Hardware Mapping of Critical Paths of a GaAs Core Processor for Solid Modelling Accelerator

by

Song Cui, B. E., M. E.

A thesis submitted for the degree of

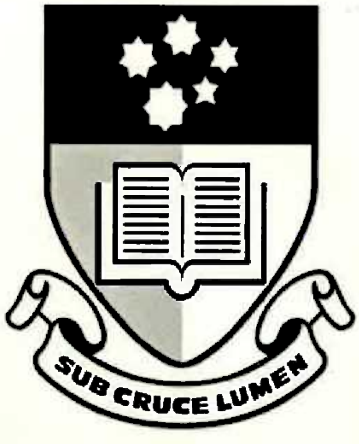
Doctor of Philosophy

in the Centre for Gallium Arsenide VLSI Technology

Department of Electrical and Electronic Engineering

The University of Adelaide

January 1996



Abstract

The field of solid modelling has been of great interest for many years. The ability to design, analyse and represent graphically three dimensional objects is highly desirable for all CAD/CAM systems. Several sophisticated solid modelling systems now exist, but none is able to process objects of useful complexity in real time [8][9]. A large class of problems share a common three-dimensional numerical structure and require numerous calculations on 3D vectors. The aim of this Ph.D thesis is to design and implement the hardware mapping of critical paths of a GaAs Core Processor for a Solid Modelling Accelerator. This solid modelling accelerator is to be designed using GaAs/CMOS/BiCMOS unified technology. High speed GaAs technology is used in the core processor to deal with floating point intensive calculations, while CMOS technology is used where high speed outputs are not required such as for frequent accesses to heavily interlinked high density data structures.

In this project, a solid modelling program called GWB was studied first to identify those operations in solid modelling systems which are most amenable to hardware acceleration. This study showed a requirement for a core processor with high speed arithmetic process elements, namely, floating point adder/subtracter, multiplier, divider and square root function. The design of a GaAs Core Processor commenced with characterization of suitable logic families and development of a design approach to produce high speed, high density and low power dissipation GaAs VLSI IC's. These have been achieved by:

- Evaluating and comparing logic families such as Direct Coupled Logic (DCFL), Source Follower DCFL (SDCFL), Super Buffer (SBFL) and Two-phase Dynamic Logic (TDFL).

- Using a novel mixed dynamic/static approach to implement an 8-bit serial divider as a test bench to optimize the speed, power and area.

- Investigating various fixed and floating point multiplier algorithms in terms of delay, power and area to select a suitable architecture for the GaAs Core Processor implementation. Developing T1P (Trailing-1's Predictor) rounding technique to speed up floating point multiplication.

· Developing a new layout approach, called Modified Ring Notation (MRN) to implement a 32-bit floating point multiplier, improving layout density and speed. The analysis of this new layout approach includes interconnections, power supply and ground considerations. The comparison between the MRN and the original Ring Notation showed that the MRN maintained the advantages of the original while improving the layout density by factor of up to 2 to 3 and also improved the speed because of shorter interconnection lines. The MRN floating point multiplier had better performance than the other floating point multipliers reported in the literature.

· Implementing, fabricating and testing a 16×16-bit multiplier chip to test the arithmetic architecture and the MRN layout methodology. Unfortunately because of a bug in the foundry's software, the chip has had to be sent for refabrication. Therefore, the test results will not be included here. We plan to report the test results later when the chip is available.

Statement of Originality

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

I consent to this thesis being made available for photocopying or loan.

SIGNED: DATE: 9/8/96

Acknowledgements

I would like to express my gratitude to the following people at University of Adelaide.

Prof. Kamran Eshraghian, my supervisor, for his valuable guidance, support and encouragement throughout this research project. His much needed comments and criticism on the draft of the thesis are also greatly appreciated.

My supervisor **Michael Liebelt**, for his time and much helpful advice on this project, as well as on the preparation of this thesis. His constructive feedback and suggestions are also very much appreciated.

Dr. Neil Burgess, for his co-operation and discussions on the arithmetic algorithms. I am very grateful to his important guidance and advice in various stages of this project.

My colleague **Andrew Beaumont-Smith** for his help and support in the use of design tools as well as suggestions on packaging and testing the chip.

Mr. Braden Phillips, for his time and great help for testing the chip.

Mr. Alireza Moini, for his useful information and assistance during this project.

I also would like to thank **Prof. Roberto Sarmiento** of CMA, University of Las Palmas for his help in fabrication of the chip.

The assistance provided by many other staff and postgraduates in this department, especially in the VLSI team is much appreciated.

I would like to thank **Prof. A. W. Thomas** of the Physics department. I am very grateful to his encouragement and support for my postgraduate study.

Finally, I would like to thank my husband **Guo-qiang Liu**, for his continual support and encouragement and my daughter **Alice Liu**, for her understanding when I was not available for her.

This project was supported by Australian Research Council.

List of Author's Related Publications

[1] Cui S., Burgess N., Liebelt M. J. and Eshraghian K. "A GaAs IEEE Floating Point Standard Single Precision Multiplier", *Proceedings of the IEEE 12th Symposium on Computer Arithmetic*, Bath, England, July, 1995, pp. 91-97

[2] Cui S, Liebelt M. J. Eshraghian K and Burgess N. "A Fast, High Density GaAs 16X16-bit Multiplier Chip for a Solid Modelling Accelerator", *Proceedings of 13th Australian Microelectronics Conference (MICRO'95)*", Adelaide, South Australia, July, 1995, pp. 328-333

[3] Cui S, Burgess N., Liebelt M. J. and Eshraghian K. "A 32-bit GaAs IEEE Floating Point Multiplier Using Trailing-1's Rounding Algorithm", *Proceedings of the IEEE Electronics Technology Directions To The Year 2000 International Conference*, Adelaide, May, 1995, pp. 246-252

[4] Cui S., Eshraghian K. and Liebelt M. J. "A Serial GaAs Dynamic/Static Divider for a Solid Modelling Accelerator", *Proceedings of Australian Microelectronics conference, Queensland*, Oct., 1993, pp. 249-254

[5] Moore, B., Betts, N., Cui, S., Liebelt, M. and Eshraghian, K., "Memory Architecture for a Solid Modelling Accelerator in Unified GaAs/BiCMOS/CMOS Technology", *Proceedings of Australian Microelectronics conference*, Queensland, Oct., 1993, pp. 209-214.

CHAPTER 1

Concepts of the Solid Modelling Accelerator

Solid modelling is playing an increasingly important role in CAD/CAM systems. There are currently several commercial solid modelling systems available, but they are not able to process objects of useful complexity in real time [3][4]. A large class of problems share a common three-dimensional numerical structure and require numerous calculations on 3D vectors. This thesis concerns the development of key sections of a Solid Modelling Accelerator using GaAs/CMOS/BiCMOS unified technology. High speed GaAs technology is used in the core processor to deal with floating point intensive calculations, while CMOS technology is used where very high speed processing is not required such as for frequent accesses to heavily interlinked high density data structures.

In this chapter, the basic concepts of solid modelling and the solid modelling accelerator are introduced. The partitioning of the solid modelling accelerator and the scope of the thesis are described.

1.1 Solid Modelling

1.1.1 Introduction

Solid modelling is the name given to a set of techniques used to represent and manipulate 3-dimensional shapes in computer systems. It is a field of increasing importance because of growing demand for CAD/CAM systems which are capable of manipulating and performing a wide range of operations on three dimensional objects. A solid modelling system for creating and managing solid models is the kernel of any advanced CAD/CAM system, and as such supports many activities in CAD/CAM [6].

Research in solid modelling became visible in the mid-1960's, and by the mid-1970's a first generation of experimental systems had appeared. These early systems, while quirky and limited in various ways, elicited great interest because they promised a level of representational tasks — detail drafting, mesh generation for finite-element analysis, verification of NC programs — to be automated fully. By the late 1970's enough had been learned about solid-modelling theory, key algorithms, and pivotal applications to enable a second generation of systems to be designed for industrial service in the 1980's. The last decade has seen explosive growth in research and applications.

Despite steady progress, there are still some significant remaining problems. Many of these relate to the representation of geometric objects of various classes [7][8][10]. However the issue with which this project is concerned is the lack of real time performance of solid modelling systems. The operations which are typically performed in solid modelling systems are highly compute intensive and as the complexity of objects increases such systems are not able to operate in real time [8][9]. Solid modelling systems, although powerful, consume substantial computational resources. The dramatic increase in the speed-to-cost ratio of computing equipment over the last few years strongly affects the industrial applications of solid modelling. Companies now find it cost effective to provide large numbers of engineers with workstations capable of running solid modellers at interactive speeds for moderately complex objects [11].

Unless competition is performed in parallel in hardware, real time operation of solid modelling system is not possible [12]. The target of the solid modelling project is to produce a Solid Modelling Accelerator which will allow solid modelling systems

running on a general purpose workstation to manage objects of typical complexity in real time.

1.1.2 The Primary Solid Representation Schemes

Within the field of solid modelling, many different representational forms have evolved [13][14]. Two of these forms have come to dominate the field because of their generality and the number of useful algorithms that have been developed for them: Constructive Sold Geometry (CSG) and Boundary Representation (B-Rep).

In a CSG based system solid objects are represented as a combination of primitive objects such as cylinders, cones, spheres, and blocks. A CSG data structure can be represented by a tree wherein the leaf nodes are primitive objects and non-leaf nodes are the boolean operations, such as *union*, *intersection* and *difference* [10]. Figure 1.1 shows a L-bracket with a hole represented as CSG tree. This representation is compact but the range of the solids which can be represented is limited by the nature of primitives available. Some applications require explicit operations on the surfaces of solids which requires the computationally expensive calculation of lines of intersection of the constituent primitive objects.

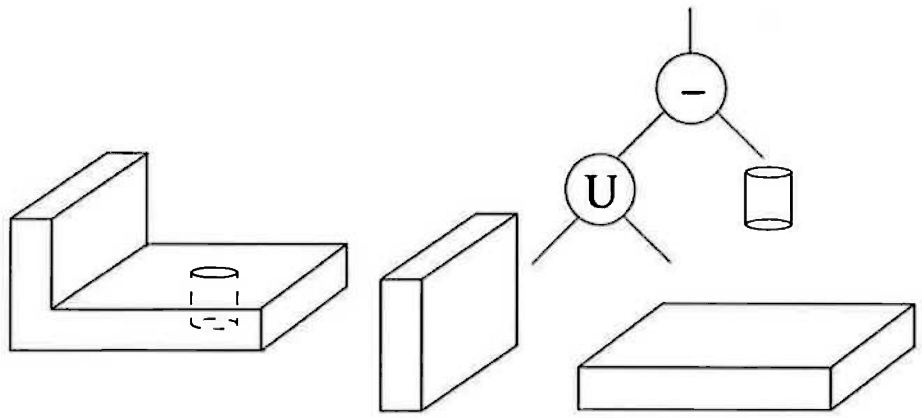


Figure 1.1 CSG representation of a simple object

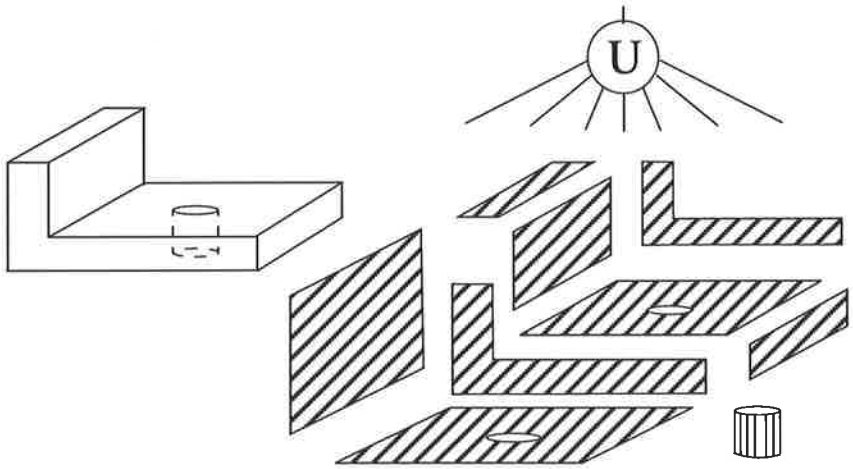


Figure 1.2 Boundary representation of a simple object.

Boundary Representation systems represent an object as group of topologically connected surfaces or faces, associated with which are edges, vertices and loops (a closed sequence of edges around or within a face). Figure 1.2 shows the L-bracket of Figure 1.1 represented as a B-Rep. A popular method of representing object topology uses the Winged Edge data structure which for each edge contains pointers to the two vertices of the edge, the four adjacent edges and the two adjacent faces as illustrated in Figure 1.3 [15].The amount of adjacency information stored in representation of the features of an object is a matter of compromise between convenience of access and storage space requirements. Variations to the winged edge have been proposed on these grounds [16]. Compared to CSG representation, the Boundary Representation of an object is cumbersome but the ability of B-Rep to describe a wider class of objects and to provide direct access to descriptions of the boundaries of objects usually outweighs the disadvantage.

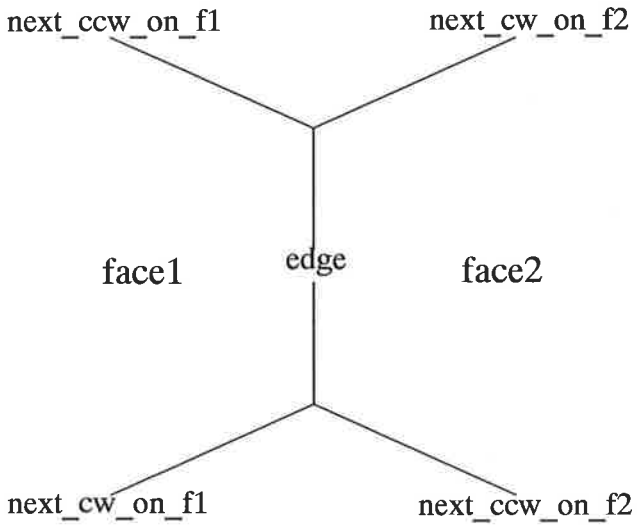


Figure 1.3 The winged-edge data structure

Boolean operations to construct composite B-Rep from other B-Rep are possible, but compared with such operations on CSG representations, the algorithms are complex and slow. An important requirement is to discover which parts of the faces of the component objects are to be included in the composite object. This requires the determination of where the component objects intersect and whether new vertices, edges or faces are created by the intersection. This can be done using set classification algorithms which typically involve reasoning based on geometric algebra [17]. Calculation of the curves of intersection of two faces can itself be difficult when the faces are not planar.

Local operations, wherein B-Rep objects are modified by creating and deleting faces, edges, vertices etc. Using Euler operators are also possible [18]. These are relatively simple to implement but can produce invalid objects, therefore it is necessary to check for geometric consistency. This involves geometric processing similar to that required from boolean operations.

Because B-Rep algorithms encompass a broad range of operations which would map into the hierarchical speed regime of unified technology as described later, and also because B-Rep system has greater flexibility, we are specifically researching the development of a boundary representation solid modelling accelerator.

1.1.3 Characterization of a B-Rep Solid Modelling System

It was necessary to analyse the resource requirements and computational behaviour of a B-Rep system to quantify the characteristics discussed above. The Geometric WorkBench (GWB) [19] was selected as the target B-Rep system on which to base initial development of the Solid Modelling Accelerator. GWB is a polyhedral B-Rep system based on single precision floating point geometry for which the source code is available. Access to source code can produce more information about the behaviour of B-Rep systems than attempts to analyse more comprehensive systems for which the source is not available.

GWB is written in C to run on the *UNIX*^{TM} operating system. The overall software architecture of GWB forms an onion-like layered structure. In this architecture, the innermost layer consists of the internal data representation of GWB, the half-edge data structure. The next layer consists of the Euler operators and a set of basic geomet-

ric procedures. The following layer is formed by high level operations such as splitting and Boolean set operations.

The half-edge data structure contains a subset of the information contained in the winged-edge structure, including a point to a structure containing a single vertex, pointers to adjacent half-edge and pointers (via a parent loop structure) to the adjacent face structure. GWB's data structures are very pointer rich — an object with F faces occupies approximately $200F$ bytes [20]. Some of the pointers are rarely used and contribute little to the efficiency of the system. Some optimization of GWB's data structures is possible but has not been pursued.

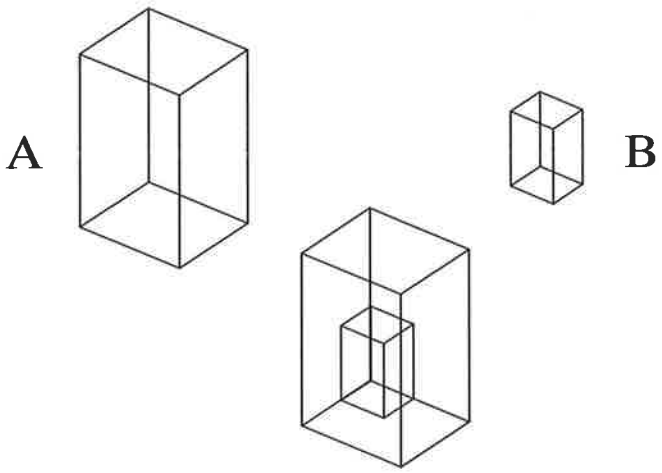


Figure 1.4 A - B

In order to identify those sections which are most computationally intensive and suitable for hardware implementation, the GPROF tracing program of Graham et al. [16] was used to profile the execution of GWB as it subtracts two blocks as illustrated in Figure 1.4. The result is shown in Figure 1.5 which reveals that almost half of execution time was spent on arithmetic calculations, such as comparing two numbers, evaluation of a face equation from the coordinates of its vertices, testing if a vertex is on an edge, etc.[1] The face equation procedure is dominated by vector cross products and normalization and is therefore floating point intensive. Clearly any effective hardware acceleration of floating point operations in general and of vector operations (dot, cross

1. The other half execution time such as "mcount" was spent on accessing to the data structure and hence on memory subsystem. The acceleration of this part is not the subject of this thesis. It is discussed in our paper[16].

product, normalization) in particular could significantly increase the performance of the system.

```
flat profile:
 %         the percentage of the total running time of the
time       program used by this function.
cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.
 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.
calls      the number of times this function was invoked, if
           this function is profiled, else blank.
 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
   else blank.
 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
   function is profiled, else blank.
name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
   the function in the gprof listing. If the index is
   in parenthesis it shows where it would appear in
   the gprof listing if it were to be printed.
granularity: each sample hit covers 2 byte(s) for 0.02% of 60.38 seconds
   %  cumulative    self              self    total
 time   seconds   seconds    calls  ms/call  ms/call name
 21.7      13.09    13.09                             mcount (151)
  9.6      18.91     5.83  2006000     0.00     0.00  _comp [15]
  8.9      24.26     5.35   118000     0.05     0.05  _faceeq [17]
  5.5      27.58     3.32   598000     0.01     0.01  _contvv [21]
  5.4      30.84     3.26   210000     0.02     0.03  _intrve [16]
  5.1      33.94     3.11   284000     0.01     0.01  _int2ee [19]
  3.3      35.95     2.01   717000     0.00     0.00  _dot [30]
  3.2      37.90     1.95   100000     0.02     0.16  _contlv [9]
  2.5      39.43     1.53   307001     0.00     0.01  _malloc [26]
  2.5      40.92     1.49   162000     0.01     0.16  _dosetopgenerate <cycle 1> [5]
  2.3      42.29     1.38   252000     0.01     0.01  _cross [38]
  1.8      43.38     1.09     4000     0.27     0.33  _sreclsectors [39]
  1.7      44.41     1.04   249000     0.00     0.01  _vecnull [24]
  1.5      45.35     0.94   342000     0.00     0.00  _dist [45]
  1.4      46.22     0.88                             _sqrt [47]
....

comp--comparing two numbers a and b (a=b or a<b or a>b)
faceeq--evaluation of face equations
contvv--comparing two vertices
intrve--testing if a vertex on an edge
int2ee--testing if two edges intesect
dot--dot product
contlv--testing if a vertex in a loop
cross--cross production
vecnull--testing if a vertex is null
dist--calculating the distance of two vertices
sqrt--square root function
```

Figure 1.5 Execution profile of A - B

1.2 Solid Modelling Accelerator Using Unified Technology

1.2.1 Unified CMOS/BiCMOS/GaAs Technology

The necessity of performing extensive geometric algebra in most operations on solid objects means that solid modelling systems are computationally intensive. Furthermore, the implementation of the winged edge boundary representation of objects as linked lists means that these systems are also memory reference intensive. A hardware accelerator for a boundary representation system must not only provide acceleration for floating point vector calculations but also include a memory architecture which minimizes the time required for data structure accesses.

Over the past several years, CMOS technology has become the dominant fabrication process for relatively high performance and cost effective VLSI circuits and systems. For example, today's micro-processors are able to handle of order of 10^5 operations a second, but the circuitry that manages communications with the memory and with other processors is too slow to keep up with this data rate. The speed/power limitations have brought the need for development of other technologies such as BiCMOS and Gallium Arsenide.

BiCMOS opens up new possibilities for VLSI performance implementation. Bipolar and CMOS devices can be fabricated on the same substrate to form a technology that offers the advantages of both type of devices. BiCMOS employs low power and high density CMOS devices to implement memory and logic functions, register, multiplexer, etc. and fast bipolar transistors to driver large capacitive loads and implement sense amplifiers.

Gallium Arsenide is attractive because of its high electron mobility and low parasitic interconnection capacitance, both of which improve the speed performance. The unified CMOS/BiCMOS/GaAs technology is based in the proposition that GaAs technology may be utilized for core processor sections of high speed stream processors for digital data. This fast data stream can be subdivided into lower rate parallel streams suitable for processing in silicon CMOS/BiCMOS subsystems at lower rates. The Solid Modelling Accelerator encompasses a broad range of operations which would map into the hierarchical speed regime of unified technology.

1.2.2 The Architecture of the Solid Modelling Accelerator

Figure 1.6 illustrates the architecture of the proposed Solid Modelling Accelerator based on the information gathered about the behaviour of GWB [22]. In particular the memory architecture was explicitly designed to exploit the topological locality of data references exhibited by GWB's algorithms. It is expected that similar locality will be found in other systems based on the winged edge data structure. However the architecture was designed specifically not to depend on any peculiarities of GWB.

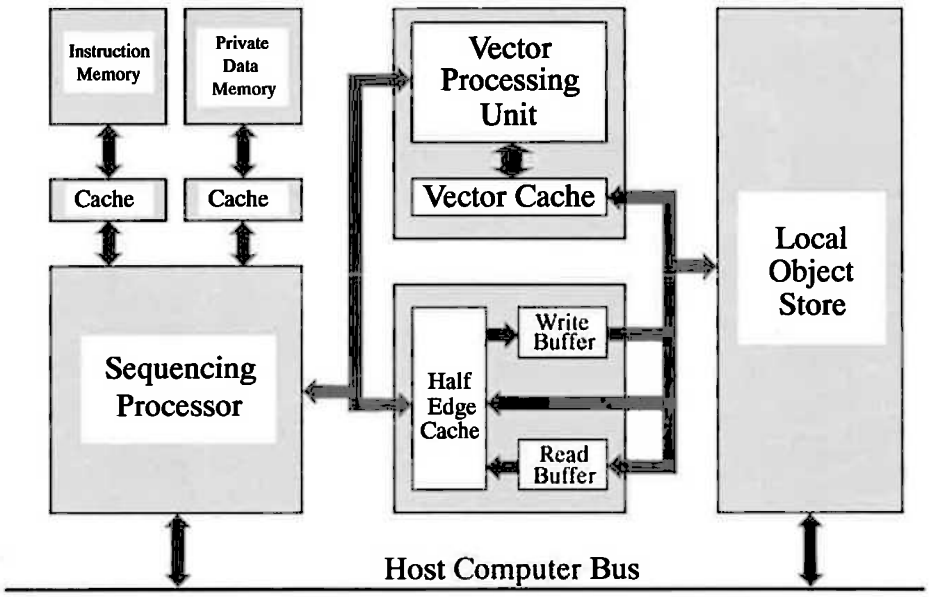


Figure 1.6 Solid Modelling Accelerator block diagram.

The accelerator shown in Figure 1.6 is based around a high speed Vector Processing Unit (VPU) which performs the compute intensive geometric floating point calculations. This processor manipulates vectors with four single precision floating point coordinates. These may be vertex coordinates in homogeneous space, face equations or any other numeric data needed.

The VPU is controlled by a Sequencing Processor (SP) which performs the list traverse and other general purpose tasks involved in carrying out modelling algorithms. It is envisaged that the SP will be either a general purpose microprocessor or a custom microcoded engine that will have a writable instruction store and a private memory for temporary data storage.

When being operated on by the accelerator the solid data structure will be loaded into a large Local Object Storage (LOS). This is necessary because of the high bandwidth access to the data structure required. The alternative of accessing the object across the bus to the host computer would cause an insurmountable bottleneck.

To perform modelling commands, the host processor will first load the objects into the LOS and write appropriate code into the instruction store. It will then be able to send high level commands, such as boolean operations and rotational sweeps, to the accelerator. These commands will be interpreted by the SP as directed by the code in its instruction memory. The operands will most often contain one or more object identifiers, which will be the LOS address of the solid data structure. The SP must follow the pointers of the interlinked data structures in the LOS, sending commands to the VPU to operate on numeric data, such as vertex coordinates, when appropriate.

The task of the memory architecture is to supply numeric data to the VPU and pointer data to the SP quickly enough to maintain operation at close to peak throughput. There are two caches, one for each processor. This allows each to be tailored to the specific needs of its processor. To simplify the architecture and allow useful caching strategies it was decided to restrict the data structure elements, such as edges and faces, to being aligned to vector cache line boundaries. Furthermore, within each data structure element any vector, such as a face equation, must be aligned.

The vector cache line size was chosen to match the size of a vector and therefore each line contains one complete vector because of the LOS alignment restrictions. This is an appropriate choice for the line size since vectors generally have little or no spatial locality (in GWB they are never adjacent). Hence temporal locality must be relied on. The cache uses a write through strategy as analysis of GWB showed that vectors are never written to twice in a short space of time. It is planned to have the vector cache on chip with the VPU, which will limit its size but allow a very wide (up to full line width) bus to be used.

The data accessed through the half edge cache also has poor spatial locality and thus a high miss rate is to be expected if a conventional cache is used. To reduce the miss rate data is prefetched into the cache using knowledge about which data structure elements will be accessed next. This is achieved by having a prefetch control line from the SP to the half edge cache. When the cache is accessed with this line high the data

transferred is taken as an address which is then prefetched into the cache by a read buffer which operates in parallel with the rest of the cache. If prefetching is completed before the data is needed then a miss can be avoided entirely. If not, then at least one, most likely more, clock cycles will be shaved off the miss penalty. Note that the alignment of data structure elements to cache line boundaries assures that a greater proportion of each cache line will contain useful information. Consistency between the vector and half-edge caches is not an issue as they never access the same data.

1.2.3 Partitioning

Because of the superior performance of digital GaAs circuits in terms of speed and power dissipation, GaAs technology is used in the core sections (VPU and Vector Cache) of the solid modelling accelerator to implement the arithmetic elements, while CMOS technology may be used where high speed outputs are not required such as for frequent accesses to heavily interlinked high density data structures. BiCMOS will be used as buffer to drive large loads in CMOS circuits. The approach necessitates the solution of some interface problems between GaAs and Silicon. However, this thesis will concentrate on the GaAs VPU only.

