  40 40 50 170 ;

B  20 20 20 170 ;

B  60 20 140 170 ;

B  20 156 180 102 ;

B  126 20 253 170 ;

B  40 40 0 110 ;

B  70 20 55 110 ;

B  20 96 100 72 ;

L CPW ;
```

```
B  286 822 163 431 ;
L CPP ;
B  60 60 140 812 ;
B  286 270 163 561 ;
B  60 60 50 226 ;
B  60 60 140 226 ;
B  60 60 276 226 ;
B  30 40 95 226 ;
B  76 40 208 226 ;
L CT ;
B  20 20 180 922 ;
B  20 20 100 872 ;
B  20 20 140 812 ;
B  20 20 50 752 ;
B  20 20 140 752 ;
B  20 20 276 752 ;
B  20 20 50 666 ;
B  20 20 140 666 ;
B  20 20 276 666 ;
B  20 20 50 596 ;
B  20 20 140 596 ;
B  20 20 276 596 ;
B  20 20 50 526 ;
B  20 20 140 526 ;
B  20 20 276 526 ;
B  20 20 0 456 ;
B  20 20 50 456 ;
B  20 20 140 456 ;
B  20 20 276 456 ;
```

```
B  20 20 326 456 ;

B  20 20 140 396 ;

B  20 20 100 336 ;

B  20 20 180 286 ;

B  20 20 50 226 ;

B  20 20 140 226 ;

B  20 20 276 226 ;

B  20 20 50 170 ;

B  20 20 0 110 ;

B  20 20 50 50 ;

B  20 20 140 50 ;

B  20 20 276 50 ;

DF ;

(Outer Call made to symbol with the greatest depth);

C 500 ;

End
```