1.3 The Scope of this thesis

At the current state of the art all the arithmetic elements can not be implemented into one single GaAs chip, a *modular* architecture has to be used. Ideally, modules should be designed such that all of the high-speed components needing to communicate with each other are in the same module. The implementation goal will be to get all parts of a module on a single chip. Modules can be packaged as a hybrid semiconductor to communicate with each other. This has led to characterization of suitable logic families and development of a design approach to produce high speed, high density and low power dissipation GaAs VLSI ICs.

The complete design of such GaAs Core Arithmetic Processor constitutes a task well beyond the scope of this thesis. Instead this thesis is to identify the critical design issues, ranging from the optimization of logic families to the impact of the algorithms and overall layout architecture on the performance of GaAs VLSI circuits. These have been done by investigating four GaAs logic families and using them to implement two arithmetic elements: an 8-bit serial divider and a 32-bit IEEE float point multiplier.

Then a 16x16-bit fixed point multiplier chip which contains some testing circuits for the divider is sent for fabrication. The divider is used as a test bench for a mixed dynamic/static approach, the multiplier uses a new layout approach called Modified Ring Notation (MRN) approach to produce high speed, high density and low power dissipation GaAs VLSI ICs. Although the designs are primarily targeted for the Solid Modelling Accelerator, in principle they have much wider applications.

In Chapter 2 general information about GaAs is introduced, and then four GaAs MESFET logic families are discussed. TDFL (two-phase Dynamic FET Logic) is used in the mixed dynamic/static approach. DCFL (direct Coupled FET Logic) is used as the most appropriate one for static logic implementation. SDCFL (Source Follower DCFL) is used as a buffer to driver high fanout loads. SBFL (Super Buffer FET Logic) is used to drive very high fanout loads such as clock signals. Interconnection issues and cross-talk are also discussed in this chapter which are the theoretical foundation of the new layout style described in Chapter 5.

Chapter 3 describes an 8-bit serial divider as a test bench for mixed dynamic/ static approach. TDFL is used as shift registers, and a DCFL based BLC (Binary Look-ahead Carry) adder is used as the static part. This mixed approach is promising to optimize speed, power and area.

Chapter 4 reviews various fixed and floating point multiplier algorithms in terms of delay, power and area in order to select a suitable architecture for the GaAs Core Processor implementation. A multi-carry select adder is designed in the final adder and a new rounding technique called T1P (Trailing-1's Predictor) is developed to improve the speed of the floating point multiplier.

A novel layout approach called Modified Ring Notation is developed in Chapter 5 to improve the GaAs layout density. The original Ring Notation approach [55] has the major advantage of minimizing coupling between power and signal buses and allows the designer to optimize easily the layout for speed-area-power performance. A comparison between Modified Ring Notation and original Ring Notation is given. It shows that the Modified Ring Notation maintains the advantages of the original and reduces the area by factor of up to 2 to 3.

Chapter 6 describes a 16×16-bit multiplier chip implemented by the modified Ring Notation approach which shows a very high density and very fast speed. The HSpice simulated results shows that it is the fastest multiplier among other 16×16-bit multipliers previously reported. Power/ground design and crosstalk analysis are carried out. The interface between a GaAs chip and a CMOS chip are investigated and bonding issues and test results are described in this chapter.

Finally, the overall objectives and the work carried out during the course of the research are summarised in Chapter 7. The outcomes together with the conclusions drawn from the research are also presented.

CHAPTER

2

GaAs Technology and Logic Circuit Design

The aim of the work described in this chapter is to look for ways to harness the speed advantage of the GaAs transistor in order to design fast, low power circuits. This was done by:

- reviewing the advantages of GaAs and comparing it with Silicon;
- optimizing four logic families which will be used as the bases for the GaAs circuits design;
- analysing the interconnection models and applying them in the GaAs layout.

2.1 Gallium Arsenide Technology

2.1.1 Introduction

Gallium Arsenide is a compound semiconductor which was first discovered in 1926 [24]. In the early stages of development, GaAs devices were used as discrete components in microwave circuits and light emission applications in the late 1960's [25]. The development of digital integrated circuit fabrication technology in the 1970's made integrated GaAs products a possibility and finally as the result of significant advances in ion implantation in the 1980's, GaAs VLSI technology is a commercial reality in the 1990's.

2.1.2 Advantages of GaAs

GaAs transistors have two distinct advantages over Si transistors: speed and power. For the same power dissipation, a GaAs circuit is usually faster, and at the same speed, the power in a GaAs circuit is usually lower (at high frequencies). The speed advantage comes from the fact that the peak average electron velocity in intrinsic or doped GaAs is several times higher than in Si and it is reached at a much lower value of electric field, and hence with a lower supply voltage. Since the current density in a device is proportional to the electron velocity, the amount of current available to charge or discharge a capacitor in a GaAs device is much larger and the switching speed is therefore higher than in a Si device with the same dimensions. In addition, a GaAs field effect transistor does not have pn-junction around its drain and source terminals and therefore the interelectrode capacitance in a GaAs device is much smaller. Smaller capacitance and higher current density, combined with a smaller voltage swing in a GaAs transistor, contribute to the realization of low-power, high-speed circuits.

2.1.3 Comparison between GaAs and Silicon

The structure of a typical GaAs MESFET and Si MOSFET are compared in Figure 2.1 [25]. The main differences between the devices are: (1) the formation of the channel and (2) the coupling of the gate-control electrode to the channel. The GaAs MESFET uses a thin, doped channel whose thickness is controlled through depletion by a metal/semiconductor junction. The metal gate directly contacts the channel. The MOSFET, on the other hand, forms a channel when the silicon surface is inverted (p-

type silicon now contains a high density of electrons due to band-bending). The gate electrode is separated from the channel by a thin oxide dielectric layer.

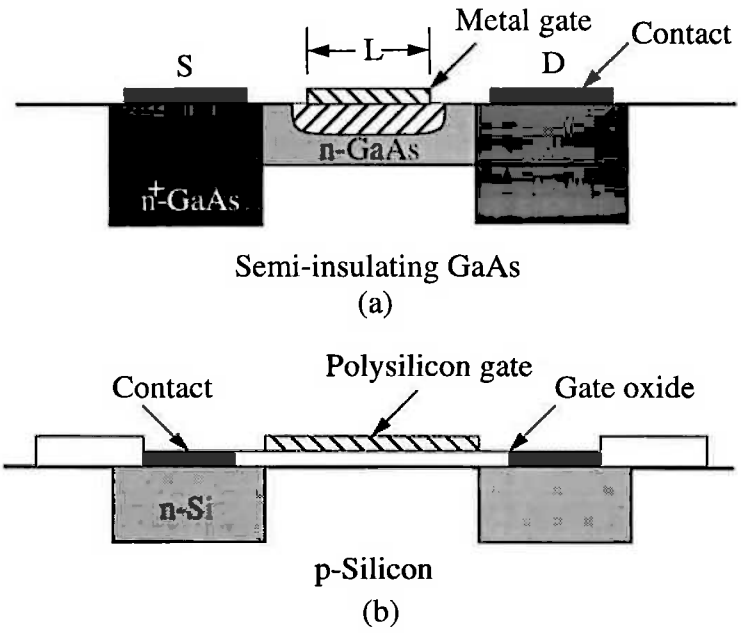


Figure 2.1 Schematic cross-sectional representations (not to scale) of (a) MESFET (b) MOSFET V_{DS} is low in both case.

The material and electronic properties of GaAs and silicon are compared in Table 2.1[25][26]. From this table it can be seen that GaAs has a relatively large band gap. This means that it can operate at relatively high temperatures. Its intrinsic carrier concentration is low so the material is a semi-insulator. Its resistivity is large so no special measures need be taken to provide isolation between devices on the chip. A direct band gap leads to short life time of minority carriers so that electron-hole pairs generated by radiation, for example, will recombine quickly before they can cause degradation of circuit performance. Its high electron mobility gives rise to high cut-off frequency of amplifiers and fast switching speed of digital circuits. On the other hand, it hole mobility is disproportionately low so that it is not practical to construct complementary circuits such as Si CMOS. In addition, GaAs does not have a native oxide so that it is not possible to build a MOS-like structure. Si_3N_4 is used as insulator on a GaAs substrate. It has a dielectric constant of 7.5 compared to 3.9 for SiO_2 so it has a higher capacitance for the same area. The thermal conductivity is relatively low so a wafer is often thinned to prevent excessive temperature rise. GaAs is also very brittle and yield loss through handling is significant.

Table 2.1 Electronic properties of GaAs and Si [25][26].

Property	GaAs	Silicon
Bandgap	Direct; 1.42 eV	Indirect; 1.12 eV
Low-field electron drift mobility	5000 $cm^2/(V \cdot s)$ at $N_D = 10^{17}/cm^{-3}$	800$cm^2/(V \cdot s)$ at $N_D = 10^{17}/cm^{-3}$
Saturated drift velocity E >> 10 kV/cm	$8 \times 10^6 cm/s$ (100 kV/cm)	$6.5 \times 10^6 cm/s$ (in MOS inversion layer)
Peak electron velocity	$1.7 \times 10^7 cm/s$ at E = 3.5 kV/cm and $N_D = 10^{17}/cm^{-3}$	$6.5 \times 10^6 cm/s$ at E >> 10 kV/cm
Low-field hole (drift) mobility	250 $cm^2/(V \cdot s)$ at $N_A = 10^{17}/cm^{-3}$	300 $cm^2/(V \cdot s)$ at $N_A = 10^{17}/cm^{-3}$
Substrate resistivity	10^6 to $10^8 \Omega \cdot cm$	Low
Surface state density	High ($10^{12}/cm^{-2}$ or greater)	Low ($10^{10}/cm^{-2}$)
Native oxide	Several reactive and unstable compounds of Ga and As	S_iO_2; very stable
Effective electron mass	0.063 m_0	0.33 m_0
Effective hole mass	0.090 m_0	0.16 m_0
Dielectric constant	13.1 ε_0	11.9 ε_0
Minority carrier lifetime	$10^{-8} s$	$2.5 \times 10^{-3} s$
Thermal conductivity	0.46 W/cm-C°	1.5 W/cm-C°
Intrinsic Debye length	2290μm	24μm
Breakdown field	$4 \times 10^5 V/(cm)$	$3 \times 10^5 V/(cm)$
Schottky barrier height	0.7 ~ 0.8 V	0.4 ~ 0.6 V

2.2 GaAs Logic Families

2.2.1 Introduction

The current-voltage characteristics of a GaAs MESFET are similar to those of an nMOS transistor. This suggests that many of the design techniques that have been developed for nMOS digital circuits can be applied directly to the design of GaAs digital circuits. However, there are fundamental differences between a GaAs transistor and a nMOS transistor. Foremost among them is that in a GaAs device, when the gate voltage exceeds the Schottky barrier voltage, the gate becomes conducting and the gate current has detrimental effects. Special design techniques must be developed and GaAs logic circuits are somewhat more complicated than comparable NMOS circuits. In

addition, when the gate becomes conducting, its voltage is clamped at a value equal to the barrier voltage. This limits the logic swing to a fraction of the supply voltage and places a severe limitation on the noise margins of logic circuits. Other differences will become apparent as the logic families are introduced.

There exists a large number of logic families in GaAs MESFET technology. They are classified into two main categories: the depletion mode (normally-on) and the enhancement mode (normally-off) logic families[24][25].

The depletion mode GaAs FETs (*dfets*) are characterized by unequal input and output voltage levels; V_{GS} must be negative in order to cut off the FET while V_{DS} must be positive at all times. Consequently, level-shifting networks, typically composed of forward-biased Schottky diodes, are necessary. In addition, since both positive and negative signals are required, low power supply voltages, a positive V_{DD} and a negative V_{SS}, are needed. Therefore, additional power is dissipated in the circuit, and the higher supply differentials make depletion mode logic more susceptible to backgating effects [20].

On the other hand, larger logic voltage swings can be obtained using a *dfet* because the gate electrode often varies between a negative voltage close to V_{TD} (*dfet*'s threshold voltage) and a positive voltage near 0.7V (if connected to another gate). The voltage range is determined by the choice of V_{TD}, and the amount of level shifting required. This is not the case with enhancement mode MESFET circuits (*efets*) where V_{GS} is always positive. The higher logic voltage swing of the *dfet* circuit leads to larger drive currents, but not necessarily to lower delays. It also increases noise margins, giving higher yields. Finally, the power dissipation is often higher because of the dual power supplies and larger current levels.

Logic circuits implemented with enhancement mode MESFETs have threshold voltage $V_{TE} \geq 0$V. Since both V_{GA} and V_{DS} are always positive, no level shifting from output to input is needed, and a single power supply voltage, V_{DD}, is usually sufficient. Also, the saturation voltage $V_{D,sat}$ of an *efet* is less than that of a *dfet*. Thus, power supply voltages can be lower for *efet* circuits than their *dfet* equivalent. The above properties can save power and layout area.

The gate-to-source voltage will be constrained (without the use of level-shifting or dynamic circuit techniques) to a minimum of 0 V and to a maximum 0.7 V by forward conduction. This small logic swing will provide high speed if the device transconductance is sufficiently high at small V_{GS}. This requirement emphasizes optimum design of the e-mode FET to minimize source resistance by self-alignment or recessed-gate fabrication techniques and to minimize the depth of the gate depletion layer to increase the intrinsic transconductance g_m.

The small range of V_{GS} also requires uniformity of threshold voltage. Both the mean V_{TE} and the standard deviation on a single wafer and wafer to wafer must be very tightly controlled so that the parametric yield of large circuits will be finite. In this area, circuit design techniques can play an important role in reducing sensitivity of circuits to parameter variations.

An additional problem is also encountered in the design of *efet* GaAs circuits. An active load can be provided with an *efet* by connecting gate and drain together. While this can be used as a substitute for a resistor, it is not suitable for use in this form for high-speed circuits, since the charging current diminishes as the voltage across the load capacitor increases. Also, the resistor or diode-like load line V_{out} vs. V_{in} transfer characteristic (Figure 2.2), leading in turn to small noise margins. In the GaAs MESFET wired with V_{GD} = 0V, the gate conduction also limits the maximum V_{GD}. Therefore, *dfet*s are also required for active-load current sources in most *efet* circuits which are also called E/D circuits.

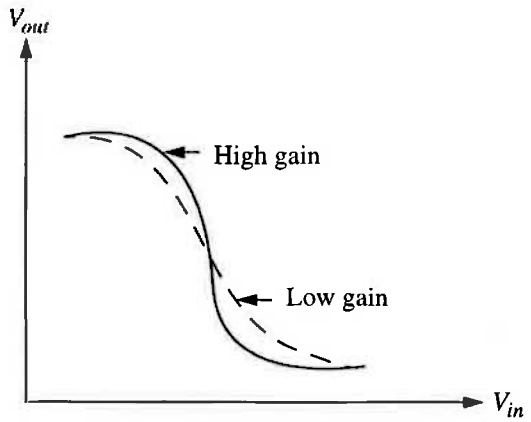
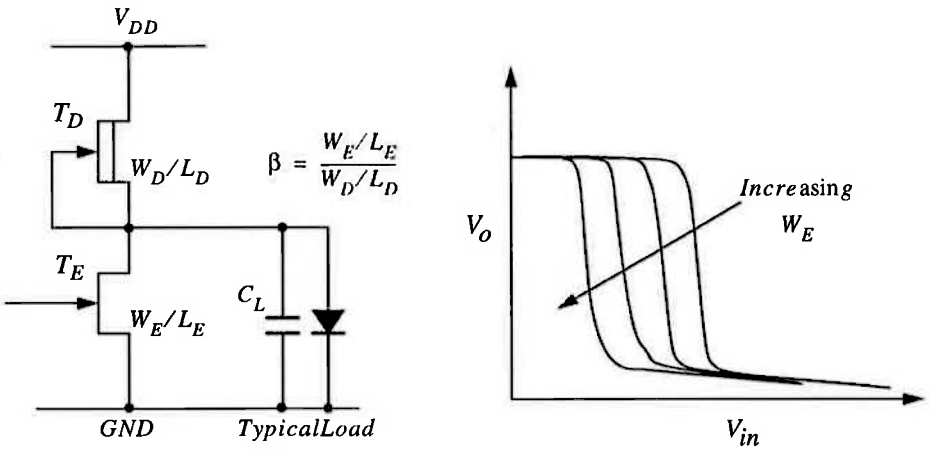


Figure 2.2 Transfer characteristics illustrate the influence of voltage gain.

Because the advantages of E/D circuits, they are commonly used in digital GaAs circuits. Four commonly used E/D logic families are chosen for the GaAs Core Processor, which are introduced as follows.

2.2.2 Direct-Coupled FET Logic (DCFL)



- W_D/L_D determined by $I = C_L\frac{dV}{dt}$ requirement
- W_E/L_E determined by restoring logic requirement (Node voltage must be $\leq V_{ol}$ to achieve proper logic low)
- Baseline speed determined by g_m/C_g of enhancement switch and by β ratio
- β ratio can be scaled to drive various capacitive loads

Figure 2.3 Summary of the basic characteristics of DCFL circuitry

The first, simplest, and most widely used e-mode GaAs logic circuit is the direct-coupled FET logic circuit (DCFL). Its inverter circuit is shown in Figure 2.3. In this diagram, the symbol of the *dfet* is distinguished from the *efet* by a double bar on the channel. The enhancement transistor T_E is sometimes called the pull-down or driver transistor and performs switching functions. While the depletion transistor T_D is called the pull-up or load transistor. Since the threshold voltage of T_D is negative, with its gate connected to the source, T_D is always conducting and it can be regarded simply as a non-linear resistor. Like nMOS, circuit performance is determined by the ratio of the transistor sizes, i.e., their comparative current-handling capability. Figure 2.3 is a summary of DCFL characteristics [27]. The DCFL logic family is characterized by low power consumption, low component count for a given logic function, and requires only a single power supply. The basic logic element is the inverter shown in Figure 2.3. Sev-

eral switching elements can be connected in parallel to form a multi-input *nor* gate. In DCFL, inverter and *nor* are the most conveniently implemented logic functions[1].

The gate of a GaAs MESFET is a Schottky-barrier diode which draws current when it is forward biased. The effect of the Schottky barrier diode gate on circuit operation is illustrated by analysing the operation of a low stage inverter chain. A schematic diagram of the inverter chain and the load-line characteristic for a single inverter are shown in Figure 2.4. A logic 1 (high voltage) applied to the input terminal causes T_2 to sink the load current I_L. Therefore, the output of the first stage inverter V_X falls to the output low voltage V_L. The value of V_L is established by the intersection of the pull-up and pull-down I-V curves. V_X rises to the output high voltage V_H, when the input voltage falls below the threshold voltage of T_2. As V_X rises, the gate-source diode of the driven pull-down transistor T_4 is forward biased and draws gate current I_L. The gate-source Schottky diode of T_4 clamps the voltage V_X at a value determined by Schottky-barrier height.

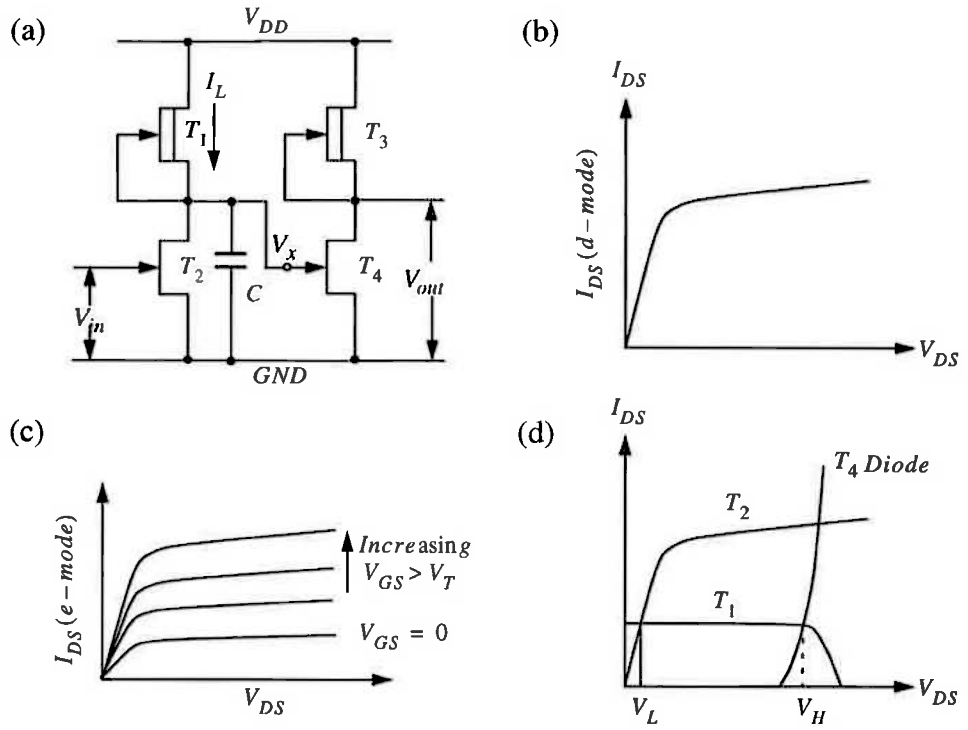


Figure 2.4 (a) DCFL 2-stage inverter chain; (b) I_{DS} - V_{DS}characteristic of the dfet; (c) I_{DS} - V_{DS} characteristic of the efet; (c) Load line characteristic.

1. Because of the low noise margin, the NAND function obtained by series connections of e-mode MESFETs is not recommended[25].

The output voltage V_{out} of the inverter chain is determined by the ability of T_4 to sink the current supplied by its load T_3. When T_4 is off ($V_X < V_T$), V_{out} is pulled up toward V_{DD}. However, when $V_X < V_T$, the current through T_4 depends on the gate voltage. If V_X is clamped at a low value by the gate-source diode, then T_4 will not sink all of the current supplied by T_3. The remaining current is available to drive the output node and V_{out} will not achieve the desired low voltage value. Therefore, the gate current requirement of the pull-down device T_4 is the constraint which determines the output low voltage of a gate.

The primary factor which determines transistor sizing for DCFL circuits, is the loading fan-out of the gate. Loading fan-out is defined as the ratio of pull-up transistor width of the driven stage to the pull-up transistor width of the gate being designed. It determines the amount of current which must be supplied to satisfy the gate current requirements of the driven stages for current DC operation and to charge the capacitance during transient operation. In addition, process tolerances, power supply variations, and operating temperature range must be considered when selecting device sizes.

2.2.3 Source-follower DCFL (SDCFL)

Logic circuits based on DCFL inverter are simple in design and efficient in area and power. However, the noise margin of DCFL gates is rather small, limited on the high side by the forward gate conduction of the next stage and on the low side by the saturation voltage of the *efet*. Besides having low noise margin, DCFL also has relatively high sensitivity to fan-out and capacitive loading [25].

Efforts to improve on these shortcomings, particularly important for large-scale integration where uniformity is critical and gate array applications where loading is large, have resulted in a Source-follower DCFL (SDCFL) circuit [55]. SDCFL is logic family which is well suited for driving large loading. It is formed by adding a source follower output stage to the DCFL gate previously described. The schematic diagram of a SDCFL 2-input NOR gate is shown in Figure 2.5. The SDCFL circuit has several powerful advantages. A source follower output driver lowers the minimum output voltage V_{OL}, which means that smaller β-ratios (transconductance parameter) can be used. The source follower also isolates the logic function of the circuit from the output and provides the necessary current to drive a large load either in the form of a capacitor or a

number of fan-out transistors. The improvement comes at a cost of increased circuit complexity and power dissipation than DCFL and slightly longer delay for similarly sized devices.

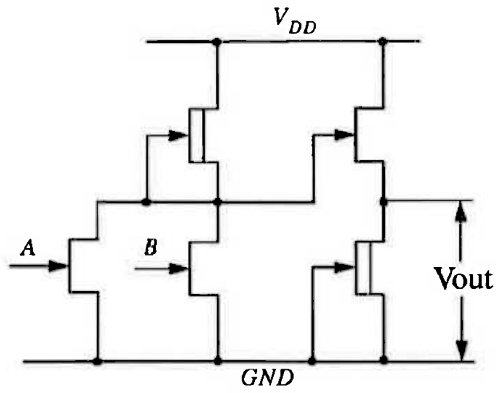


Figure 2.5 Schematic diagram of a SDCFL 2-input NOR gate.

In addition to inverter and *nor* gates, SDCFL also supports And-Or-Invert (*AOI*) structures to give increased function complexity per gate. A schematic diagram of such structure is illustrated in Figure 2.6. In this *AOI* structure, the output of two SDCFL gates are connected together to form an *or* operation. In another words, an *or* can be used between two DCFL gates without incurring extra delay. The *AOI* structure enables very high speed operation as there is no gate delay involved and a very compact layout can be achieved.

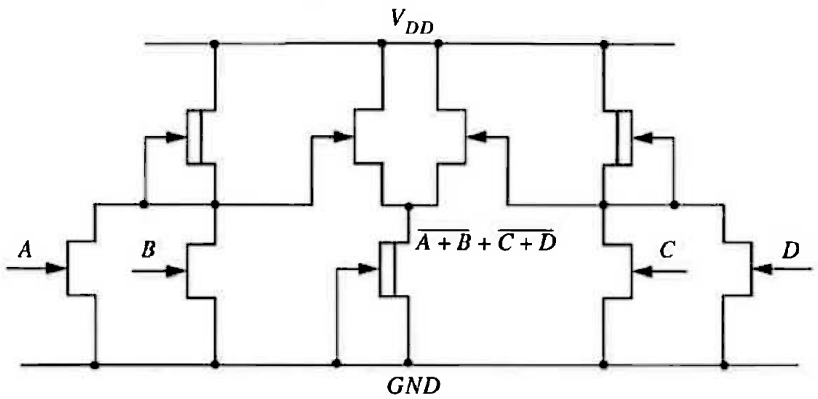


Figure 2.6 And-Or-Invert (AOI) structure in SDCFL.

A possible design approach is to utilize clusters of DCFL gates for logic operations and to use SDCFL in critical paths and higher fan-out applications.

2.2.4 Super Buffer FET Logic (SBFL)

In a SDCFL inverter, the depletion transistor of the source follower acts as a fixed nonlinear resistor whose resistance is a function of its drain voltage only and is not controlled by any external signal. The performance of the inverter would be improved if the nonlinear resistance could be made (a) smaller as the load capacitor is discharged through it to reduce the pull-down delay, and (b) larger as the capacitor is charged through the enhancement transistor so as to reduce the current taken away from the charging current. One circuit that accomplishes this goal is the Super Buffer FET Logic (SBFL) shown in Figure 2.7. The input stage is a DCFL inverter and the output stage consists of two enhancement transistors controlled by the input and its complement so that as one input goes high the other goes low, and the resistance of the corresponding transistor is increased or decreased, as the load capacitor is being charged or discharged.

Referring to Figure 2.7, when the input is low, V_x will be close to V_{DD}. At the same time, transistor T_4 will be cut off. These two efforts reinforce each other so that the output voltage V_{out} becomes high and is clamped by the gate of next stage. When the input is high, V_x is low, T_3 is cut off while T_4 becomes highly conductive, and the output voltage V_{out} drops to a very low value. These improve the low noise margin and show the strong drive ability. However, dynamically the operation is slightly more complicated. The low to high transition on the output follows the inverter output. After

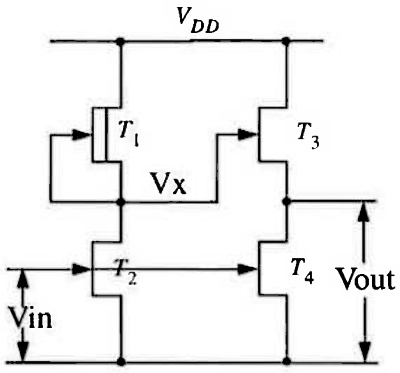


Figure 2.7 Schematic of SBFL inverter.

the inverter transits from low to high (input is low), T_4 is already cut off, and load capacitance will receive the full current of T_3. However, on the high to low transition,

T_4 begins to discharge the load before the inverter has cut off T_3. Therefore, on this transition, there will be a momentary current spike on V_{DD} and ground during which both T_3 and T_4 are in conduction. As a consequence, circuits that use superbuffers must be designed with ample power bus width so that a momentary voltage drop during high to low transition does not dynamically upset other logic elements. Nevertheless, after the propagation delay of the inverter has passed, the load receives the full current of either T_3 or T_4. The static power of the superbuffer driver is the same as that of the inverter.

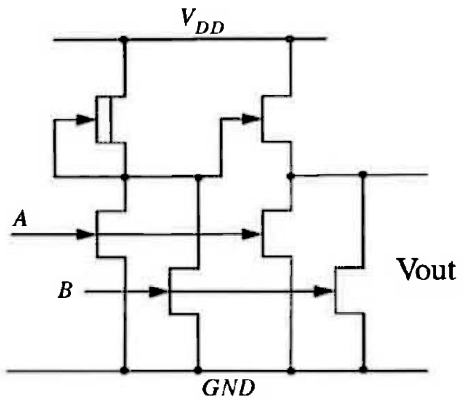


Figure 2.8 A SBFL 2-input *nor* gate.

A SBFL *nor* gate with two inputs is shown in Figure 2.8. Since each input must drive two transistors, the area requirement is considerably larger than that of a SDCFL *nor*. In addition, the two transistors present a capacitive load to the driving stage about twice as large as that presented by a SDCFL gate. For these reasons, the SBFL inverters are ordinarily used only as buffers and not as logic elements.

2.2.5 Two-Phase Dynamic FET Logic (TDFL)

Much of the motivation for using E/D logic circuits with GaAs MESFET has been to obtain low power per gate and high circuit density so that LSI and VLSI circuits can be achieved within a manageable thermal budget and chip size. It is also important to maintain the high-speed operation that is the main reason for using GaAs logic rather than CMOS. Thus, the power-delay product is an important factor which can be reduced by circuit design trade-offs and topologies.

So far several *static* logic circuit approaches, in which direct current flows at all times from the power supply, have been described. Static logic is usually dependent on

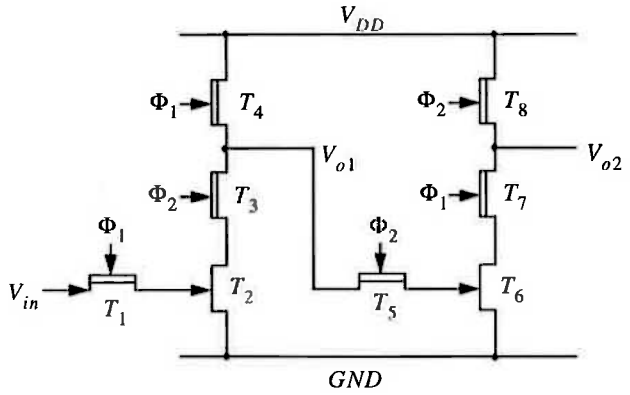


Figure 2.9 Schematic of two TDFL inverters in serial.

width ratios to control logic levels and noise margins, hence these circuits are often referred to as ratioed logic. For a given logic circuit, speed and power are traded off over a wide range as device widths are scaled [25]. Changes in FET threshold voltages also strongly influence both speed and power. Power reduction in static logic also requires circuits and devices which operate efficiently with low logic voltage swings so that power supply voltages can be reduced. Logic swing reduction also implies using FETs with more positive threshold voltage so that the full on-to-off ratio of current can be achieved. Finally the device widths are also reduced so that supply currents can be made small. While all of these factors contribute to lower dc power dissipation, the lower voltage swing reduces noise margin as well, requiring more stringent process control to achieve acceptable circuit yield. For VLSI circuits, high yield is not just desirable but is essential for successful commercial production. Finally, the smaller current levels often lead to increased propagation delay.

While there have been many demonstrations of LSI circuits with some of the above techniques, the higher complexity examples have demanded very strict control of process parameters and uniformity of device parameters. Hence there is an interest in alternative circuit techniques that offer power reduction in combination with an acceptable noise margin.

In efforts to reduce the above problem, a dynamic GaAs logic family called two-phase dynamic FET logic (TDFL) was developed [28].

Two-phase dynamic FET logic (TDFL) is a ratio-less dynamic logic family which uses two nonoverlapping clocks and dissipates power only during clock level transitions. Figure 2.9 shows the schematic diagram of two TDFL inverters in series.

When Φ_1 is high the circuit is in the precharge phase. Transistors T_1 and T_4 conduct, the output (V_{o1}) is charged to high and the gate of T_2 is charged to high if the input is high or discharged to low if the input is low. When Φ_1 is low, Φ_2 is high, indicating the start of the evaluate phase. At this point T_3 conducts, if the gate of T_2 is charged high during Φ_1, then T_2 conducts and the output is discharged to ground by T_2 and T_3. Otherwise T_2 remains off and the output remains high at about 0.5V. This is mainly because it is clamped by T_6 of the next inverter stage.

Since TDFL uses *dfet*s as pass transistors, referring to Figure 2.9 the nonoverlapping clock voltage level is determined as follows [29]:

$$\text{when } T_1 \text{ is on: } V_{td} \le V_{gs} \le V_{Schottkydiode} \Rightarrow V_{td} \le V_{clk}(on) - V_{in} \le 0.8; \quad \text{(Eq2.1)}$$

$$\text{when } T_1 \text{ is off: } V_{gs} < V_{td} \Rightarrow V_{clk}(off) - V_{in} < V_{td}; \quad \text{(Eq2.2)}$$

where

$$0 \le V_{in} \le 0.5. \quad \text{(Eq2.3)}$$

The VITESSE HGaAsII process is going to be used, whose *dfet* has a typical threshold voltage V_{td} = *-0.88 V*, therefore, the region of clock voltage levels are:

$$-0.38V \le V_{clk}(on) \le 0.8V, \quad \text{(Eq2.4)}$$

$$V_{clk}(off) < -0.88V. \quad \text{(Eq2.5)}$$

Shift registers are very easy to construct and occupy little area since they are just cascades of TDFL inverters as shown in Figure 2.9. Figure 2.10(a) and (b) show the schematics of TDFL *nand* and *nor* gates. *and-or-inverter* (*AOI*) gates are also possible. Unlike DCFL, *nand* gates are reliable in TDFL because there is no static current flow in the input transistors (T_3, T_4 and T_5 in Figure 2.10(a)). Proper *nand* gate operation is difficult to achieve in DCFL, because when the inputs are both high so that static current flows through the input transistors, V_{DS} of the lower of the two input transistors causes a significant difference between the V_{GS} of the two input transistors. The V_{GS} of the upper transistor is significantly less than the V_{GS} of the lower. With a TDFL *nand* gate, however, this is not a concern. All that is required for proper operation is the dynamic discharge of the output load capacitance through the series resistance of the input FETs.

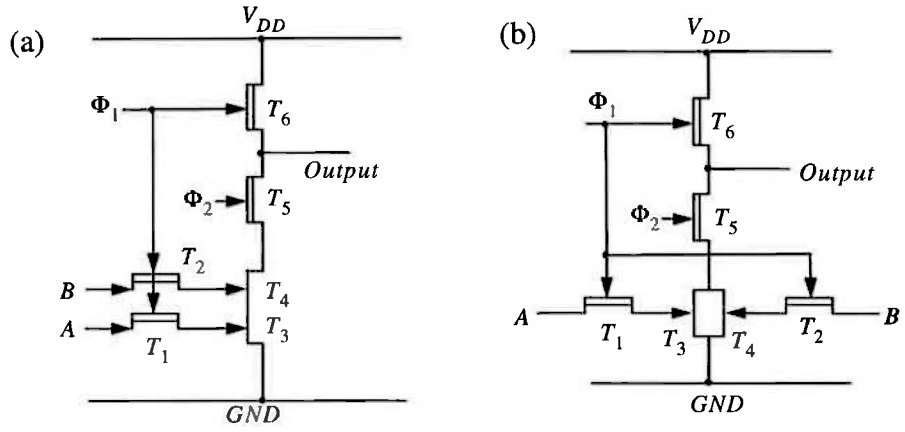


Figure 2.10 Schematic of 2-input TDFL (a) NAND gate; (b) NOR gate

2.3 Design Considerations For Logic Families

2.3.1 Definition of Design Parameters

In the following section the gates are evaluated by simulation using HSPICE (95). Measurements are done on the middle gate in a chain of three identical gates (Figure 2.11), in this way the input and output of the gate under observation are realistic. The evaluations are done in terms of noise margin, propagation delay and power dissipation. There are various definitions for these parameters. The definitions used in this work are given below:

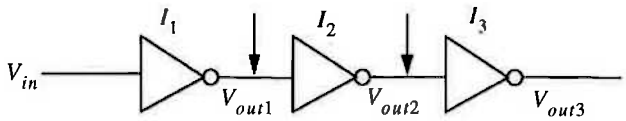


Figure 2.11 Circuit used for measurements.

1. Noise Margin

Noise margin is a parameter closely related to the input-output voltage characteristics. This parameter allows us to determine the allowable noise voltage on the input of a gate so that the output will not be affected. In this section only dc noise margin is considered, because this definition is efficient for the dc design of a logic circuit. Noise margin can be defined in several ways working from the transfer characteristic of the logic circuit [30][31][32]. The definition most commonly used to specify noise margin

(or noise immunity) is in terms of two parameters —the low noise margin, NM_L, and the high noise margin, NM_H. With reference to Figure 2.13(a), NM_L is defined as the difference in magnitude between the maximum low output voltage of the driving gate and the highest input low voltage recognized by the driving gate. Thus

$$NM_L = |V_{IL} - V_{OL}| \qquad \text{(Eq2.6)}$$

The value of NM_H is the difference in magnitude between the minimum high output voltage of the driving gate and lowest input high voltage recognized by the receiving gate. Thus

$$NM_H = |V_{OH} - V_{IH}| \qquad \text{(Eq2.7)}$$

These definitions are illustrated in Figure 2.12(b).

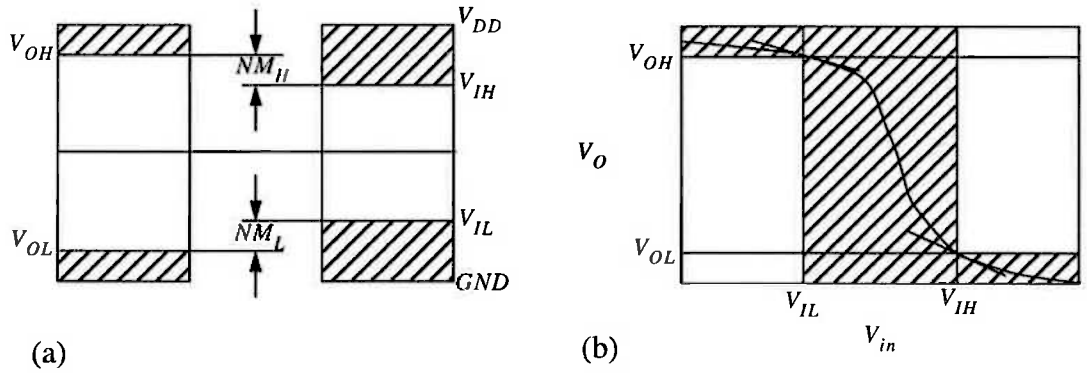


Figure 2.12 (a) Definition of high level and low level noise margin; (b) Transfer curve of a logic gate.

2. Propagation Delay

The propagation delay t_d is defined as the average of rise time t_r and fall time t_f e.g. $t_d = (t_r + t_f)/2$, where t_r and t_f are shown graphically in Figure 2.13.

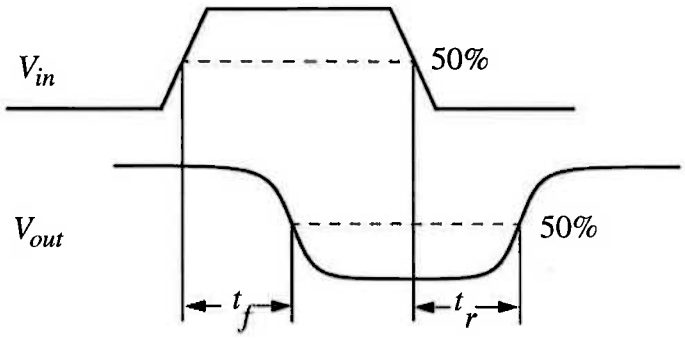


Figure 2.13 Delay time calculation.

3. Power Dissipation

The power dissipation consists of static and dynamic components. For high speed circuits, the dynamic component of the power dissipation is significant and must be included in the calculations. The HSPICE *MEASURE* statement which calculates the area under the output variable divided by the periods of interest is used to measure the average of the instantaneous power dissipation. This includes both the static and dynamic components [33].

4. Power Supply

From simulation the four logic families described previously can work with a supply voltage form 1.2 V to 2.5 V. However, since the mean threshold voltages for the VITESSE HGaAsII process are quoted at power supply voltage of 2V+10%, a 2 V supply is chosen.

5. Temperature

The commercial temperature range is from 0°C to 70°C (case temperature) so for the absolute slowest case, circuits use the slow parameters at 0°C, and the high-current (fast) case should be simulated using the fast parameters at 85°C (to allow for the temperature difference between device junctions and the outside world). Industrial temperature range is -40°C to 85°C while military temperature range is -55°C to 125°C.

All the transistors sizing was chosen at temperature of 75°C, then simulated at temperature range from 0°C to 125°C.

6. Backgating Voltage

Backgating is a widely reported effect [34] in which the drain current of a MESFET is reduced by the presence of other nearby neighbouring FETs which happen to be biased negatively with respect to the source of the first device. The backgating voltage is set to 0.6 V above the most negative voltage.

7. Process Spread

The threshold voltage of a set of *identical* MESFETs is not constant over a wafer or from wafer to wafer. The variations have several causes including: short and narrow channel effects, variations in channel implant dose, activation efficiency, built-in volt-

age variations, and changes in the effective acceptor concentration within a substrate and over all substrates used for circuit fabrication. These variations produce a distribution of V_T across the process (global spread) and across a die (local spread). An increase in threshold voltage due to process spread results in a *slow* MESFET, while a decrease in threshold voltage gives a *fast* MESFET. Although in theory the depletion mode and enhancement mode process spreads can have opposite polarities, this possibility has been substantially eliminated by the use of the *additive implant* manufacturing technique. Consequently *slow-slow* (*ss*) and *fast-fast* (*ff*) are the only two deviations that need be considered, *slow-fast* and *fast-slow* situations being rare [27].

The magnitude of the process spread is characterized by a number, indicating how many standard deviations from the mean the threshold voltages lie. For H-GaAs II process, LSI circuits (greater than 1000 devices) typically require insensitivity to $\pm 2\sigma$ in V_T variations between adjacent devices and across a die (where σ is one standard derivation)[22]. Therefore, the circuits are designed using the "*typical*" set of models, simulated to demonstrate functionality, then resimulated using "2σ *fast*" (*ff2*) and "2σ *slow*" (*ss2*) models to ensure design robustness over process variation.

2.3.2 DCFL Optimization

The design goal of optimization is a trade-off between speed, noise margin, power dissipation and load driving ability. The design priority in this research emphasizes the speed with acceptable noise margin. Since the only parameters which are not fixed by the foundry are the sizes of transistors, therefore the optimization is done by changing the pull-up/pull-down ratio. From Curtice Model [35]

$$I_{ds} = \beta \cdot \frac{W}{L} \cdot (V_{gs} - V_t)^2 \cdot (1 + \lambda \cdot V_{ds}) \cdot \tanh(\alpha \cdot V_{ds}) \qquad \text{(Eq2.8)}$$

where

β—transconductance parameter of the device

I_{ds}—the drain-to-source current of the device

W—the width of the device

L—the length of the device

V_{gs}—gate-to-source voltage

λ—channel-length modulation parameter

V_{ds}—drain-to-source voltage

α—saturation voltage parameter

When the input is high, there is a current flowing from V_{DD} to GND through the *dfet* and the *efet* which can be rewritten as

$$I_{dsd} = \beta_d \cdot \frac{W_d}{L_d} \cdot (V_{td})^2 \cdot [1 + \lambda \cdot (V_{DD} - V_o)] \cdot \tanh[\alpha_d \cdot (V_{DD} - V_o)], \quad \text{(Eq2.9)}$$

and $$I_{dse} = \beta_d \cdot \frac{W_e}{L_e} \cdot (V_{in} - V_{te})^2 \cdot (1 + \lambda_e \cdot V_o) \cdot \tanh(\alpha_e \cdot V_o), \quad \text{(Eq2.10)}$$

respectively. Note that the subscript "*d*" denotes parameters associated with *dfet* and the subscript "*e*" denotes parameters associated with *efet*.

By equating the two currents, the following expression can be obtained:

$$\frac{W_d/L_d}{W_e/L_e} = \frac{\beta_e \cdot (V_{in} - V_{te})^2 \cdot (1 + \lambda \cdot V_o) \cdot \tanh(\alpha_e \cdot V_o)}{\beta_d \cdot (V_{td})^2 \cdot [1 + \lambda_d \cdot (V_{DD} - V_o)] \cdot \tanh[\alpha_d \cdot (V_{DD} - V_o)]}. \quad \text{(Eq2.11)}$$

Since only when input is high, the *efet* conducts and $I_{dse} = I_{dsd}$. By substituting the appropriate values for the H-GaAs II process (Table 2.2) with V_{DD}= 2 V, V_{in}= 0.65 V (a normal high input voltage) and V_O= 0.1 V (a normal low output voltage), the ratio turns out to be 0.17. This provides a good starting point for using HSPICE to do the optimization process. A DCFL gate with satisfactory speed and noise margin at different process spread was found which has the transistor dimensions $L_d = 2\mu m$, $W_d = 2\mu m$, $L_e = 1.2\mu m$, and $W_d = 8\mu m$. Table 2.3shows the performance of this optimal DCFL gate. The dc characteristics and transient of a DCFL inverter are shown in Figure 2.14(a) and (b), respectively.The power plot is for instantaneous power, so are the power plots shown in Figure 2.16 and Figure 2.18.

Table 2.2 Parameter value used in the MESFET model for Vitesse H-GaAs-II process

	$\beta\,(A/V^2)$	α	$\lambda\,(V^{-1})$	$V_t\,(V)$
dfet	2.34×10^{-4}	3.51	0.083	–0.798
efet	3.02×10^{-4}	6.53	0.072	0.227

Table 2.3 summarizes the performance of the optimal DCFL inverter. It can be seen that with increasing fan-out the performance of DCFL is degraded by increasing the propagation delay t_d and decreasing the high noise margin NM_H. This is due to the reduced ability of the D-MESFET to drive the E-MESFETs of the fan-out load. The fan-in capability of DCFL is restricted by the drain-to-source leakage current of the E-MESFET which, when multiplied by the fan-in, reduces NM_H.

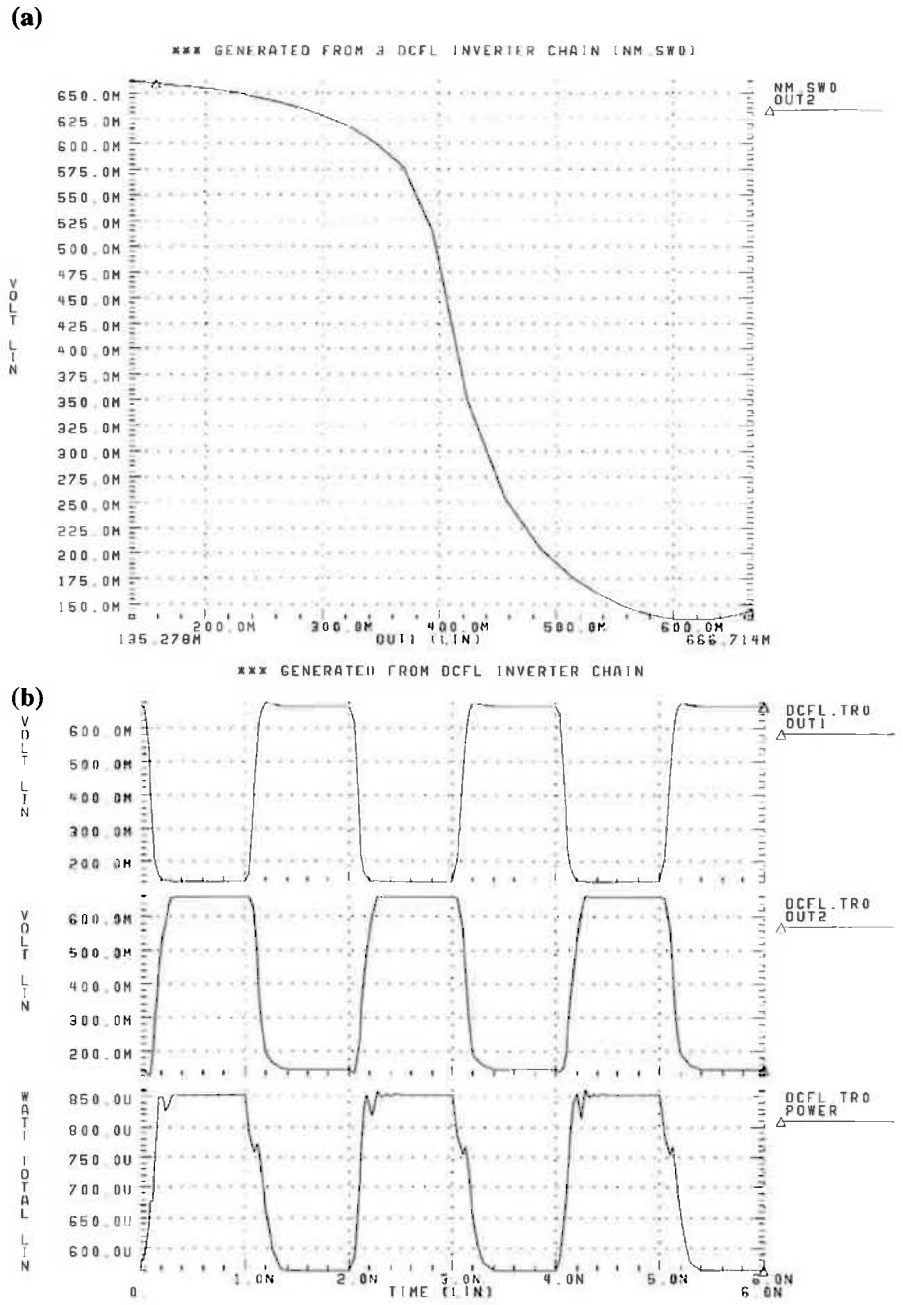


Figure 2.14 (a)The dc characteristics; and (b) the transient of a DCFL inverter.

Table 2.3 Performance of an optimal DCFL inverter

	fan-out	t_r (ps)	t_f (ps)	t_d (ps)	NM_L (mV)	NM_H (V)	P_o (μW)
inv	1	70	45	57.5	160	100	235
	3	150	70	110	178	50	260
nor2	1	80	40	60	160	90	239
	3	170	72	121	173	48	265
nor3	1	90	35	62.5	160	80	240
	3	190	75	132.5	170	45	270

2.3.3 SDCFL Optimization

The optimization of SDCFL is first started by keeping the DCFL stage unchanged while changing the sizes of source follower stage. It was found that the noise margin is improved but the delay and the driving ability is not good unless larger transistors are used in the source follower stage. By changing the sizes of the DCFL stage too, a SDCFL inverter is found which is shown in Figure 2.15. The dc characteristics and the transient of a SDCFL inverter are shown in Figure 2.16. The performance of the optimal SDCFL inverter is summarized in Table 2.4.

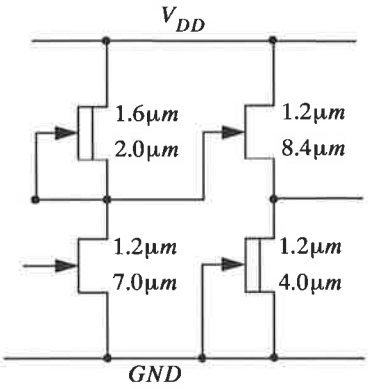


Figure 2.15 Optimized SDCFL inverter

Table 2.4 Performance of a optimal SDCFL inverter

	fan-out	t_r (ps)	t_f (ps)	t_d (ps)	NM_L (mV)	NM_H (mV)	P_o (μW)
inv	1	78	50	64	230	110	710
	3	89	67	78	245	70	741
nor2	1	110	61	85.5	160	200	712
	3	122	78	100	180	170	735
nor3	1	133	61	97	130	260	714
	3	144	83	113.5	120	240	721

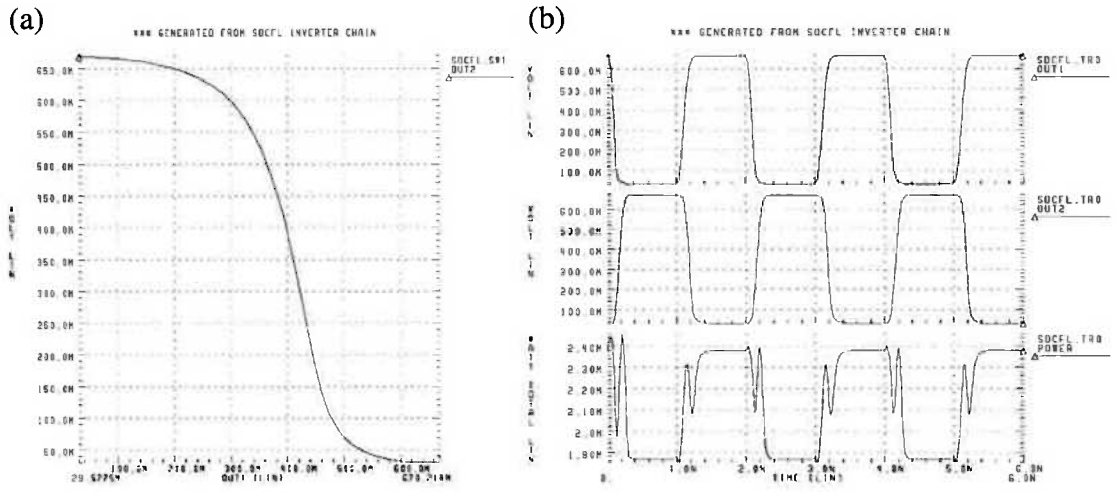


Figure 2.16 (a)The dc characteristics; and (b) the transient of a SDCFL inverter.

2.3.4 SBFL Optimization

Optimizing the SBFL gate sizing is similar to SDCFL, however the devices in the output stage are chosen to have the same sizes. This will improve the driving ability, the penalty is larger area and more power dissipation. Therefore, SBFL gates are only used as drivers. The sizing of the SBFL gate is shown in Figure 2.17

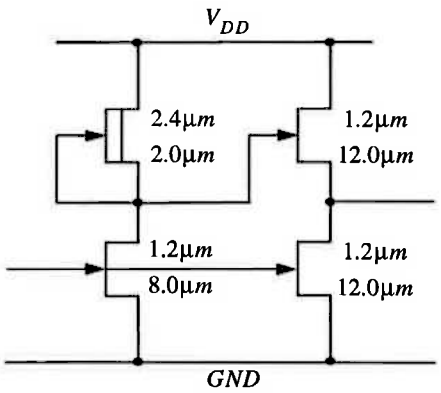


Figure 2.17 Optimized SBFL inverter.

The dc characteristics and the transient of a SBFL inverter are shown in Figure 2.18(a) and (b), respectively. The peak in Figure 2.18(b) panel 3 indicates that the SBFL inverter has a momentary current spike on V_{DD} and ground during the high to low transition.

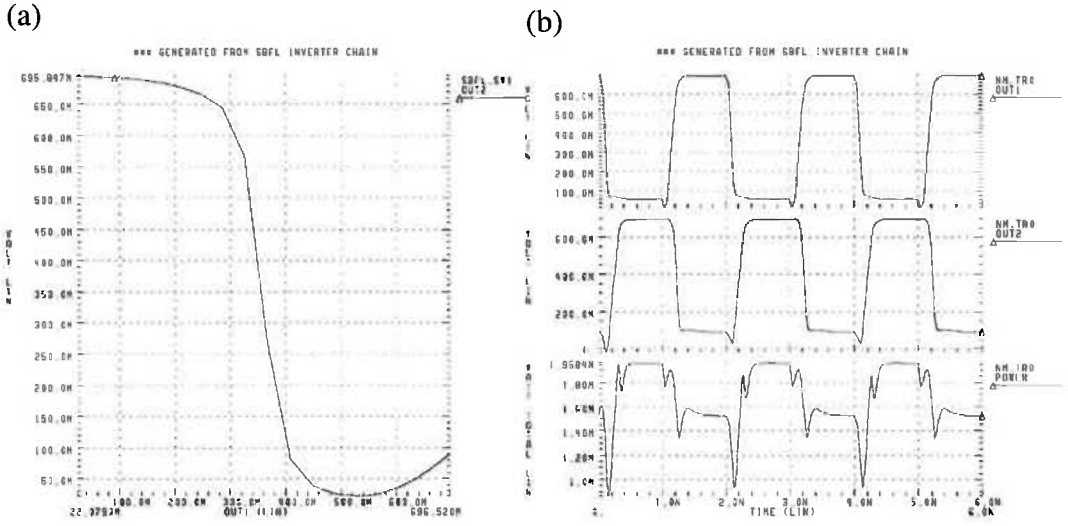


Figure 2.18 (a)The dc characteristics; and (b) the transient of a SBFL inverter.

2.3.5 Performance Comparison

The performance of the optimized DCFL, SDCFL and SBFL inverters are shown in Figure 2.5. Their performance are compared on the basis of speed, noise margin, area and power dissipation. From this it can be seen that the DCFL gate, which is fast and occupies little area, can be used extensively for local logic blocks. SDCFL shows that it has strong fan-out and capacitive driving ability, however it has higher power consumption. SDCFL can be used in critical paths, while SBFL can be used for buffering global signals (such as clocks) due to high power dissipation and noise injection to the power buses during high to low transition. As for SDCFL, the optimized SBFL sizing is minimum. The widths of transistors can be adjusted in proportion to the load. By mixing DCFL, SDCFL and SBFL, high speed and low power operation can be simultaneously achieved.

Table 2.5 The performance of the optimized DCFL, SDCFL and SBFL inverters.

	DCFL	SDCFL	SBFL
propagation delay (*ps*)	57.5	67	92
noise margin ($NM_L/NM_H mV$)	160/100	220/120	220/210
power dissipation (μW)	235	708	482
area (μm^2)	339.2	593.6	784
maximum fan-in	3	5	—
delay/fan-out (*ps*)	26	5	16
delay/20fF load (*ps*)	25	10	12

2.3.6 TDFL Optimization

Unlike GaAs static logic families, TDFL is ratio-less logic which means the transistors can be chosen as small as possible. By simulation, the TDFL inverter sizing as shown in Figure 2.20 were chosen, which satisfies the negative backgating voltage requirement and process spread consideration at the cost of relative large transistors. The *efet* size can be made a little bit larger if the output capacitance is larger. Figure 2.20 shows the simulated operation of TDFL inverter with typical processing parameters and V_{ss}=-1.5 V(to generate -1.2 V clocks) at 1GHz frequency.

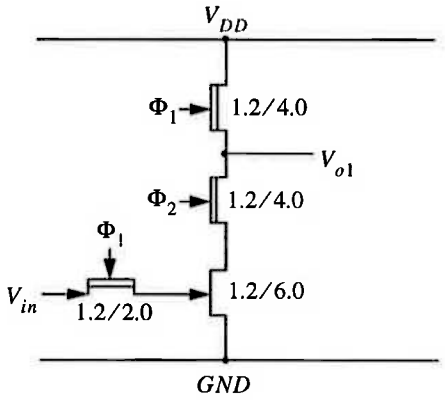


Figure 2.19 The sizing of TDFL inverter.

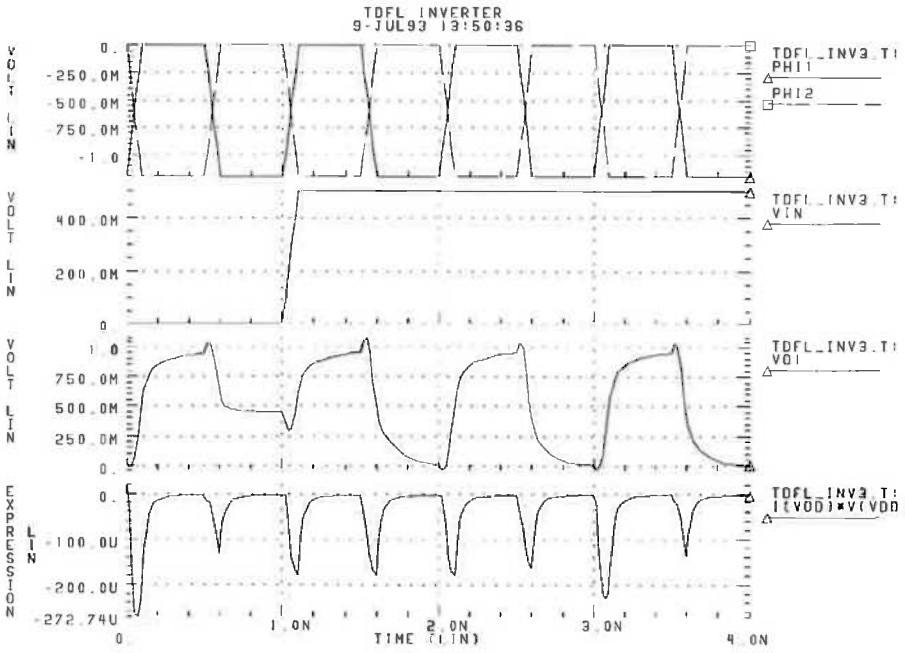


Figure 2.20 Simulated operation of TDFL inverter.

2.4 Interconnections

The high frequency of operation of GaAs circuits requires special attention to the design and analysis of interconnections. A big problem which occurs as a consequence of the fast transitions will be the apparent noise on the signal paths and power supply lines which comes about from reflected waves and crosstalk between lines. The faster the logic family, the more sensitive it will be to transient noise. The effects can be modelled with transmission lines. Any interconnection between two nodes may be thought of as a transmission line if there is an identifiable path for current to flow away from the driving node and back through a return path or reference node. If the interconnecting material is metal and its series resistance is small compared with its characteristic impedance, the losses in the circuit can be neglected for first-order analysis [25]. The properties of the transmission line play a central role in predicting the influence of interconnections. In this section, interconnecting wiring on-chip will be discussed.

2.4.1 Interconnection Analysis

Interconnecting wiring on GaAs IC chips generally consists of metal strips deposited on GaAs(ε_r = 12.9) or on a dielectric layer such as S_iO_2 (ε_r = 3.9), silicon nitride or oxynitride ($4 \leq \varepsilon_r \leq 8$), or polymide ($\varepsilon_r$ = 3 to 3.5), or even in some cases suspended in air above the GaAs wafer surface (air bridges). These metal strips are often thin compared with their widths. Also, the GaAs semi-insulating substrate is quite thick (typically 600 μm) [25] in comparison with the line-to-line spacing. Thus, any backside ground plane has less influence than neighbouring conductors on the surface. These metal interconnect lines also are often short enough that their series resistance is smaller than their characteristic impedance. Therefore, they can sometimes be approximated by ideal transmission lines or segments of transmission line.

Ideally, when working with a well-established and fully characterized IC process, at least the capacitance of interconnect lines of typical widths, spacing, and on all levels will have been measured and will usually be provided by the processing facility. Such measurements require either scale models or interconnect test structures specifically designed for accurate characterization. However, it is also useful to have some approximate models of these interconnections when considering a new process, changing an old process, or interpolating between measured data. Also, line characteristic

impedances and inductances are seldom measured or provided, and they can be calculated if capacitances are known.

The relationships between the characteristic impedance Z_0, the capacitance per unit length C_0 and the inductance per unit length L_0 are [25],

$$L_0 = \frac{Z_0\sqrt{\varepsilon_{eff}}}{c} \qquad \text{(Eq2.12)}$$

and

$$C_0 = \frac{\sqrt{\varepsilon_{eff}}}{Z_0 c} \qquad \text{(Eq2.13)}$$

where ε_{eff} is the effective dielectric constant for a given transmission line. c is the speed of light in free space.

The effective dielectric constant ε_{eff} allows one to calculate electrostatic quantities of interest for a system of conductors in a nonuniform dielectric using a quasi-TEM approximation. More precisely, if the total capacitance of the conductor embedded in its dielectric layers is C_2 and the capacitance of the conductor in free space ($\varepsilon_r = 1$) is C_1, $\varepsilon_{eff} = C_2/C_1$. Integrated circuit wiring can consist of metal strips embedded in slabs of dielectrics, including air.

In order to have an accurate prediction of the parasitic elements in the actual circuit, proper models of transmission line behaviour are required. The general structure of interconnections, however, has prevented the formulation of the characteristics of the lines in a generic form. Assuming the interconnection lines have rectangular cross-section and constant width can help to simplify the analysis. Further simplification can be achieved by categorizing the structure into one of the well defined and accurately modelled structures which have already been studied and characterized. The usual interconnection structures which may occur in GaAs chip are shown in Figure 2.21(a). The most commonly used transmission line structures are listed below.

1. coplanar waveguide (CPW)

The *coplanar waveguide* (CPW) geometry is shown in Figure 2.21(c) which can approximate multiple interconnect lines running in parallel along a wiring channel

which occur frequently in many IC layouts (Figure 2.21(b)). CPW is a surface-oriented structure which assumes that the dielectric and the reference or ground (earth) strips have semi-infinite extent. Even though on the IC the adjacent conductor strips, assumed to be ground, are not semi-infinite, the error is small (Z_0 is 6 percent low when b/c = 0.6), and a correction for this error may be made if desired [36]. Also, the first-order CPW analysis can ignore the influence of the lower ground plane on the substrate back side. The lower ground is significant only if the substrate thickness h < 5b [36].

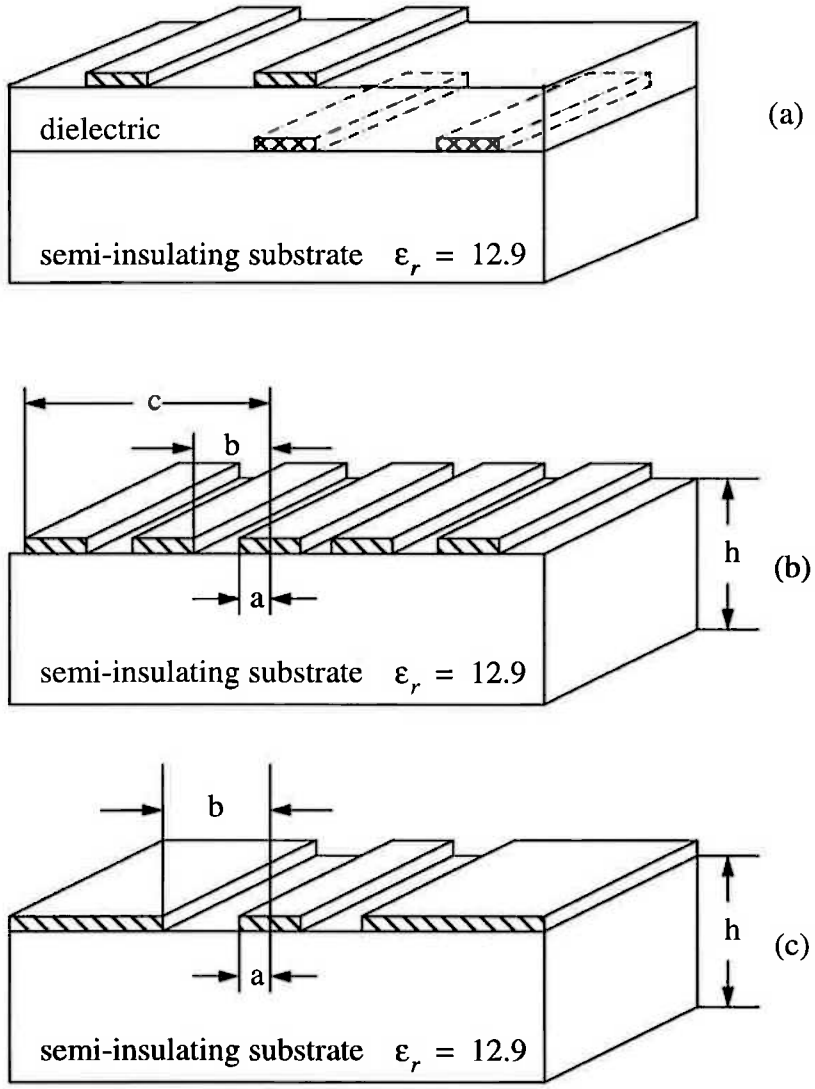


Figure 2.21 (a) Different interconnection structures in a GaAs technology. (b) Section of a typical wiring channel. Interconnect strips are not covering the surface completely and are not semi-infinite in extent. (c) A section of the coplanar waveguide (CPW) transmission line. The usual analytical solution for line impedance or capacitance assumes semi-infinite extension of ground strips and dielectric.

For the ideal CPW of Figure 2.21(b) without a lower ground plane, Ref. [37] provides the follow equations:

$$Z_0 = \frac{30\pi}{\sqrt{\varepsilon_{eff}}} \frac{K(k')}{K(k)} \qquad \text{(Eq2.14)}$$

$$\varepsilon_{eff} = 1 + \frac{\varepsilon_r - 1}{2} \frac{K(k')}{K(k)} \frac{K(k_1)}{K(k'_1)} \qquad \text{(Eq2.15)}$$

In these equations, $K(k)$ is a complete elliptic integral of the first kind and $K'(k) = K(k')$ is the complementary function. k' is defined as

$$k' = \sqrt{1 - k^2}, \qquad \text{(Eq2.16)}$$

where

$$k = \frac{a}{b}, \qquad \text{(Eq2.17)}$$

$$k_1 = \frac{\sinh(\pi a/2h)}{\sinh(\pi b/2h)}. \qquad \text{(Eq2.18)}$$

For the usual case where $h >> b > a$, $\sinh x \approx x$ and $k_1 \approx a/b$. Thus in most instances, Eq.2.15 reduces to

$$\varepsilon_{eff} = 1 + \frac{\varepsilon_r - 1}{2}, \qquad \text{(Eq2.19)}$$

where ε_r is the relative dielectric constant of GaAs.

An approximation for the elliptic function was provided in Ref.[38] and is reproduced below. For convenience, Figure 2.22 provides a plot of $K(k)/K(k')$ for arguments k in a useful range for CPW calculations:

$$\frac{K(k)}{K'(k)} = \begin{cases} \frac{1}{\pi}\ln\left(2\frac{1+\sqrt{k}}{1-\sqrt{k}}\right) & 0 \le k \le 0.707 \\ \left[\frac{1}{\pi}\ln\left(2\frac{1+\sqrt{k'}}{1-\sqrt{k'}}\right)\right]^{-1} & 0.707 \le k \le 1 \end{cases} \qquad \text{(Eq2.20)}$$

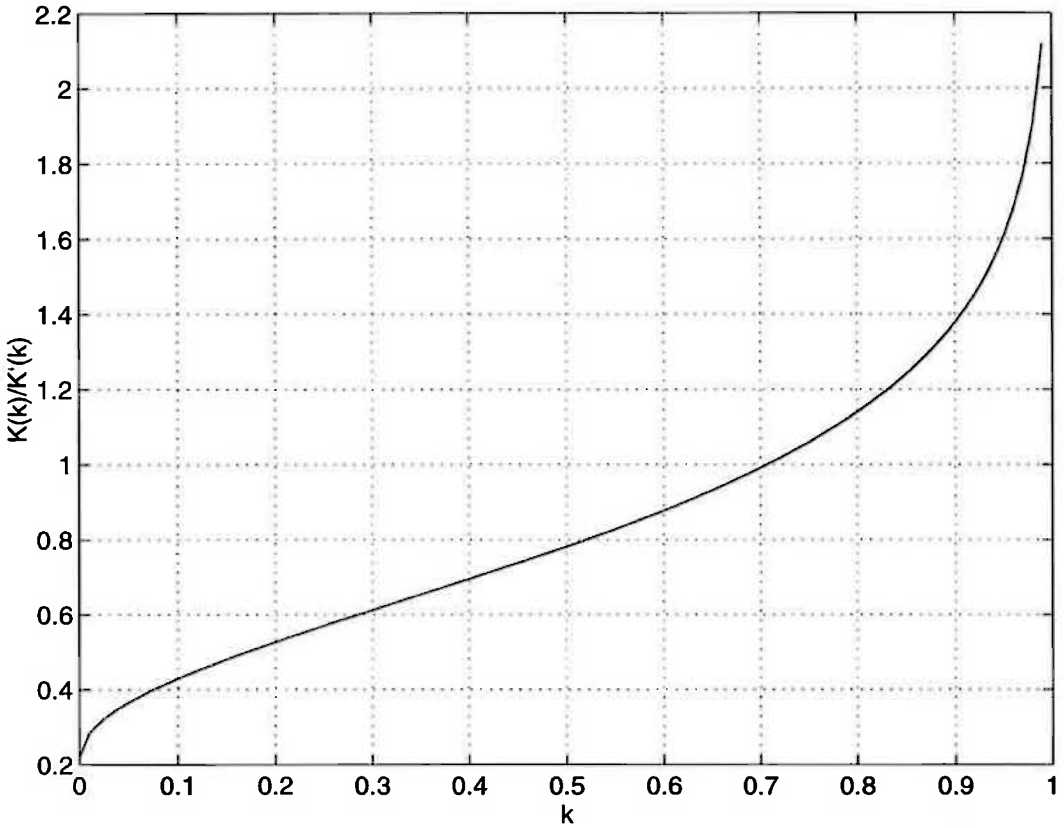


Figure 2.22 Elliptic function K(k)/K'(k) where K(k) is the elliptic integral of the first kind and K'(k) is the complementary function.

2. coplanar striplines (CPS)

Another transmission line geometry which may occur occasionally in digital IC layouts is that of the *coplanar striplines* (CPS) shown in Figure 2.23(a). It differs from CPW in that the two strips are assumed to be completely separated from neighbouring conductors on the surface. In practice, if the spacing between the coplanar strips and the nearest conductors is at least 10b, then the CPS formula from Ref.[37] can be applied:

$$Z_0 = \frac{120\pi}{\sqrt{\varepsilon_{eff}}} \frac{K(k)}{K'(k)} \qquad \text{(Eq2.21)}$$

Equations Eq.2.15 to Eq.2.19 also apply to CPS.

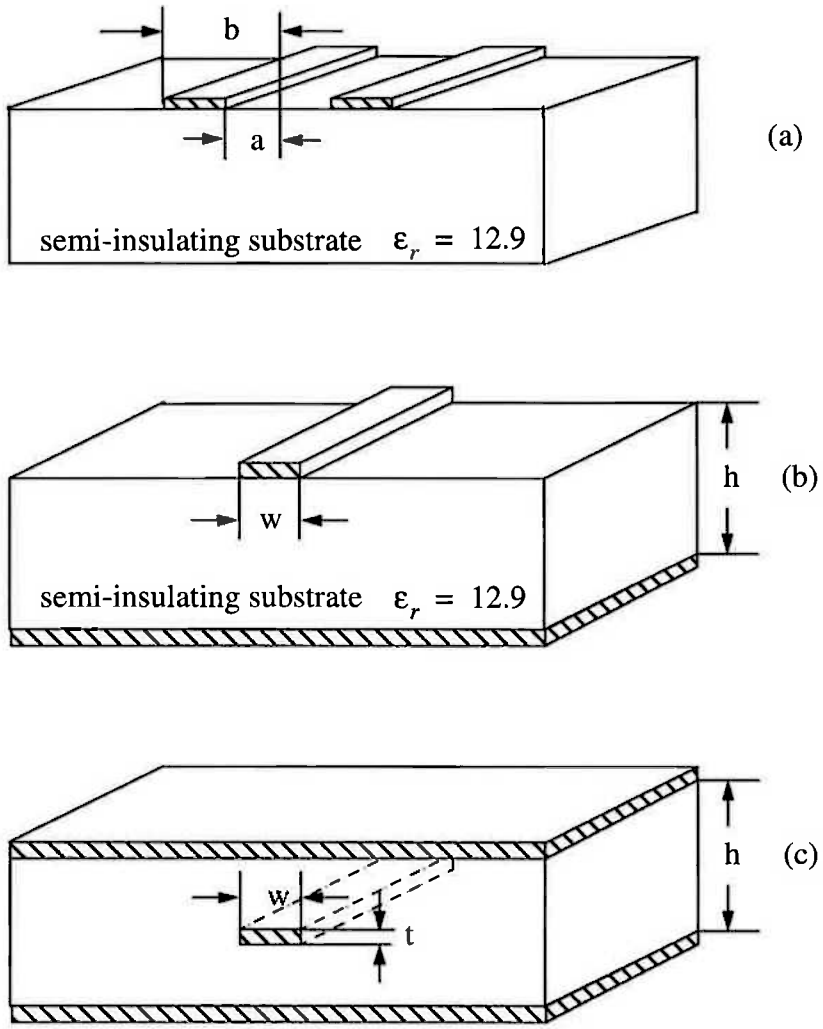


Figure 2.23 (a) Coplanar stripline (CPS). (b) Microstrip line. (c) Stripline.

3. microstrip line

The microstrip line shown in Figure 2.23(b) would apply to digital IC chips, if the substrate were conductive. Lightly-doped p-type substrates (or semi-insulating substrates with a buried p-layer) might be used as an alternative to semi-insulating substrates for isolation of devices. On semi-insulating substrates, the distance from the line to neighbouring conductors is generally less than the substrate thickness so the coplanar models are more appropriate. Microstrip occurs more frequently on circuit boards and on the microwave monolithic ICs (MMICs).The impedance and effective dielectric constant for the microstrip have been derived in Ref.[39] and are given below:

$$Z_0 = \frac{60}{\sqrt{\varepsilon_{eff}}} \ln\left(\frac{8h}{w} + \frac{w}{4h}\right) \qquad for \qquad \frac{w}{h} < 10, \tag{Eq2.22}$$

$$\varepsilon_{eff} = \frac{\varepsilon_r + 1}{2} + \frac{\varepsilon_r - 1}{2}\left(1 + \frac{10h}{w}\right)^{-1/2}. \tag{Eq2.23}$$

4. stripline

Finally, the stripline geometry shown in Figure 2.23(c) is a frequent configuration found on or in multilayer circuit boards. The equation for Z_0 is [40]:

$$Z_0 = \frac{30\pi K'(k)}{\sqrt{\varepsilon_r}\, K(k)}, \tag{Eq2.24}$$

$$k = \tanh\left(\frac{\pi w}{2b}\right). \tag{Eq2.25}$$

2.4.2 Power Supply and Ground Considerations

The most critical problem associated with the packaging of high speed chips is delivering clean, constant and equal power supplies and ground to every device on a chip. This task is the hardest to do correctly and, if done incorrectly, is difficult to detect and fix. Since the metal thickness is constrained by the processing technology, it may be necessary to increase line width in order to reduce voltage variations. At the same time, the area consumed by power buses has to be minimized so that the density of circuits and signal interconnections can be as high as possible. There are three key factors associated with power supply issues: resistance, electromigration and inductance.

1. Resistance: *IR* drop

Variations in power supply and ground will happen because of the finite resistivity of metal interconnect lines. This happens even though the resistivity of most metals is quite small, because the on-chip interconnect lines are very thin (*0.3*μ*m* to 10μ*m* typically).The resistance of a conductor is:

$$R = \rho\frac{l}{Wt}, \tag{Eq2.26}$$

where $\rho(\Omega \cdot cm)$ is the resistivity of the conductor, and l, W and t are the length, width and thickness of the conductor, respectively. Therefore, the voltage variation can be calculated as:

$$\Delta V = IR = \rho \frac{Il}{Wt}. \qquad \text{(Eq2.27)}$$

Eq.2.27 can be used to estimate the voltage drop on a conductor when the power line is from point to point. However, usually a power bus line distributes current to a number of gates rather than only one gate. If the current is uniformly distributed along the length of the line (l), the current at position x can be calculated as:

$$I(x) = I_T\left(1 - \frac{x}{l}\right). \qquad \text{(Eq2.28)}$$

The voltage drop at the far end of the line will be:

$$\Delta V = \frac{\rho I_T}{A}\int_0^l \left(1 - \frac{x}{l}\right) dx = \frac{\rho I_T l}{2A}, \qquad \text{(Eq2.29)}$$

where I_T is the total current, and $A = Wt$. It can be seen that the voltage drop is half of that if all the load were concentrated at the far end of the line.

2. Electromigration: current density limitations

The maximum current flowing in one direction through metal wires on the die is limited due to a phenomenon called *electromigration.*When current densities are high, metal atoms will migrate in the presence of electron flow because of the force on metal ions due to the electric field and the exchange of momentum caused by the motion of electrons (electron "wind"). This phenomenon is called *electromigration.* The rate of diffusion is increased in thin-film structures by grain boundaries. When the current density is high, the grain boundaries will form larger vacancies in the metal crystal structure which tend to merge to form larger voids. The current density is further increased in the region of the voids, causing an increasing rate of migration. Eventually, cracks are formed which cause open-circuit of the interconnect line. Therefore, electromigration presents a reliability problem.

The maximum current limits for VITESSE HGaAsII metal lines (I_{cl} per micron) are given in Table 2.6[27]. These are the absolute maximum currents allowed as determined by worst case simulation.

The maximum current in a metal line is calculated as:

$$I_{MAX} = I_{cl}(W - \Delta W) \qquad \text{(Eq2.30)}$$

where W is the drawn line width and ΔW is the process control factor.

Table 2.6 Layer sheet resistance and maximum current (T=85°C)

Layer	$R_s\,(\Omega/\square)$	Max. Current Limit, $I_{Cl}\,(mA/\mu m)$			$\Delta W\,(\mu m)$
		DC	AC	Peak	
Gate Metal	0.5 - 1.5	5.0	5.0	25.0	0.4
Ohmic Implant	190 - 230	1.0	1.0	2.0	0.0
Ohmic Metal	< 10.0	0.3	0.3	0.6	0.0
Metal 1	< 0.070	1.0	1.0	5.0	0.2
Metal 2	< 0.035	2.0	2.0	10.0	0.0
Metal 3	< 0.025	2.8	2.8	14.0	0.0
Metal 4	< 0.025	2.8	2.8	14.0	0.0

Power and ground wires are especially prone to the electromigration limit because current generally flows in one direction through them. If the total current flow in a power bus is greater than the electromigration limit I_{MAX}, circuit performance degradation can result after period of time. Therefore, the number of internal power/ground pads has to be carefully calcutlated.

3. Inductance: *LdI/dt* voltage variation

The third factor is the self-inductance associated with the transmission line nature of the wiring, which is troublesome for both on-chip and off-chip power supply and ground interconnections. If the current on the line changes at the rate *dI/dt*, the inductance L will cause a transient change in voltage according to the well known equation:

$$V = L\frac{dI}{dt}. \qquad \text{(Eq2.31)}$$

The inductance of a transmission line can be calculated by the models describe in Sec. **2.4.1.** Since L is independent of ε_{eff}, it is more convenient to use $\varepsilon_{eff} = \varepsilon_r = 1$ when calculating L, without loss of accuracy.

2.4.3 Propagation Delay

In fact in all cases concerning digital signal propagation, *RC* effects dominate transmission line effects at the frequencies of interest for GaAs integrated circuits. This is due to the low metallization resistance and short line lengths used for on-chip routing.

Usually if the propagation delay is less than the signal rise time, the signal path can be modelled by lumped *RC* elements. The propagation delay can be calculated as below:

$$t_{pd} = l/v_r, \qquad \text{(Eq2.32)}$$

where l is the signal path length, v_r is the speed of light in material, and $v_r = c/\varepsilon_r$. c is the velocity of light in free space. ε_r is the relative dielectric constant. For the Vitesse process, ε_r is approximately 4.5.

Since usually the signal rise time is around 100*ps*, for path length < 5 *mm*, Eq.2.32 gives $t_{pd} < 75$ *ps*. Therefore, propagation delays are indeed dominated by *RC* effects and not by speed of light effects for normal signal lengths. *RC* propagation delays can be obtained by HSPICE simulations.

2.4.4 Crosstalk

When two unshielded signal lines are placed very close, the electric and magnetic fields of the lines overlap sufficiently so that a wave propagating in one line will induce a wave in the adjacent line. This coupling is called *crosstalk*.

Whenever signal lines carrying gigahertz frequencies must be routed in parallel for even a few centimetres, crosstalk must be considered. This effect is proportional to the length of interconnection lines and to signal speeds as well as other parameters. Crosstalk is inversely proportional to the distance between two traces.

Crosstalk can be simulated by HSPICE using transmission models. Figure 2.25 and 2.26 show the HSPICE simulated crosstalk between two signal lines with length of 5 *mm* and 1 *cm* respectively. The model used in the crosstalk simulation is illustrated in Figure 2.24. The detailed HSPICE files are listed in Appendix C.

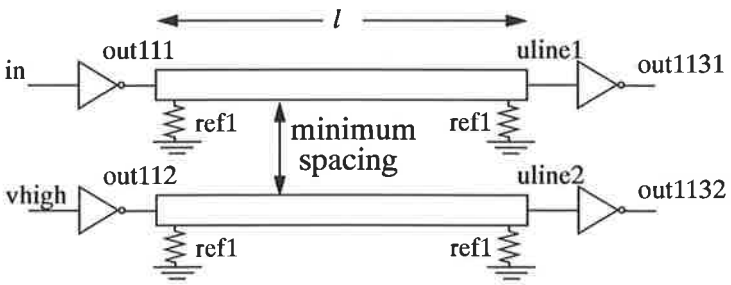


Figure 2.24 The model used in the crosstalk simulation.

It can be seen that for short interconnections (Figure 2.25), the coupled voltage on the quiet line, induced by adjacent active line, is below 60 *mV*. In this case, crosstalk is generally not a problem. However, for long interconnections (Figure 2.26) the coupled voltage is about 115 *mV*. This will cause problem for static GaAs logic families mentioned previously, since their high voltages are about 700 *mV*. Therefore, long parallel bus structures on chips can present crosstalk problems and should be carefully modelled.

2.5 Summary

In this chapter general information about GaAs was introduced, and then four GaAs MESFET logic families were discussed. In conclusion, TDFL (two-phase Dynamic FET Logic) can be used in the mixed dynamic/static approach; DCFL (direct Coupled FET Logic) can be used as the most appropriate one for static logic implementation; SDCFL (Source Follower DCFL) can be used as a buffer to driver high fan-out loads; SBFL (Super Buffer FET Logic) can be used to drive very high fan-out loads such as clock signals. Interconnection issues and crosstalk were also discussed in this chapter which will be the theoretical foundation of the new layout style described in Chapter 5.

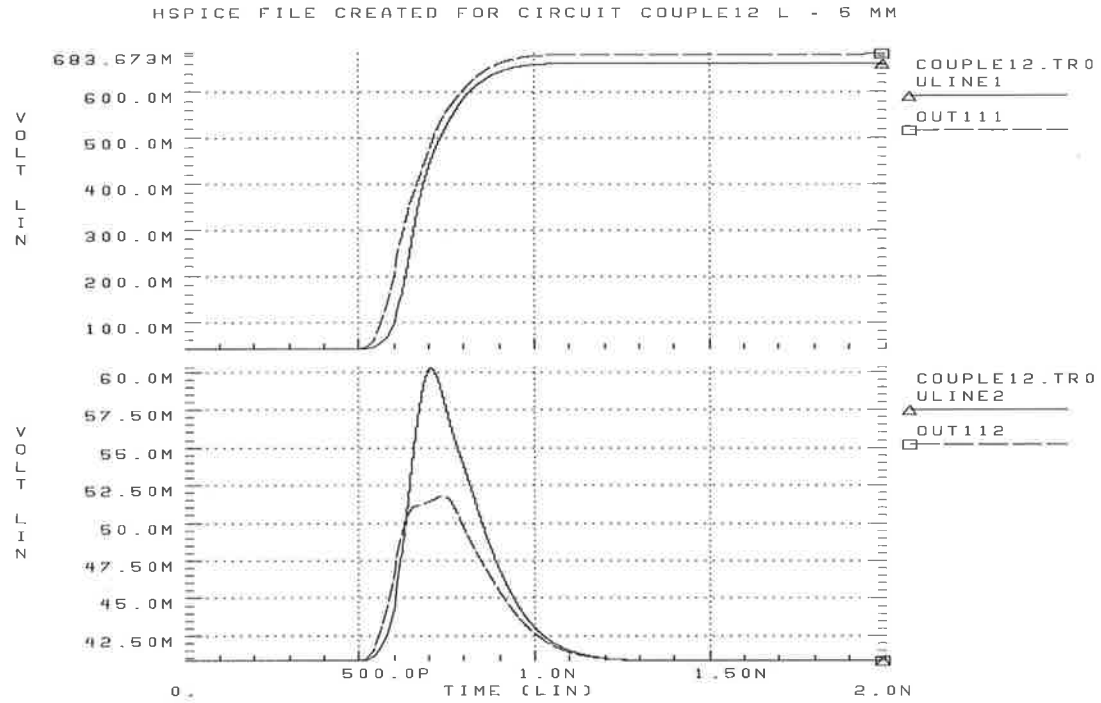


Figure 2.25 The simulated crosstalk between two 5 *mm* signal lines.

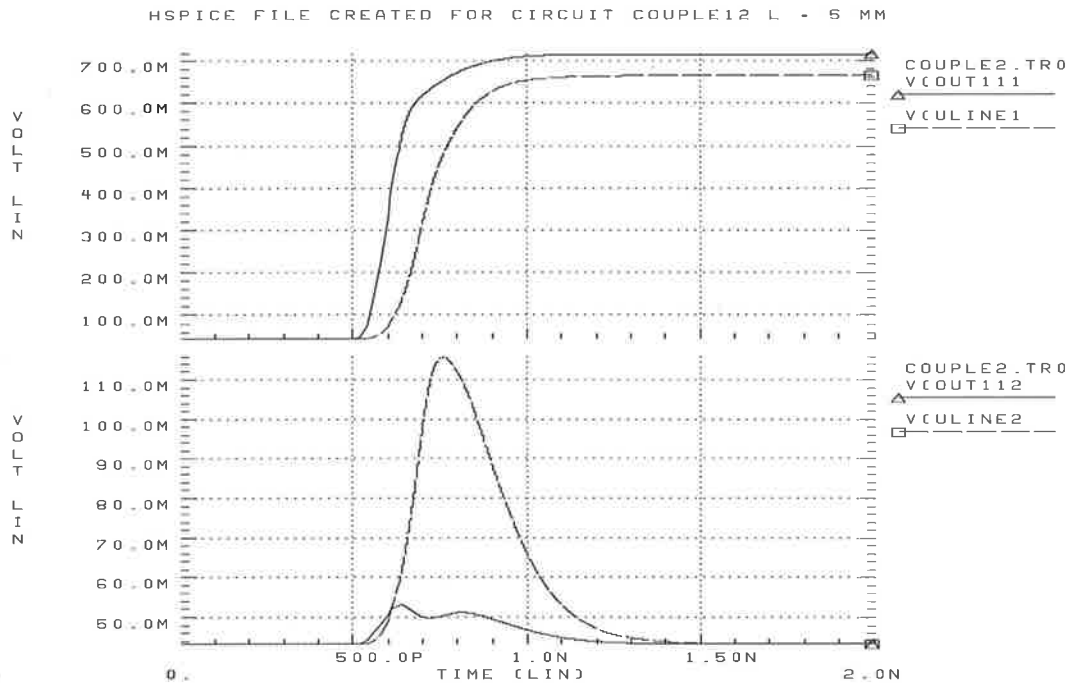


Figure 2.26 The simulated crosstalk between two 1 *cm* signal lines.

CHAPTER 3

An 8-bit Serial Dynamic/Static Divider

High-speed and low power digital GaAs chips are required for the GaAs Core Processor and many digital signal processing and image processing applications. However, the relatively high static power dissipation and relatively low layout density as compared to CMOS tend to limit the utilization of GaAs VLSI circuits.

From Chapter 2, it was seen that TDFL family has the characteristics of very low-power and high-density. Since TDFL is compatible with other static logic families described in Chapter 2, a GaAs dynamic/static mixed approach is used to reduce the power and the area of chips while maintaining high speed.

In this chapter, an 8-bit serial divider is described, which serves as a test bench for this mixed approach.

3.1 The Algorithm of The Serial Divider

The implementations of division in the Floating Point Unit's (FPU's) of current microprocessors are based on one of two categories of algorithms [36]. Subtractive methods such as the conventional approach using subtraction and shifting, and the many variations of radix-2 SRT (higher-radix subtractive division), generally use dedicated, parallel hardware. Multiplicative techniques, examplified by the Newton-Raphson iteration and series expansion algorithm, share functionality with the floating point multiplier and use multiplication and addition to develop increasingly accurate approximations to the desired quotient. These different approaches give rise to the distinct area and performance characteristics which are explored in this section in order to choose the better method for implementation using GaAs logic.

3.1.1 Subtractive Division

The subtractive division uses subtraction and shifting in a manner similar to the paper-and-pencil approach that people use. This class of division includes *restoring* and *nonrestoring* algorithms [43].

1. Restoring Division

The simplest divider operates on two unsigned binary numbers, one bit at a time, as illustrated in Figure 3.1. The divident and divisor are $a_{n-1}a_{n-2}\cdots a_0$ and $b_{n-1}b_{n-2}\cdots b_0$ respectively and they are placed in register A and B, respectively. Register P is initially zero. Then the division proceeds as follows:

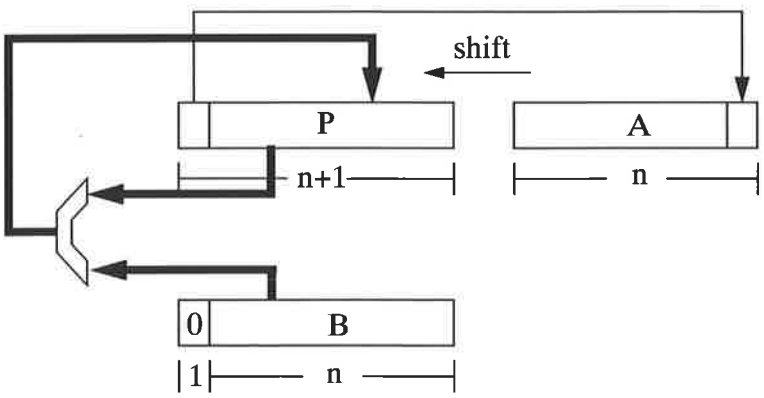


Figure 3.1 Block diagram of simple divider for n-bit unsigned integers.

1. Shift the register pair (P, A) one bit left.

2. Subtract the contents of register B from register P.

3. If the result of step 2 is negative, set the low-order bit of A to 0, otherwise to 1.

4. If the result of step 2 is negative, restore the old value of P by adding the contents of register B back into P.

After repeating this n times, register A will contain the quotient, and register P will contain the remainder. This algorithm is the binary version of the paper-and-pencil method.

Example:

Consider the division of 15/6. The numbers 15 and 6 are represented in binary as 1111 and 0110, respectively.

P	A	
00000	1111	Divide 15 = 1111 by 6 = 0110, B always contains 0110
00001	111	step (1): shift
-00110		step (2): subtract
-00101	**1110**	**step (3): result is negative, set quotient bit to 0**
00001	1110	step (4): restore
00011	110	step (1): shift
-00110		step (2) subtract
-00011	1100	step (3): result is negative, set quotient bit to 0
00011	1100	step (4): restore
00111	100	step (1): shift
-00110		step (2): subtract
00001	1001	step (3): result is positive, set quotient bit to 1
00011	001	step (1): shift
-00110		step (2): subtract
-00011	0010	step (3): result is negative, set quotient bit to 0
00011	0010	step (4): restore. The quotient is 0010 and the reminder is 00011.

This division algorithm is called *restoring*, because if subtraction by b yields a negative result, register P is restored by adding b back in. It is obvious that it may need as many as $2n + 1$ cycles to select all the quotient digits, that is, there are $n + 1$ cycles for the trial subtraction, and there may be additional n cycles for the restoration. How-

ever, these restoration cycles can be easily eliminated by a more powerful subtraction division algorithm: *nonrestoring division*.

2. Nonrestoring Division

The *nonrestoring* algorithm is:

1. Shift the register pair (P, A) one bit left.

2. Subtract the contents of register B ($b_{n-1}b_{n-2}\cdots b_0$) from register P.

If P is negative,

3.1a. Set the low-order bit of A to 0.

3.2a. Shift the register pair (P, A) one bit left.

3.3a. Add the contents of register B to P.

Else,

3.1b. Set the low-order bit of A to 1.

3.2b. Shift the register pair (P, A) one bit left.

3.3b. Subtract the contents of register B from P.

After repeating this n times, the quotient is in A. If P is nonnegative, it is the remainder. Otherwise, it needs be restored (i.e., add b), and then it will be the remainder. Note that the sign of P must be tested before shifting, since the sign bit can be lost when shifting. Register P must be extended to $n + 1$ bits in order to detect the sign of P. Therefore, the implementation of *nonrestoring* division uses the same hardware as *restoring* division (the control is slightly different) but without extra additions.

Take the same example for *restoring*:

P	A	
00000	1111	Divide 15 = 1111 by 6 = 0110, B always contains 00110
00001	111	step (1): shift
+11010		step (2): subtract b (add 2's complement)
11011	1110	step (3.1a): result is negative, set quotient bit to 0

P	A	
10111	110	step (3.2a): shift
+00110		step (3.3a): add *b*
11101	1100	step (3.1a) result is negative, set quotient bit to 0
11011	100	step (3.2a): shift
+00110		step (3.3a): add *b*
00001	1001	step (3.1b): result is nonnegative, set quotient bit to 1
00011	001	step (3.2b): shift
+11010		step (3.3b): subtract *b* (add 2's complement)
11101	0010	step (3.1a): result is negative, set quotient bit to 0
+00110		remainder is negative, so do final restore step
00011	**0010**	The quotient is 0010 and remainder is 00011.

3. Higher-Radix Subtractive Division

Even with nonrestoring division, the speed of quotient formation is seriously limited by the requirement of an inspection and conditional operation in order to form each new bit of the quotient. An *n*-bit quotient requires *n*-serial addition/subtraction operations—significantly slower than multiplication.

A good deal of the early literature on division concerned methods of improving division speed by developing two or more quotient bits per serial addition/subtraction time. These algorithms use multiple simultaneous subtractions or comparisons to situate the new quotient bits. The radix-2 member of this class is also called radix-2 SRT division named by Freiman [44] because it was discovered independently at about the same time by D. Sweeney of IBM, J. E. Robertson of the University of Illinois [45], and T. D. Tocher, then of Imperial College, London [46]. Then J. E. Robertson extended the technique to higher-radix subtractive division [47].

The basis for these algorithms is that multiple trial divisors can be simultaneously compared with the present partial remainder to determine both the new quotient bits as will as the next trial divisor action. In order to form n bits of quotient in an iteration $2^n - 1$ equally spaced divisor partitions must be derived. These partitions together with the two extreme trial divisors, 0 and D, define 2^n possible outcomes. Thus, $2^n - 1$ comparisons are simultaneously made with the present partial remainder. The smallest partition that has a positive comparison determines the next n quotient bit configuration.

The radix-2 SRT algorithm may be expressed by following equations:

$$X'_i = X_i - q_i D \quad \text{(Eq3.33)}$$

$$X_{i+1} = 2X'_i \quad \text{(Eq3.34)}$$

where D is the divisor expressed in unsigned binary ($D = d_0 d_{-1} \ldots$)

X_i is the partial remainder after i iterations ($X = X_2 X_1 X_0 X_{-1} \ldots$)

X_0 is the dividend expressed in 2's complement

q_i is the i'th digit of the quotient ($q_0 q_1 q_2 \ldots q_m$)

X_i' is the unshifted version of X_{i+1} ($X'_i = X_2' X_1' X_0' X_{-1}' \ldots$)

For radix-2 SRT, the resultant quotient is in a redundant signed digit form in which each digit is selected to be one of +1, -1 or 0. The arithmetic value of the quotient can be evaluated using

$$Q_i = \sum_{j=0}^{m} q_j \cdot 2^{-j}. \quad \text{(Eq3.35)}$$

Given a divisor and dividend within the range [1, 2), it is possible to select a quotient digit such that the new partial remainder always falls within the bounds

$$|X_{i+1}| \leq 2D. \quad \text{(Eq3.36)}$$

3.1.2 Multiplicative Division

Algorithms of this class first produce a reciprocal of the divisor $1/b$, and then the result is multiplied by the dividend a. Thus, the main difficulty is the evaluation of a reciprocal. There are two popular techniques of iteration to find the reciprocal [41]. One is the *series expansion*, and the other is the *Newton-Raphson iteration*. For the clarity, these two algorithms are discussed here.

1. Division by Series Expansion

The series expansion is based on the Maclaurin series (a special case of the familiar Taylor series). Let b, the divisor, be equal to $1 + x$ and between [0.5, 1.0]:

$$\frac{1}{b} = \frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - \ldots \quad \text{(Eq3.37)}$$

Since $x = b - 1$, and $0.5 \le b < 1.0$, Eq.3.37 can be factored:

$$\frac{1}{b} = (1-x)\left(1+x^2\right)\left(1+x^4\right)\left(1+x^8\right)\left(1+x^{16}\right)\ldots \quad \text{(Eq3.38)}$$

Since:

$$2 - \left(1 + x^n\right) = 1 - x^n,$$

the two's complement of $1 - x^n$ is $1 + x^n$.

This algorithm was implemented in the IBM 360/91 [49], where division to 32-bit precision was evaluated as follows:

1. $(1 - x)(1 + x^2)(1 + x^4)$ is found from a ROM look-up table.

2. $1 - x^8 = [(1 - x)(1 + x^2)(1 + x^4)(1 + x)]$.

3. $1 + x^8$ is the two's complement of $1 - x^8$.

4. $1 - x^{16}$ is computed by multiplication $(1 - x^8)(1 + x^8)$.

5. $1 + x^{16}$ is the two's complement of $1 - x^{16}$.

6. $1 - x^{32}$ is the product of $(1 - x^{16})(1 + x^{16})$.

7. $1 + x^{32}$ is the two's complement of $1 - x^{32}$.

In the ROM look-up table the first i bits of b are used as an address of the approximate quotient. Since b is bit-normalised ($0.5 \le b < 1$), then $|x| \le 0.5$ and $\left|x^{32}\right| \le 2^{-32}$; i.e., 32-bit precision can be obtained in Step 7.

2. Newton-Raphson Iteration

Newton-Raphson iteration is based on the following procedure to solve the equation $f(x) = 0$, shown in Figure 3.2 [42]:

- draw a graph $y = f(x)$

- estimate the root, x_i, where $f(x)$ crosses the x axis
- the next estimate, x_{i+1}, is the place where the tangent to $f(x)$ at point $(xi, f(x_i))$ crosses the x axis

From Figure 3.2 the equation of this tangent line is:

$$y - f(x_i) = f'(x_i)(x - x_i) \qquad \text{(Eq3.39)}$$

This equation has a zero at

$$x = x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \qquad \text{(Eq3.40)}$$

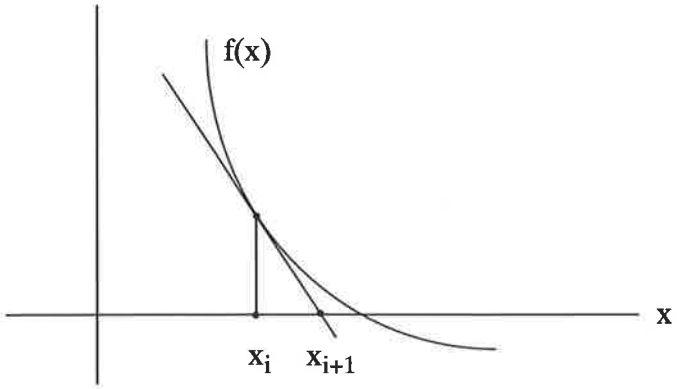


Figure 3.2 Newton-Raphson iteration for zero finding.

The above recursive iteration can be used to solve many equations. For division, computing the reciprocal is of interest. Thus, the equation $f(x) = \frac{1}{x} - b = 0$ can be solved using the above recursion (where b is the reciprocal) as follows:

$$f(x) = \frac{1}{x} - b$$

Then

$$f'(x) = -\left(\frac{1}{x}\right)^2$$

At $x = x_i$, Eq.3.40 becomes:

$$x_{i+1} = x_i - \frac{1/x_i - b}{-1/x_i^2} = x_i + x_i - x_i^2 b = x_i(2 - x_i b)\,. \qquad \text{(Eq3.41)}$$

(Eq3.42)

Then the following method could be used to compute *1/b*:

1. Scale *b* to lie in the range $0 < b < 1$ and get an approximate value of *1/b* $x_0 = 1$.
2. Iterate $x_{i+1} = x_i(2 - x_i b)$ until reaching an x_n that is accurate enough.
3. Compute x_n and reverse the scaling done in step 1.

An example:

Find *1/b* where *b=0.85* (ε = error).

$x_0 = 1$ $\qquad\qquad$ $\varepsilon_0 = 0.017647$

$x_1 = 1(2 - 0.85 \times 1) = 1.15$ $\qquad$ $\varepsilon_1 = 0.02647$

$x_2 = 1.15(2 - 0.85 \times 1.15) = 1.175$ $\qquad$ $\varepsilon_2 = 0.00147$

$x_3 = 1.175(2 - 0.85 \times 1.175) = 1.176$ $\qquad$ $\varepsilon_3 = 0.00047$

The quadratic convergence (i.e. $\varepsilon_{i+1} \le \varepsilon_i^2$) of this method can be proved as follow:

$$x_{i+1} = x_i(2 - bx_i)\,, \qquad \text{(Eq3.43)}$$

$$\varepsilon_i = \frac{1}{b} - x_i \qquad \text{(Eq3.44)}$$

or

$$x_i = \frac{1}{b} - \varepsilon_i = \frac{1 - b\varepsilon_i}{b}, \qquad \text{(Eq3.45)}$$

and

$$\varepsilon_{i+1} = \frac{1}{b} - x_{i+1} = \frac{1}{b} - [x_i(2 - bx_i)] = \frac{1 - 2bx_i + (bx_i)^2}{b}, \qquad \text{(Eq3.46)}$$

substituting for Eq.3.45,

$$\varepsilon_{i+1} = \frac{1 - 2b\left(\frac{1 - b\varepsilon_i}{b}\right) + (1 - b\varepsilon_i)^2}{b} = b\varepsilon_i^2; \qquad \text{(Eq3.47)}$$

recall that $b < 1$,

therefore

$$\varepsilon_{i+1} \leq \varepsilon_i^2. \qquad \text{(Eq3.48)}$$

The division execution time of the Newton-Raphson approximation can be reduced by using a ROM look-up table. For example, computing the reciprocal of a 32-bit number can start by using 1024×8 ROM to provide the 8 most significant bits, the next iteration will provide 16 bits, and the third iteration produces a 32-bit quotient.

The advantage of iteration is that it doesn't require special divide hardware, but can instead use the multiplier (which, however, requires extra control). There are two disadvantages with inverting by iteration: The first is that the IEEE standard requires division to be correctly rounded, but iteration only delivers a result that is close to the correctly rounded answer. In the case of Newton-Raphson iteration, which computes *1/b* instead of *a/b* directly, there is an additional problem. Even if *1/b* were correctly rounded, there is no guarantee that *a/b* will be. Take *5/7* as an example: To two digits of accuracy *1/7* is *0.14*, and *5×0.14* is *0.70*, but *5/7* is *0.71*. The second disadvantage is that iteration does not give a remainder. This is especially troublesome if the floating-point divide hardware is being used to perform integer division, since a remainder operation is present in almost every high-level language.

Normally a divider is not used as frequently as a multiplier. Furthermore, the divider is used not only for division function, but also as a vehicle for the mixed dynamic/static approach. Thus, a *nonrestoring* algorithm is chosen for these purposes. The main elements of the *nonrestoring* division are shift registers and a adder/subtract. Since TDFL is a very low-power, high-density logic family and is compatible with other static logic families described in Chapter 2, it can be conveniently used for the shift registers. In the following sections a DCFL based adder/subtracter, TDFL registers and an 8-bit divider will be discussed as a test bench for the mixed dynamic/static approach.

3.2 Analysis of Adder Designs for GaAs VLSI

Conventional fixed time adders are ripple-carry adder, carry look-ahead adder, carry select adder, carry skip adder and binary carry look-ahead adder. Because the fastest Silicon adder is based on high fan-in and fan-out capability, and GaAs technology is restrained to low fan-in and fan-out as described in Chapter 2. The fastest adder in Silicon is not necessarily the fastest in GaAs technology. Therefore, in the following sections various adders are evaluated for GaAs VLSI implementation. The circuits are based on DCFL gates and are fully optimized in terms of speed, area and power dissipation.

3.2.1 Ripple-carry Adder

The main problem in building an adder for n-bit numbers is propagating the carries. The most obvious way to solve this is using a *ripple-carry adder*, which is formed by cascading n full adders as shown in Figure 3.3. The c_{i+1} output of the ith adder is fed into the c_{i+1} input of $i+1$th adder. The lowest order carry in c_0 is set to 0. Since c_0 is zero, the lowest order adder could be a half adder. However, c_0 can be set to 1 to perform subtraction later.

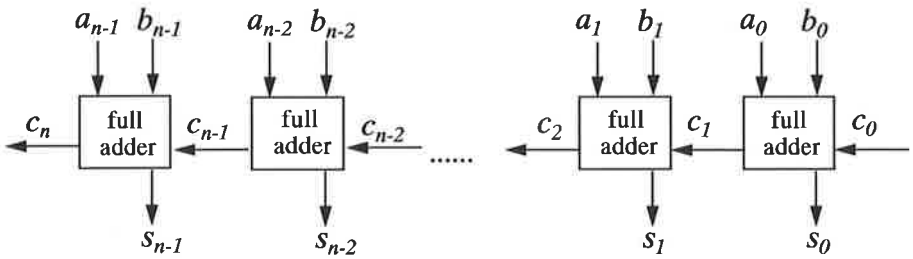


Figure 3.3 Block diagram of a ripple-carry adder.

The full adder is defined by the following logic equations:

$$s_i = a_i\bar{b}_i\bar{c}_i + \bar{a}_i b_i\bar{c}_i + \bar{a}_i\bar{b}_i c_i + a_i b_i c_i = a_i \oplus b_i \oplus c_i \qquad \text{(Eq3.49)}$$

$$c_{i+1} = a_i b_i + b_i c_i + c_i a_i \qquad \text{(Eq3.50)}$$

To implement the full adder in GaAs DCFL, these equations must be represented by *nor* and *invert* functions.

$$s_i = \overline{\left(\overline{a_i + \bar{b}_i + \bar{c}_i}\right) + \left(\overline{\bar{a}_i + \bar{b}_i + c_i}\right) + \left(\overline{\bar{a}_i + b_i + \bar{c}_i}\right) + \overline{(a_i + b_i + c_i)}}\,, \qquad \text{(Eq3.51)}$$

$$c_{i+1} = \overline{\overline{(a_i + b_i)} + \overline{(b_i + c_i)} + \overline{(c_i + a_i)}}\,. \qquad \text{(Eq3.52)}$$

From Eq.3.52, there are two logic gates involved in computing c_{i+1} from c_i. Thus, if the least significant bit generates a carry, and that carry gets propagated all the way to the last adder, the c_0 signal will pass through $2n$ logic gates before the final gate can determine whether there is a carry out of the most significant place. Therefore, the ripple-carry adder is a slow but cheap adder. It can be built with only n simple cells, connected in a simple, regular way.

3.2.2 Carry Look-ahead Adder

The ripple-carry adder requires $2n$ logic gate delays because the carry bit has to ripple through all n full adders. A *Carry look-ahead adder* (CLA) avoids the $2n$ logic gate delays by accelerating the computation of carries using a treelike circuit.

The key observation is that in ripple-carry addition, for $i \geq 1$, the full adder FA_i has two of its inputs, namely a_i and b_i, ready long before the carry-in c_i is ready. The idea behind the carry look-ahead adder is to exploit this partial information. The goal is to represent c_i in terms of a_i and b_i. To accomplish this, take a 4-bit slice as an example to derive its equations.

The general carry equation Eq.3.50 can be rewritten as [43]:

$$c_{i+1} = a_i b_i + c_i (a_i + b_i)\,. \qquad \text{(Eq3.53)}$$

This general equation for the carry can be verbalized further as follows: there is a carry into the $i+1$th stage if a carry is generated locally at the ith stage or if a carry is propagated through the ith stage from the $i-1$th stage. Carry is generated locally if both a_i and b_i are ones, and it is expressed by the generate equation:

$$g_i = a_i b_i. \qquad \text{(Eq3.54)}$$

A carry is propagated only if either a_i or b_i is one, and the equation for the propagate term is:

$$p_i = a_i + b_i. \qquad \text{(Eq3.55)}$$

Substitute the generate and propagate equations in Eq.3.53, the carry equations are functions only of the previous generate and propagate terms:

$$c_1 = g_0 + p_0 c_0,$$

$$c_2 = g_1 + p_1 c_1.$$

Substitute c_1 into the c_2 equation (in general substitute c_i in the c_{i+1} equation):

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0,$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0,$$

$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0;$$

therefore, the general equation of carry look-ahead addition is:

$$c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_0 c_0. \qquad \text{(Eq3.56)}$$

Above equations implies that a carry look-ahead adder requires one logic gate level to form *p* and *g*, two levels to form the carries, and two for the sum, for a total of five logic gate levels, if it were not limited by fan-in and modularity. This is a vast improvement over the *2n* levels required for the ripple-carry adder. Unfortunately, the fan-in is a serious limitation since for *n*-bit look-ahead the required fan-in is *n*, and modularity requires a somewhat regular implementation structure so that similar parts can be used to build adders of differing operand sizes. The latter modularity requirement, in fact, is what distinguishs the CLA algorithm from the Brent & Kung algorithm to be discussed in the next section.

The solution of the fan-in and modularity problems is to have several levels of carry look-ahead. This concept is illustrated by rewriting the c_4 equation (assuming fan-in of 4, or 5 if a c_0 term is required):

$$c_4 = g_0' + p_0' c_0,$$

where $g_0' = g_4 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$, and $p_0' = p_3 p_2 p_1 p_0 c_0$.

Notice the similarity between the new c_4 equation to that of c_1. Similarly, the equation for c_5 and c_6 resemble those for c_2 and c_3.

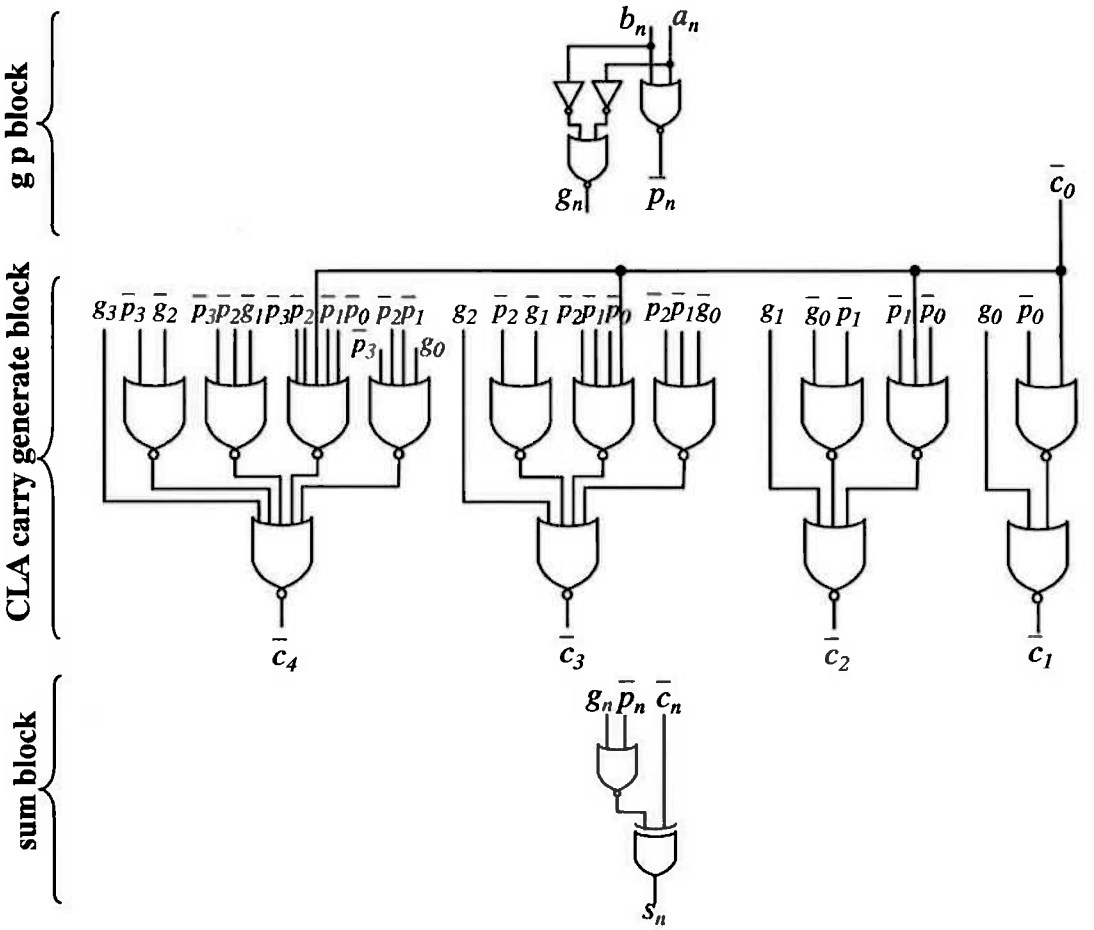


Figure 3.4 The logic diagram of a 4-bit CLA.

These equations should be transformed into their equivalent *nor* gates and *inverters* for GaAs DCFL implementation. The logic diagram of a 4-bit slice carry look-ahead adder is illustrated in Figure 3.4. Figure 3.5 shows another possibility using 4-bit CLA blocks to form an *n*-bit adder.

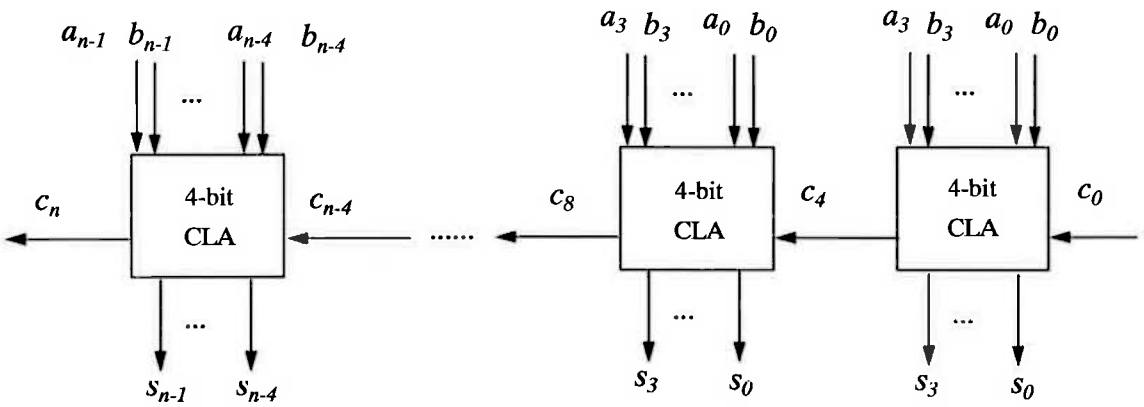


Figure 3.5 An *n*-bit adder built using the 4-bit CLA adders.

3.2.3 Brent & Kung Algorithm (Binary Carry Look-ahead Adder)

As mentioned in the previous section, the Brent & Kung Algorithm can be used to meet the modularity requirement for a CLA. The *Brent & Kung adder*, also called the *Binary Carry Look-ahead Adder*, like the CLA is based on parallel computation of the carries. It uses an associative operator "*o*" to compute the carry signals in a binary tree structure. The function of "*o*" is defined as follows [59]:

$$(g,p)\,o\,(g',p') \;=\; (g + (p \cdot g'), p \cdot p') \qquad \text{(Eq3.57)}$$

where *g, p, g'* and *p'* are boolean variables.

The carry signals are defined as:

$$c_i \;=\; G_i \qquad for \qquad i \;=\; 1, 2, \ldots, n, \qquad \text{(Eq3.58)}$$

where

$$(G_i, P_i) \;=\; \begin{cases} (g_1, p_1) & if \quad i \;=\; 1, \\ (g_i, p_i)\,o\,(G_{i-1}, P_{i-1}) & if \quad 2 \le i \le n, \end{cases} \qquad \text{(Eq3.59)}$$

and

$$(G_i, P_i) \;=\; (g_i, p_i)\,o\,(g_{i-1}, p_{i-1})\,o \ldots o\,(g_1, p_1)\,. \qquad \text{(Eq3.60)}$$

Using Eq.3.58 to Eq.3.60, c_i can be evaluated in any order from the given g_i's and p_i's. In other words, identical circuit elements can be arranged in a binary tree structure to implement the carry bits.

This can be illustrated in the following for an 8-bit binary carry look-ahead adder:

$$c_1 \;=\; g_1$$

$$c_2 \;=\; g_2 + p_2 \cdot g_1$$

$$c_3 \;=\; g_3 + p_3 \cdot c_2$$

$$c_4 \;=\; g_4 + p_4 \cdot g_3 + p_4 p_3 c_2$$

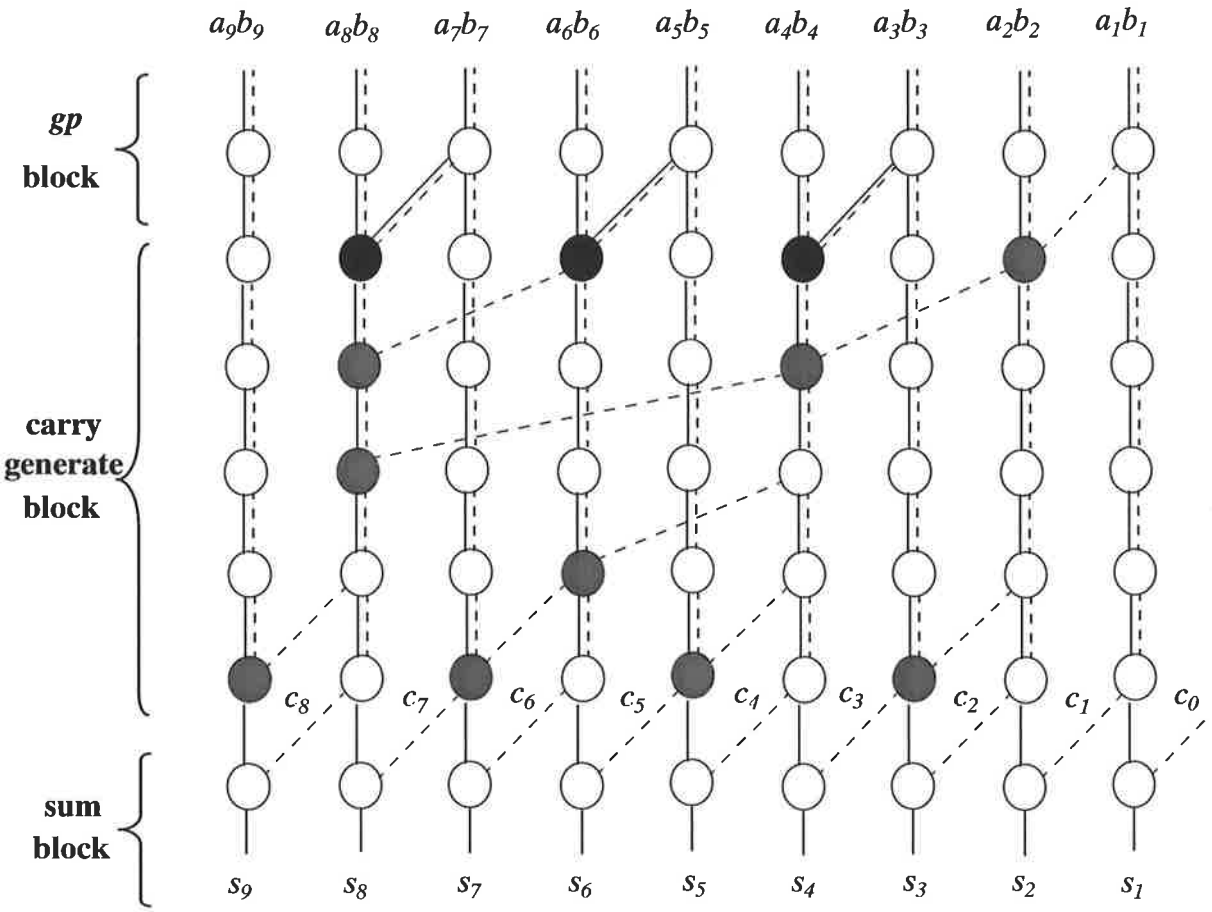
$$c_5 \;=\; g_5 + p_5 \cdot c_4$$

$$c_6 = g_6 + p_6 \cdot g_5 + p_6 p_5 c_4$$

$$c_7 = g_7 + p_7 \cdot c_6$$

$$c_8 = g_8 + p_8 \cdot g_7 + p_8 p_7 c_6$$

$$c_9 = g_9 + p_9 \cdot c_7$$



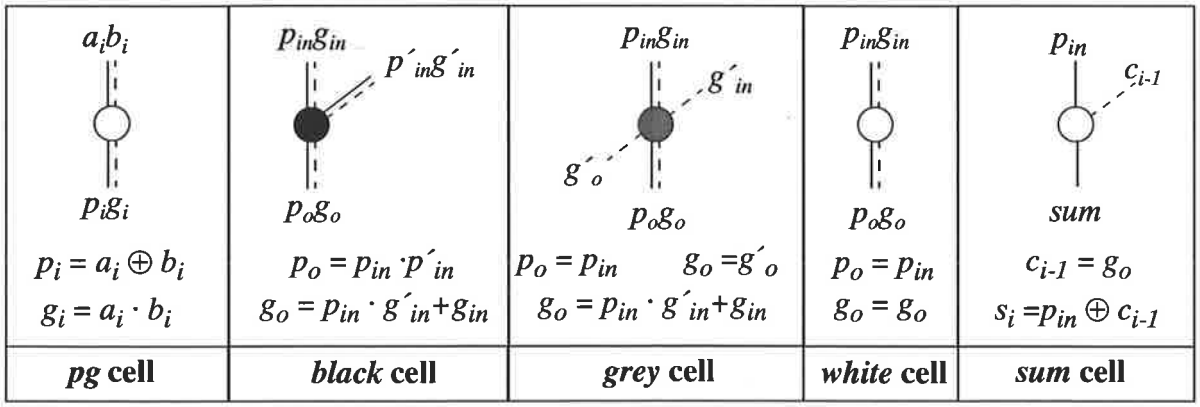


Figure 3.6 The block diagram of a 9-bit binary carry look-ahead adder.

The complete structure of a 9-bit binary carry look-ahead adder is illustrated in Figure 3.6. The similarity in the equations above results in a simple regular carry generator block consisting of only three cells, namely, *black* cell, *grey* cell and *white* cell. The black and grey cells perform the "*o*" operation and the white cells just transmit the data. The functions performed by each cells are also illustrated in Figure 3.6. The variables g_i' and p_i' are the g_i's and p_i's from the previous stage. The *pg* blocks, like the CLA, generate the g_i's and p_i's to the carry generator block and the sum blocks perform the *xor* function on the carries c_i and the propagate signals p_i from the *pg* blocks to generate the sum.

Figure 3.7 shows the logic diagrams of the cells in Figure 3.6. All the circuits are implemented by DCFL nor gates and inverters. SDCFL logic gates are used for driving large fan-out.

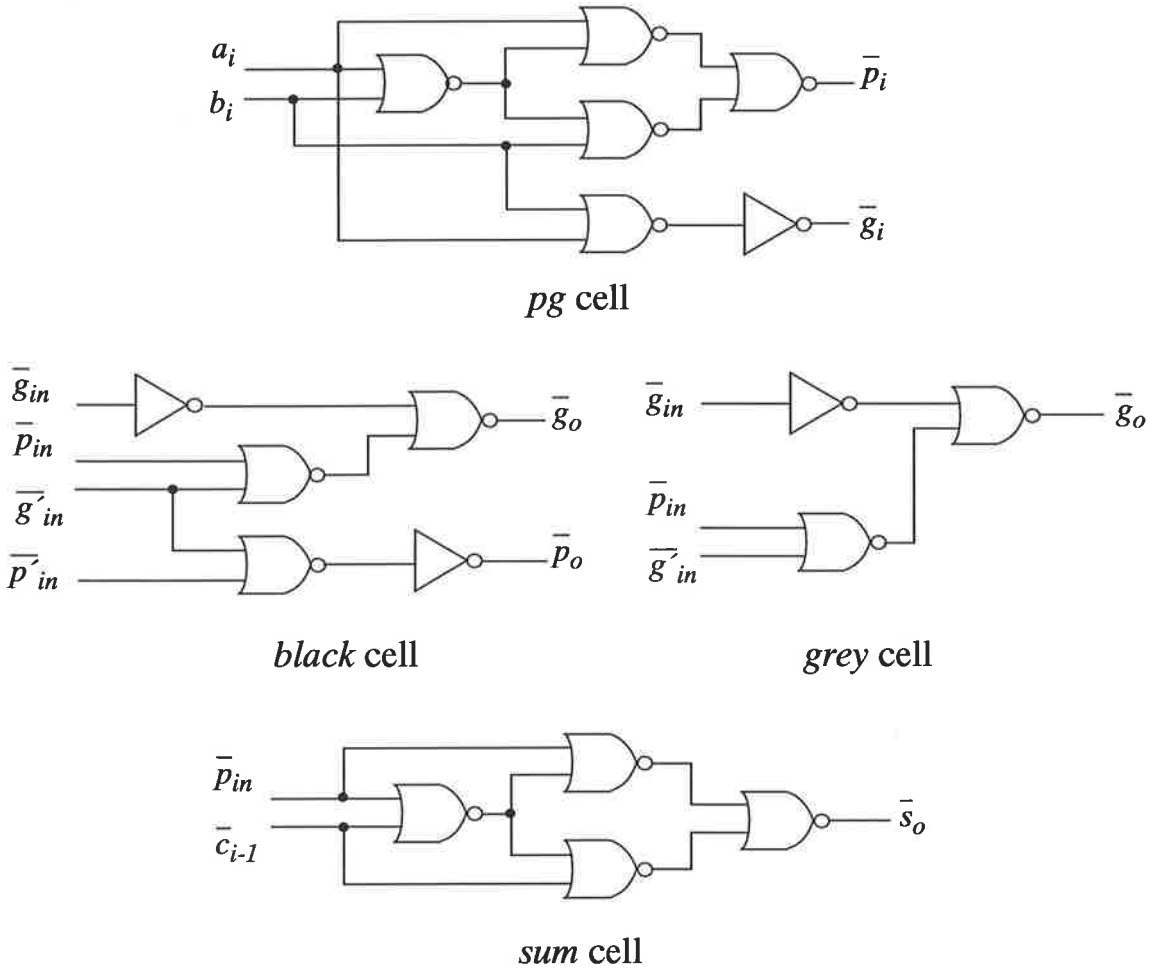


Figure 3.7 Logic diagrams of the cells of binary carry look-ahead adder.

3.2.4 Carry Select Adder

The principle of a *carry select adder* (CSA) [56] can be described as follows: Two additions are performed in parallel, one assuming the carry in is zero and the other assuming the carry in is one. When the carry in is finally known, the correct sum (which has been precomputed) is simply selected. An 8-bit carry select adder is illustrated in Figure 3.8. The 8-bit adder is divided into two half blocks, and the carry out from the lower half is used to select the upper half. If each block is computing its sum using a ripple carry adder as shown in Figure 3.8, then the design is about twice as fast at 50% more cost in area. Each block can also use CLA or any type of adder to further speed up the addition, but at more cost. The logic diagram of the multiplexer used in the CSA is illustrated in Figure 3.9. In Figure 3.9, c is the select signal, when $c = 0$, output signal $m = m_0$, otherwise, $m = m_1$.

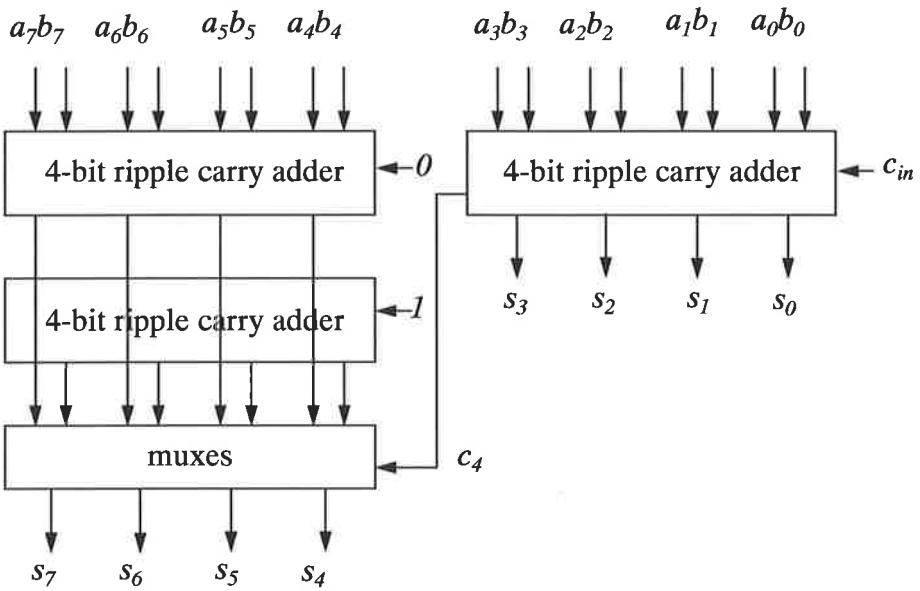


Figure 3.8 A simple carry select adder.

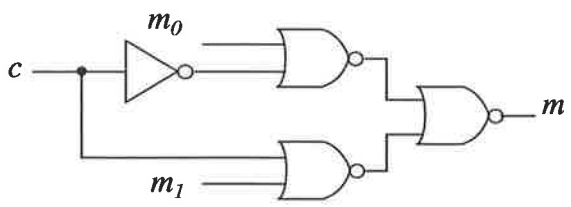


Figure 3.9 The logic circuit of a mux used in CSA.

3.2.5 Carry- Skip Adder

From Eq.3.50 it is clear that c_{i+1} is dependent on c_i, and the sum of two n-bit numbers is not achieved until the *n-1* bit carry out is known. However, it is worth noting that if a_i and b_i are the same (i.e. $a_i \oplus b_i = 0$), there is no need to know c_i in order to know c_{i+1}. That is:

If $a_i = b_i = 0$, then $c_{i+1} = 0$;

If $a_i = b_i = 1$, then $c_{i+1} = 1$.

Thus, an adder can be designed which is divided into several blocks, where a special circuit associated with each block detects quickly if all the bits to be added are different, in which case carry into the block will propagate to the next block otherwise it won't. The adder based on this principle is called *carry-skip adder.*

Figure 3.10 illustrates a carry-skip adder with each block being a ripple carry adder. The inputs of the *and* gates come from the *exclusive or* ($a_i \oplus b_i$) gate of each full adder [63].

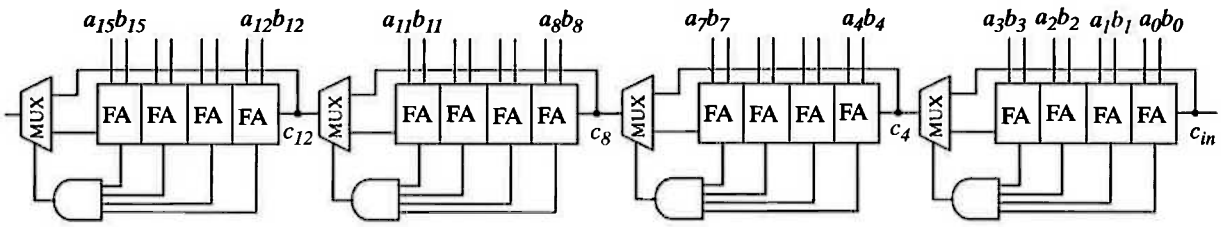


Figure 3.10 A carry-skip adder using multiplexers.

- If for each cell in the block $a_i \neq b_i$, then the output of the *and* gate is true, carry in can be selected to skip over the block.

- If $a_i = b_i$ for some i in the block, a carry is generated in the block (whether 1 or 0).

It is noted that if each block has a cell i such that $a_i = b_i$, no block is skipped, but in each block, a carry is generated; thus, the carries are propagated in parallel and the

total time of computation is bounded by the time of propagation of a carry in the largest block.

Figure 3.10 is called first level skip. By combing first level skips a second level skip can be made, and a third level skip can be obtained over second level skips and so on [66]. Figure 3.11 illustrates a three level carry-skip adder. The multi-level carry-skip adder has a better speed performance at a little more cost in terms of area and complication.

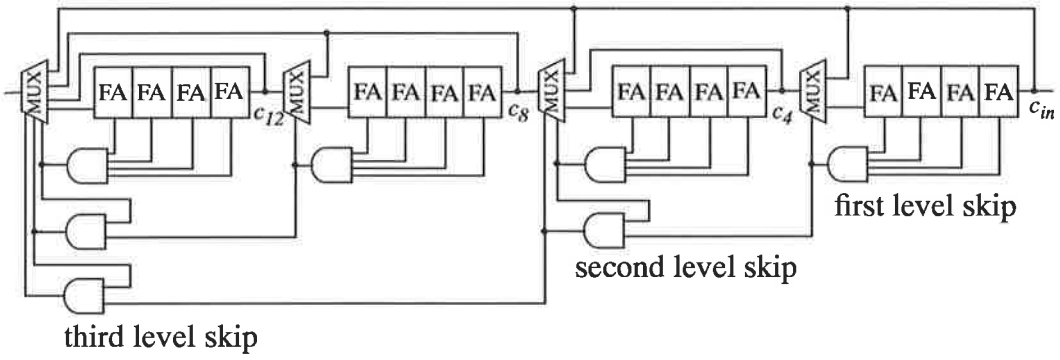


Figure 3.11 A three level carry-skip adder.

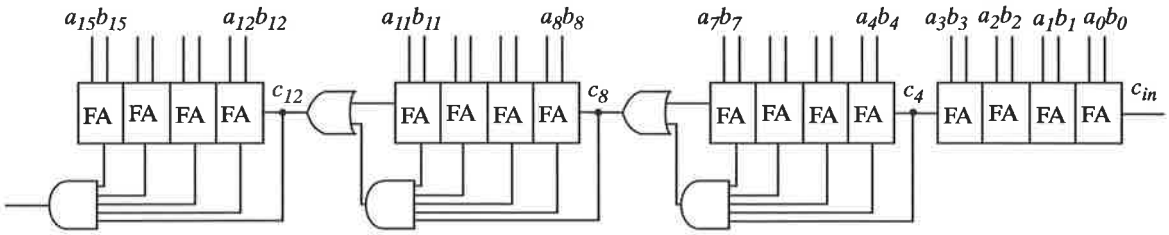


Figure 3.12 A carry-skip adder using *or* gates.

Figure 3.12 shows another version of carry-skip adder using *or* gates instead of *muxes* as shown in Figure 3.8 [41]. Because of the *AOI* structure discussed in **2.2.3**, an *or* function is easy to build between DCFL logic gates, and it is fast and more compact, while it is difficult to build a *mux* in GaAs, compared to CMOS. Thus, Figure 3.12 is a better solution than Figure 3.10. Note that the inputs of the *and* gates come from *or* gates ($a_i + b_i$) instead of from *xor* gates ($a_i \oplus b_i$). The *and* gates are implemented by DCFL *nor* gates and *inverters*.

Assuming one gate delay is *d*, then it takes *2d* for a signal to pass through a full adder. Thus, it will take *2kd* for a carry to ripple across a block of size *k*, and *2d* for a carry to skip a block. The longest signal path in the carry-skip adder starts with a carry being generated at the *0*th position. Then it takes *2kd* to ripple through the first block, takes $\frac{n}{2}kd-2$ to skip blocks (on average), and another *2kd* to ripple through the last block. Take a 24-bit adder for example: If it is broken into groups of 4-bit, it will take *24d* gate delay to preform an add. If it is grouped into 4-5-6-5-4, then the time of the adder drops to *22d*. In general, for a carry-skip adder, making the interior blocks larger will speed up the adder. In fact, the same idea of varying the block sizes can sometimes speed up other adders as well.

3.2.6 Performance Comparison of Different Adders for GaAs VLSI

The adders described in the previous sections were implemented using Vitesse H-GaAs-II 0.8μm Technology. GaAs DCFL *nor* gates and *inverter*s were used to perform logic functions, SDCFL gates are used to drive large fan-outs. MAGIC, a full custom layout tool, was used to draw the layouts. A full HSPICE simulation was carried out to obtain the characteristics of speed and power consumption. The performance in terms of speed, area and power consumption is the basis for selecting a particular adder type for GaAs VLSI.

Figure 3.13 to Figure 3.15 show the adders' delay, area and power consumption against the number of bits where the CLA uses the structure shown in Figure 3.7. For a small number of bits, the binary carry look-ahead adder has the best performance. For adders exceeding 8 bits in width, the carry select adder is the fastest adder with moderate area. This is because SDCFL is used as driver which can drive a large number of multiplexes. However, SDCFL also brings the highest power consumption. For a large number of bits, if the speed is the main concern, the Carry Select Adder is the right choice; if power dissipation is the main concern, the carry-skip adder can be used.

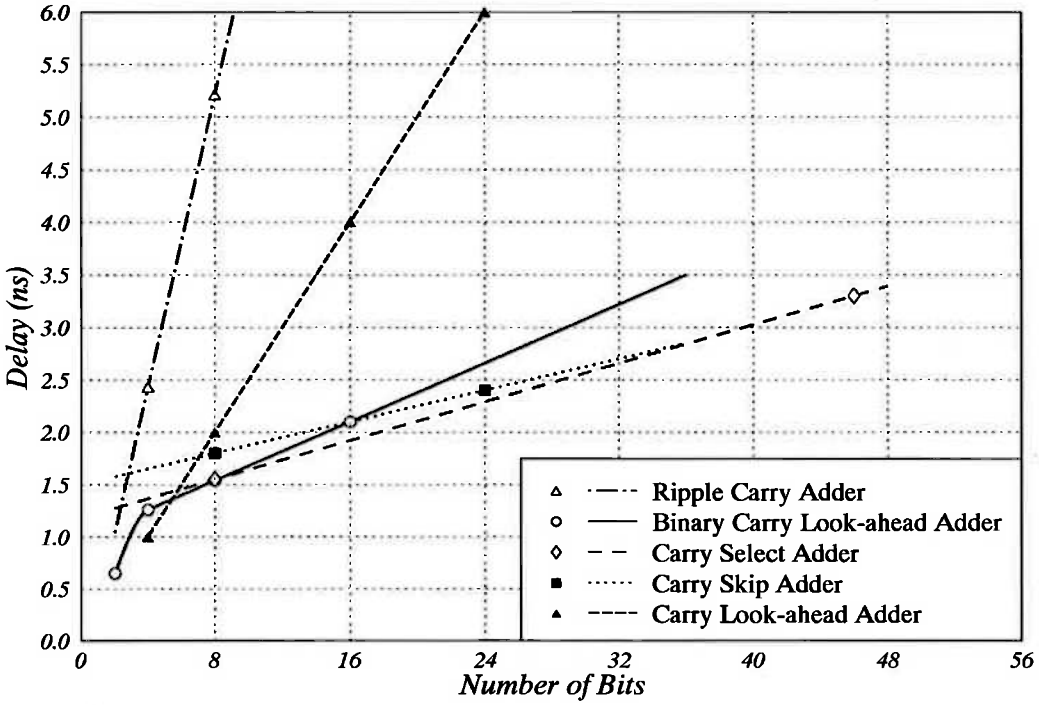


Figure 3.13 Delay against number of bits.

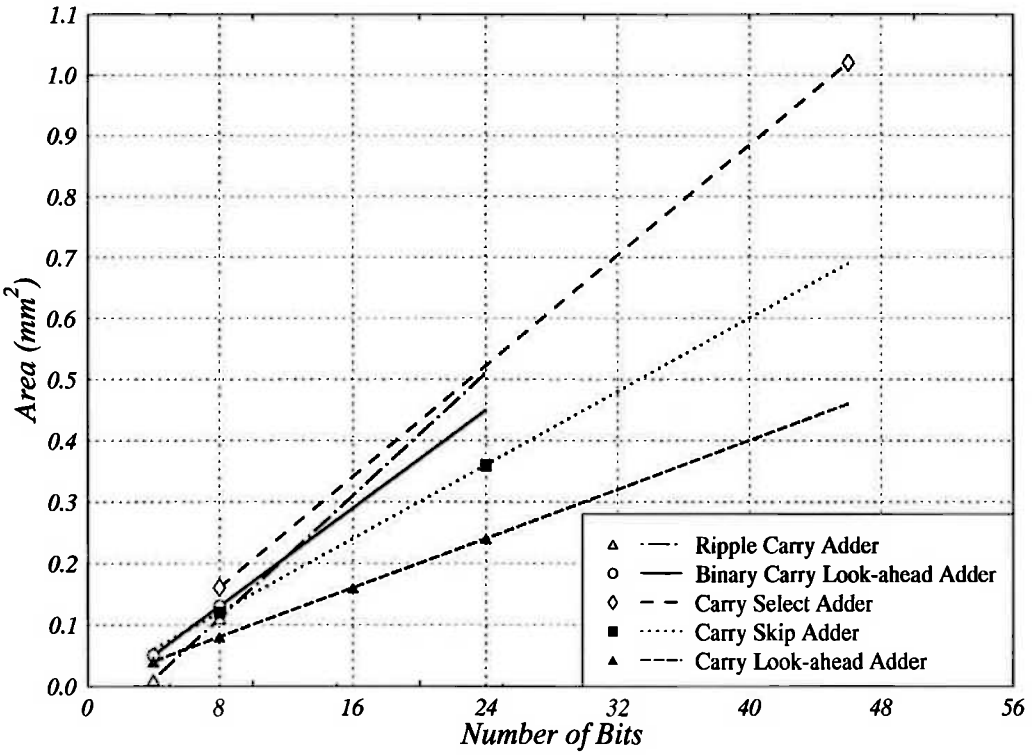


Figure 3.14 Area against number of bits.

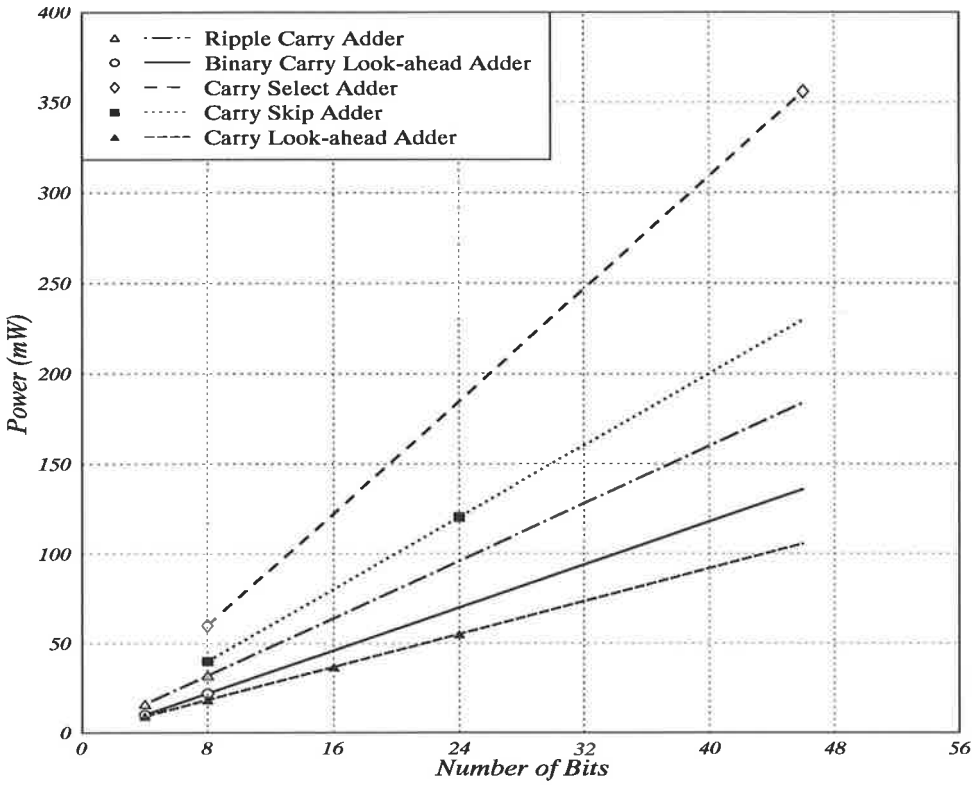


Figure 3.15 Power consumption against number of bits.

3.3 An 8-bit Serial Dynamic/Static Divider

3.3.1 The Structure of the 8-bit Divider

The block diagram of the 8-bit divider is illustrated in Figure 3.16 where *clk* is used to shift data, *R* is used to start the division, ctr_a, ctr_b and ctr_p are control signals used to load or shift data for register A, B and register P, respectively. *xor* gates either pass the value of register B or $\overline{B}$ to the adder/subtracter.

3.3.2 The Adder/Subtracter

From the discussion in previous sections it can be seen that for the addition of a number with a small number of bits, the *binary carry look ahead adder* has the best performance. An 8-bit *nonrestoring* divider was to be implemented to test the mixed GaAs *dynamic/static* approach, thus, a 9-bit binary carry look-ahead adder was chosen. The block and logic diagrams are shown in Figure 3.6 and Figure 3.7, respectively. Figure 3.17 shows the worst case delay of this 9-bit binary carry look-ahead adder from

HSPICE simulation results. The worst case occurs when all the inputs of *a* are high, all the inputs of *b* are low, and a carry in signal c_{in} is from low to high which causes the carry out signal c_9 from low to high. Note that c_9- is the complement of the carry out of the most significant bit c_9. The bottom graph shows the power consumption of the adder is about 25.4mW with 2V power supply.

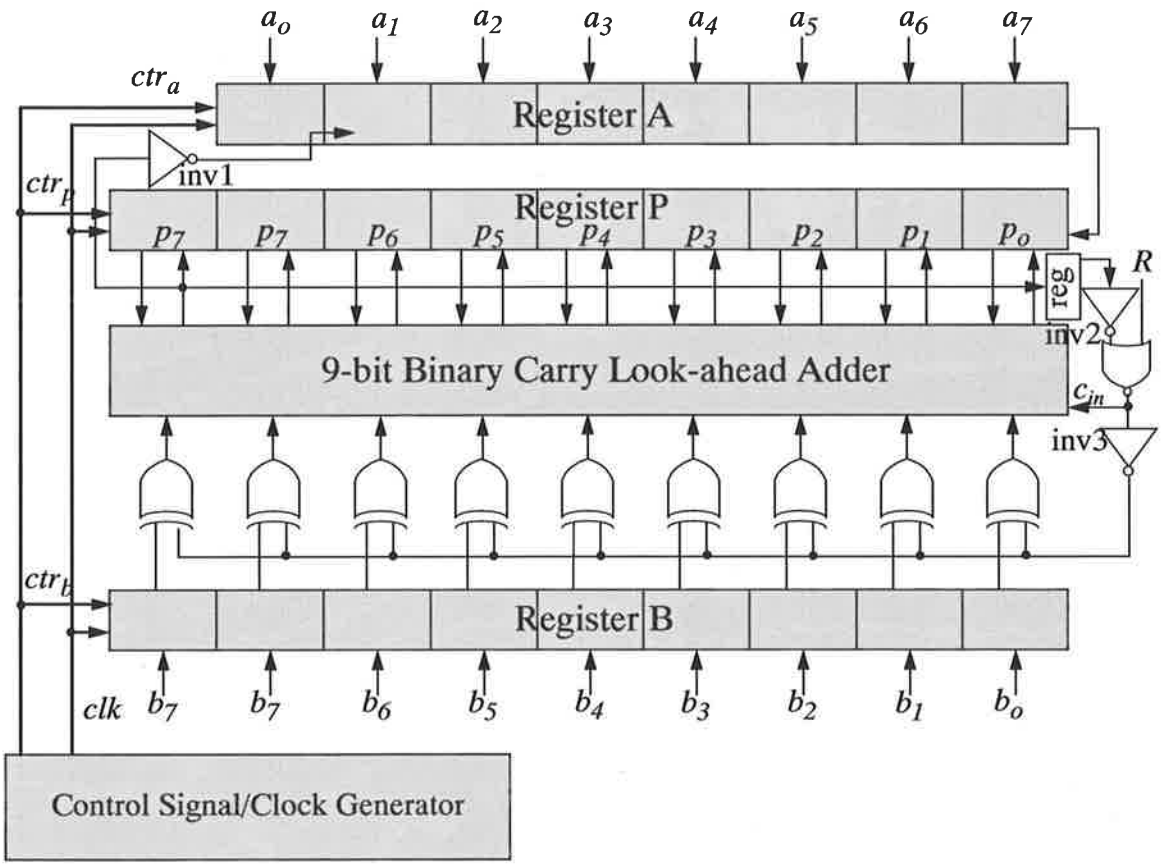


Figure 3.16 The logic architecture of the 8-bit divider.

Obtaining a two's complement number involves complementing each bit and then adding 1. Thus, to implement a - b using an adder, simply feed a and $\overline{b}$ (where $\overline{b}$ is the number obtained by complementing each bit of b) into the adder, and set the low-order bit to 1. This is why a carry in signal c_{in} is needed in the adders discussed.

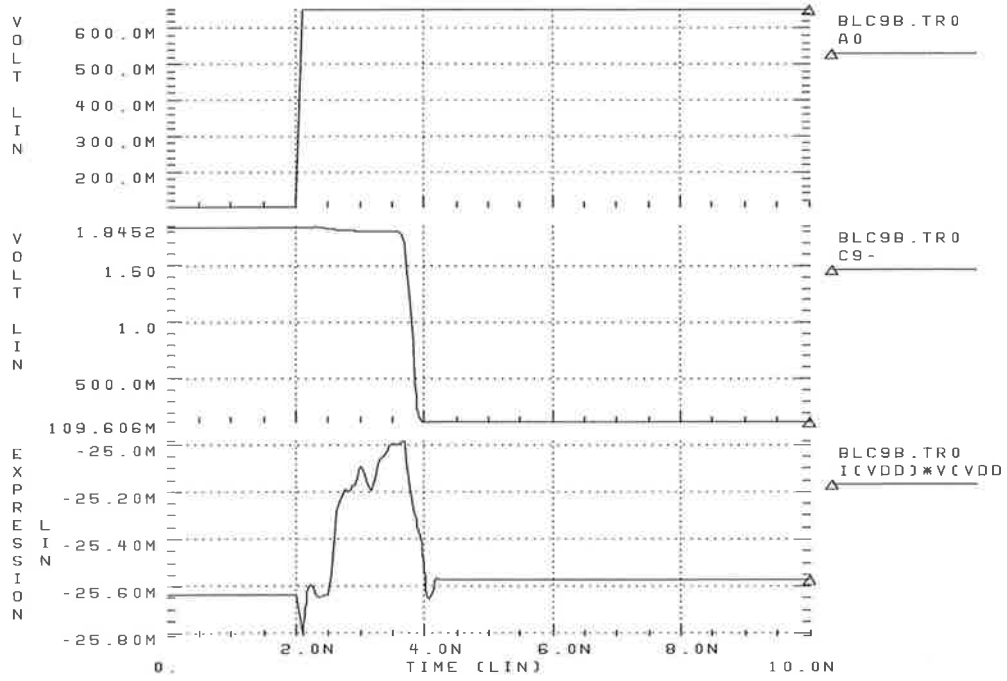


Figure 3.17 The hspice simulation results of the 9-bit binary carry look-ahead adder.

3.3.3 TDFL Based Registers

1. Why Use TDFL Registers?

· Suitable for the mixed dynamic/static approach

Figure 2.20 shows that TDFL logic levels are compatible with DCFL and other static logic families described in Chapter 2. In fact, the output of a TDFL gate can drive the input of a DCFL gate through a pass transistor, and the output of a DCFL gate can be directly connected to the input of a TDFL gate.

· Low power and high density

One can see from **2.2.5** that shift registers can be made easily by cascading TDFL inverters. TDFL shift registers have the advantages of low power and high density. The

latter can be seen from the layout of these two registers drawn in the same scale as shown in Figure 3.18; The former can be seen from Table 3.1 which displays a comparison between DCFL register and TDFL register. Compared to a DCFL register a TDFL register can save *82%* of the area and use only *6%* power dissipation. Figure 3.19 shows the HSPICE simulation results of a 8-bit TDFL shift register operated at *1Ghz* clock. It can be seen that after 8 clock cycles, the high input voltage V_{IN} shifts to the output of the 8th shift register (V_{O8}).

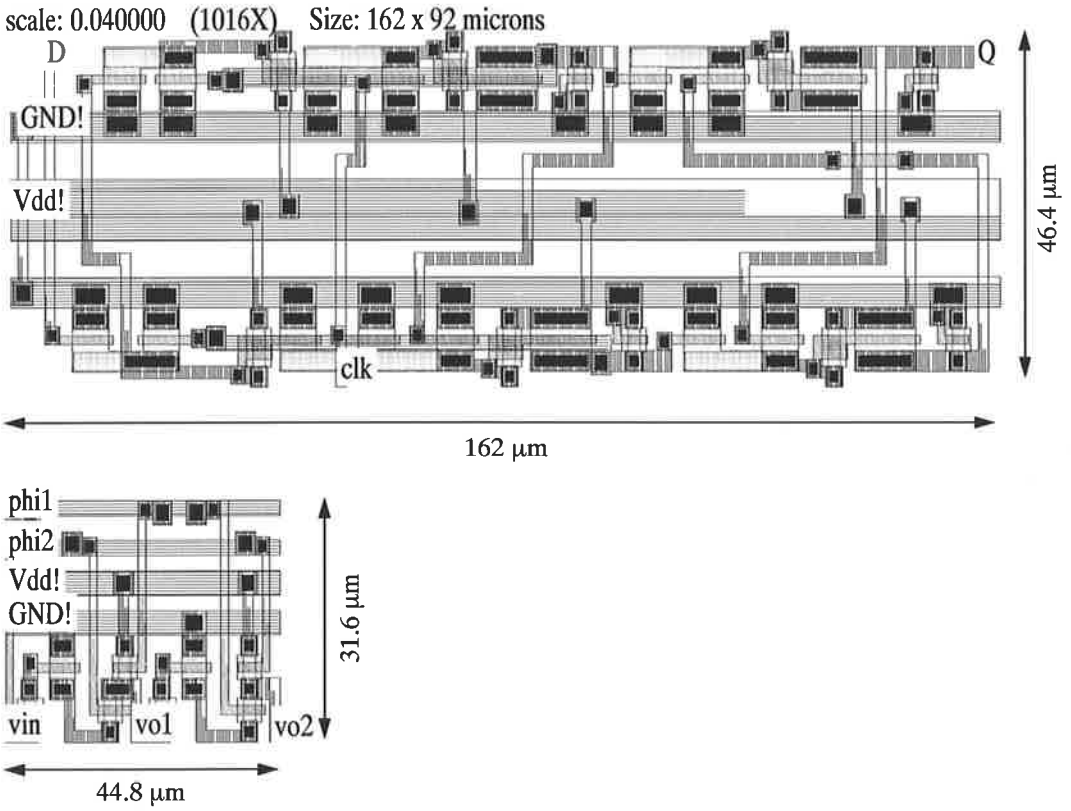


Figure 3.18 The layouts of a DCFL and a TDFL shift register

Table 3.1Comparison of DCFL and DCFL registers.

	transistors	area (μm)	delay (ns)	power dissipation (mW)
DCFL register	27	7516.8	1.17	1.8
TDFL register	16	1415.7	1.0	0.1

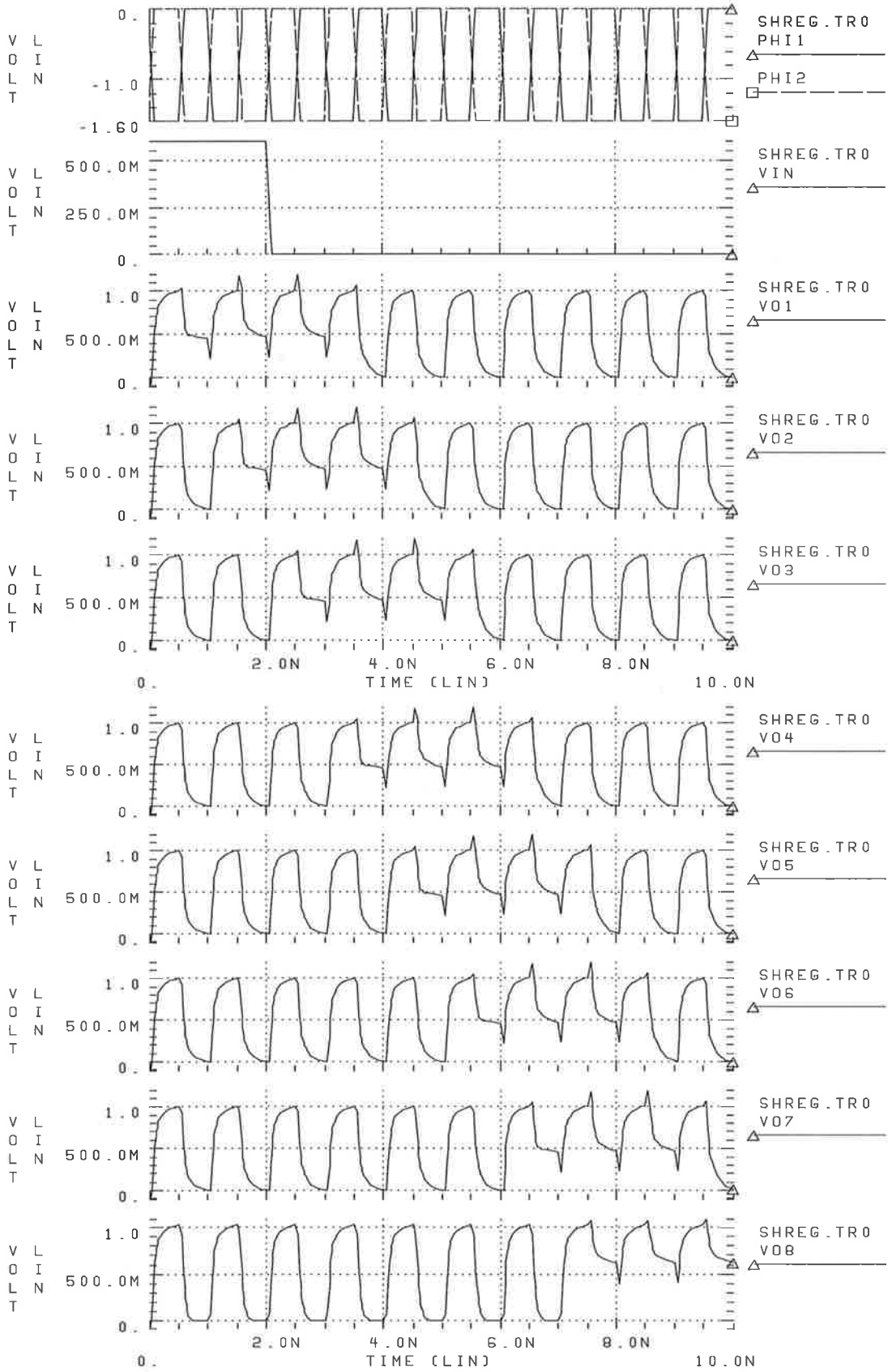


Figure 3.19 The HSPICE simulation results of a 8-bit TDFL shift register.

2. TDFL Shift Register A

Register A in Figure 3.16 requires both serial and parallel inputs and outputs. These was done for TDFL registers by adding pass transistors at the input. When data is loaded in Register A, the next clock can't arrive until the 9-bit addition/subtraction is finished. Since it takes about 2ns to add two 9-bit integers for the binary carry look-ahead adder as shown in Figure 3.17, the TDFL registers need to refresh the data or use low frequency clock. Because clock is used to shift data, the low frequency will affect the total speed of the divider. Thus, a pass transistor is added between input and output of the registers to refresh the data until the addition is completed. The schematic diagram of register A is illustrated in Figure 3.20, and the layout is shown in Figure 3.21 with signals defined as follows:

Vin_s—serial input,
Vin_p—parallel input,
Vo2—serial output,
Vo_p—parallel output,
phi1h—shift data,
phi1h—refresh data,
phi1Sa—control signal for parallel input data,
phi1/phi2—nonoverlapping clocks.

} ctr_a

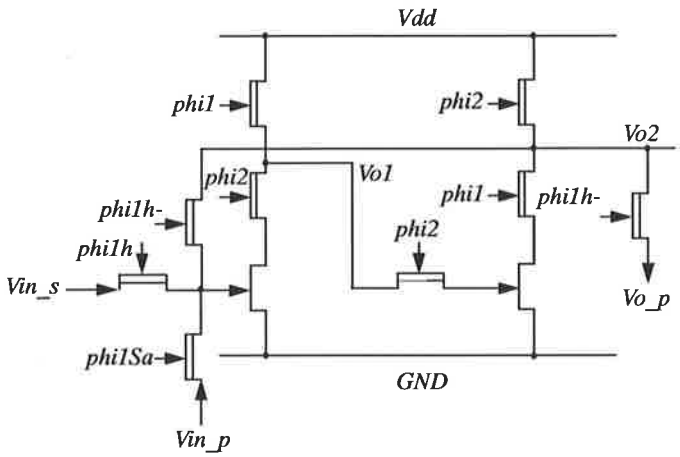


Figure 3.20 Schematic diagram of register A.

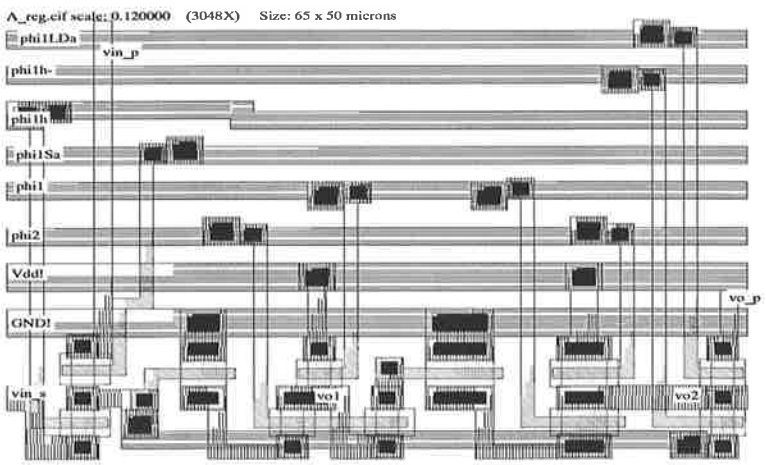


Figure 3.21 The layout of register A.

3. TDFL Shift Register P

Register P is very similar to register A except register P needs a clear function to clear the register before addition. This was done by connecting a pass transistor between output to ground as shown in Figure 3.22. The layout of register P is shown in Figure 3.23 with the signal definition similar to register A where control signals clr_p are *phi1h, phi1h-, phi1Sp* and *clr* is the clear signal.

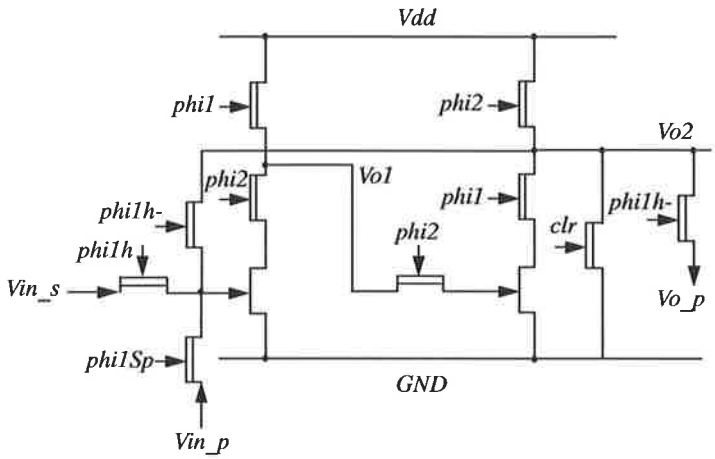


Figure 3.22 The schematic diagram of register P.

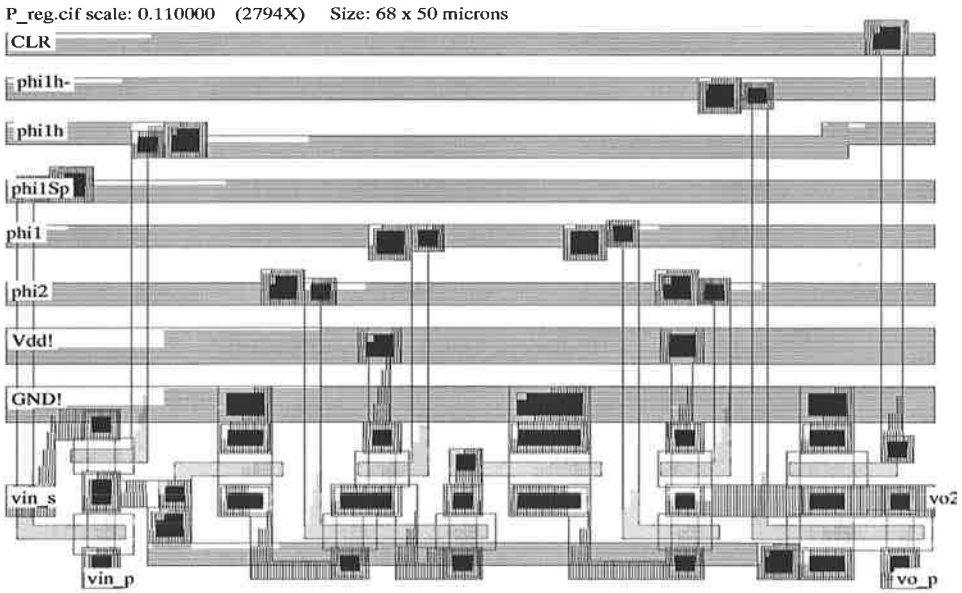


Figure 3.23 The layout of register P.

4. TDFL Shift Register B

Register B doesn't need to be shifted, thus, it has only one input v_{in_s}, and *phi1h-* is used to refresh the data (together with shift data signal *phi1h* are the control signal ctr_b) as illustrated in Figure 3.26. However, the output of register B v_{o_p} needs to connect to a *xor* gate which either passes input value B to the adder/subtracter or acts as an inverter to deliver $\overline{B}$ to the adder/subtracter. *Exclusive-or* (*xor*) gates can be constructed using three TDFL *nand* gates and one DCFL *nand* gate [68]. This kind of *xor* gate does not have too much advantage over DCFL and is less reliable, so a DCFL *xor* gate is used. Figure 3.25 shows the layout of register B with a DCFL *xor* gate.

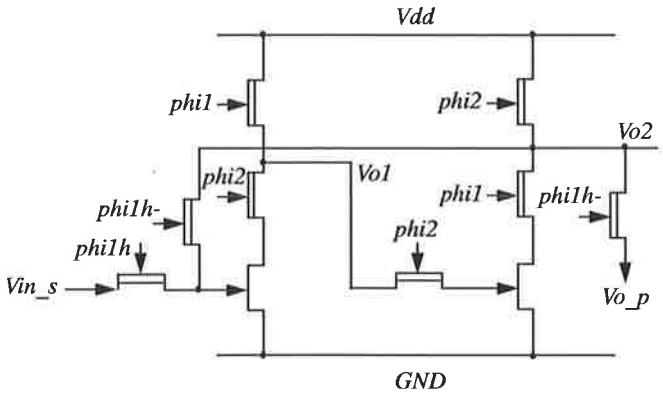


Figure 3.24 The schematic diagram of register B.

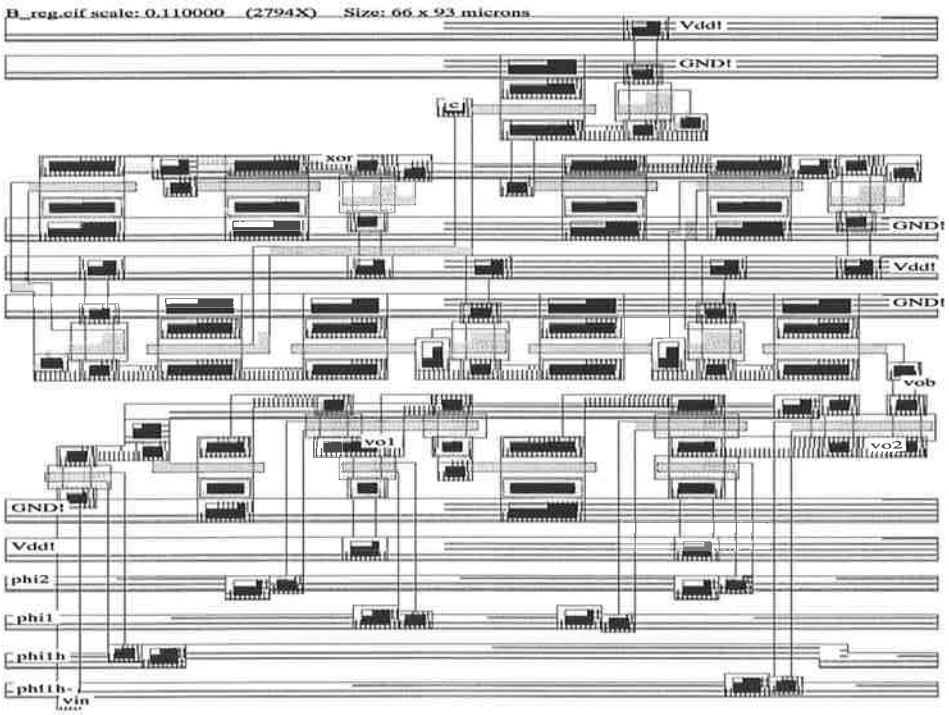


Figure 3.25 The layout of register B connected to a DCFL *xor* gate.

3.3.4 The Control Circuit and The Clock Generator

1. The Control Signals

Referring to the architecture of the 8-bit divider shown in Figure 3.16, *inv2* and *inv3* are DCFL inverters, while *inv1* and *reg* are a TDFL inverter and register. The schematic diagrams of *inv1* and *reg* are illustrated in Figure 3.26(a) and (b), respectively. Figure 3.27 shows the required timing diagram of the control signals with control signals *phi1Sa = phi1h = phi1set.*

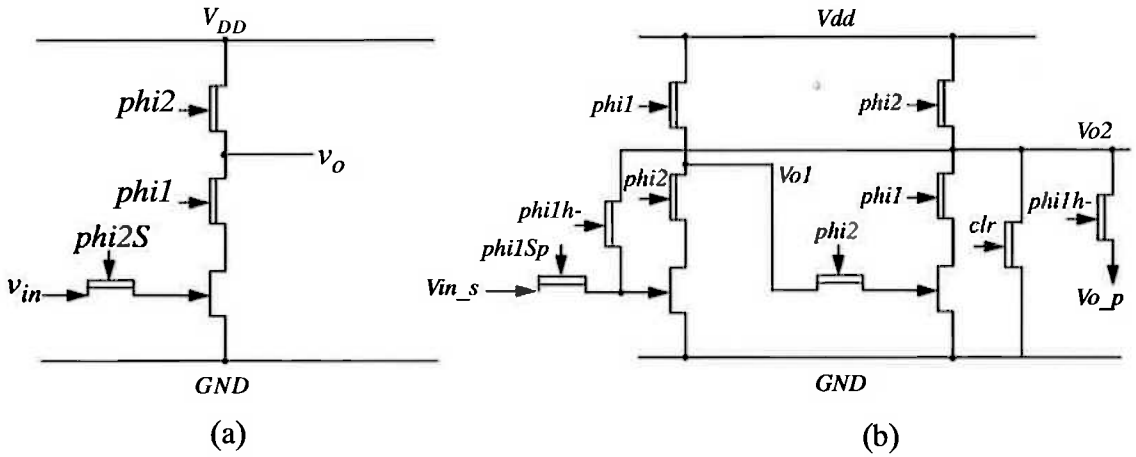


Figure 3.26 The schematic diagram of (a) *inv1*, (b) *reg*.

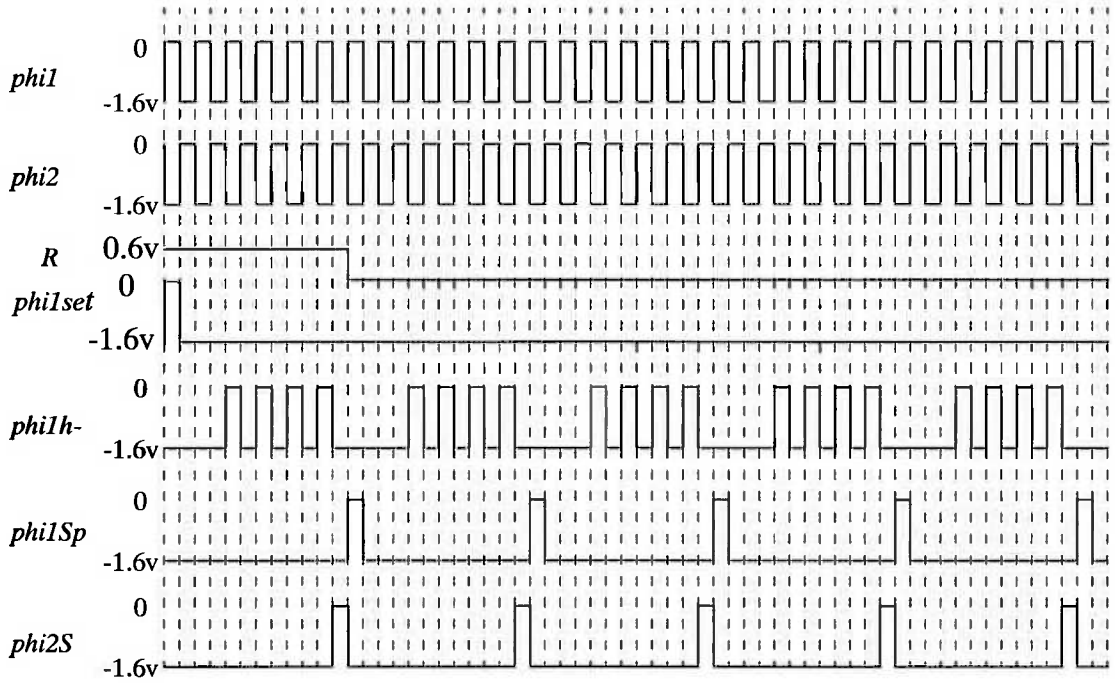


Figure 3.27 The timing diagram of control signals.

The control signals are generated by a 6-bit ring counter shown in Figure 3.28. The structure is simply a TDFL shift register with each serial output connected back to the next serial input. The first element in the counter is initially set to logic 1 and the remaining elements are set to logic 0 as shown in Figure 3.29 (a) and (b). This is then shifted. Since the voltage level of the clock is from -1.6v to 0v, a negative voltage supply V_{ss} = -2v and voltage level shift diodes are needed as illustrated in Figure 3.30.

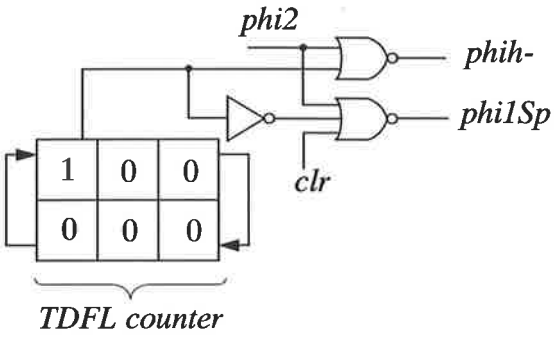


Figure 3.28 Control circuit block diagram.

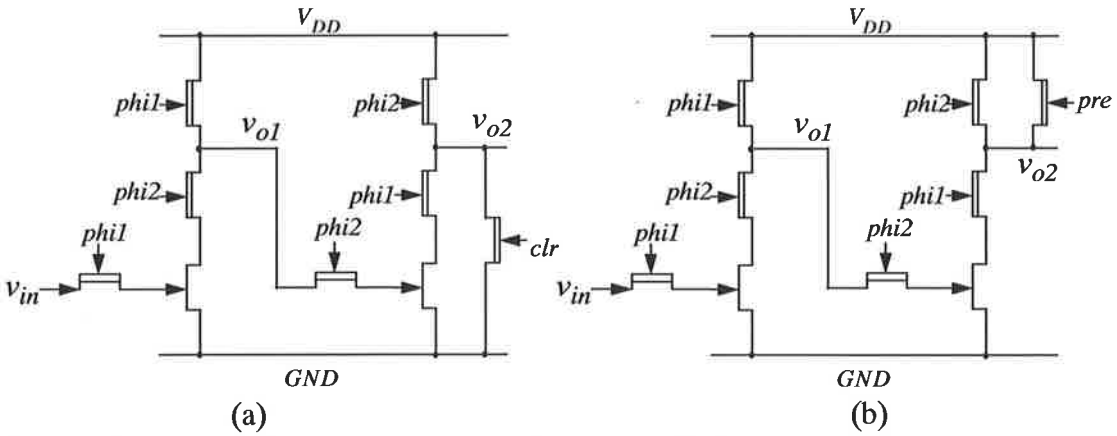


Figure 3.29 The schematic diagram of TDFL shift register with (a) *clear* function (b) *preset* function.

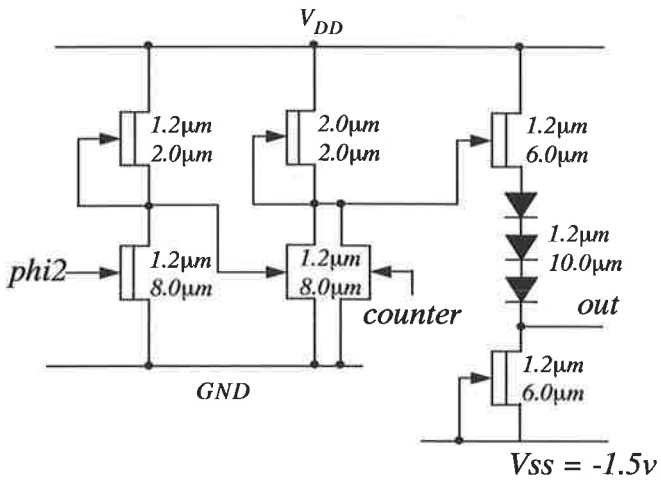


Figure 3.30 The schematic diagram of the special *nor* gate.

2. The Two Nonoverlapping Clock Generator/Driver

Figure 3.31 illustrates the clock schematic diagram for the generation of two non-overlapping clock signals from a single clock input. The clock generator consists of three stages. The first stage has the normal DCFL *nor* gates and inverter. The second stage has SBFL inverters as the interface between the first stage and the driver. The last stage is the clock driver. Since the capacitive load is around 600 fF (from layout extraction), the clock driver consists of a large SDCFL inverter and a large modified SBFL inverter. The detailed size of the driver is shown in Figure 3.32. The HSPICE simulated results of the control signals and the clock generator are shown in Figure 3.33. These results are simulated with *typical-typical* (*tt*) process parameters at *1Ghz*. The simulation shows that the generated control signals and the nonoverlapping clocks meet the requirement of the 8-bit divider, as shown in Figure 3.27. However, when circuits become large, for example, for 64-bit double precision floating point divider, the *clock skew* for TDFL will be a problem. Therefore, it is suggested that the TDFL gate or the mixed static/dynamic logic should only be used locally.

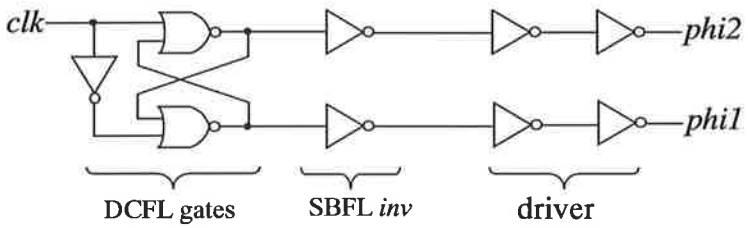


Figure 3.31 The schematic diagram of the nonoverlapping clocks generator.

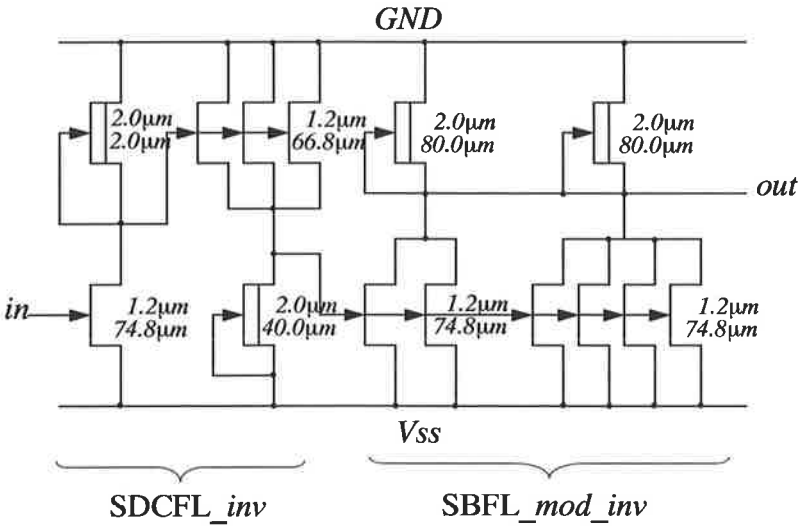


Figure 3.32 The schematic of the clock driver.

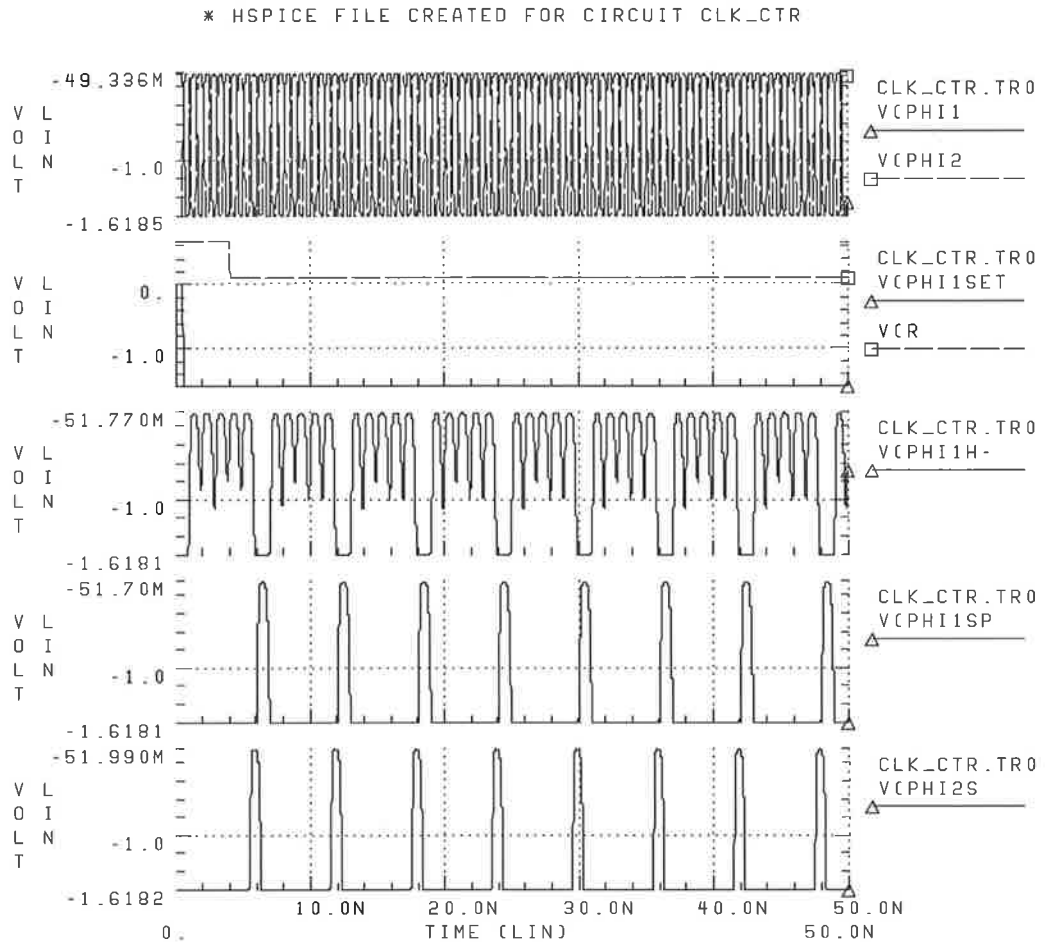


Figure 3.33 The simulated results of the control signals and clock generator.

3.3.5 Simulated Results for the 8-bit Divider

The HSPICE simulated results of the 8-bit divider are shown in Figure 3.34 with *tt* process parameters using the same example of 15/6 as used in the beginning of this Chapter. The graphs in Figure 3.34 from second bottom to top one show the quotient *V*(*AB*_7/vo_p)···V(*AB*_1/vo_p), V(*AB*_0/vo_p) = 00000010 which is the correct answer. The last graph is the power consumption which is only *0.35mW* on average. It can be seen that the delay of the 8-bit divider is *49ns* with *1GHz* clock. The floor plan of the 8-bit divider is illustrated in Figure 3.36. Figure 3.36 shows the layout of the divider using GaAs compatible I/O pads [69]. The area of the die is *0.31* mm^2.

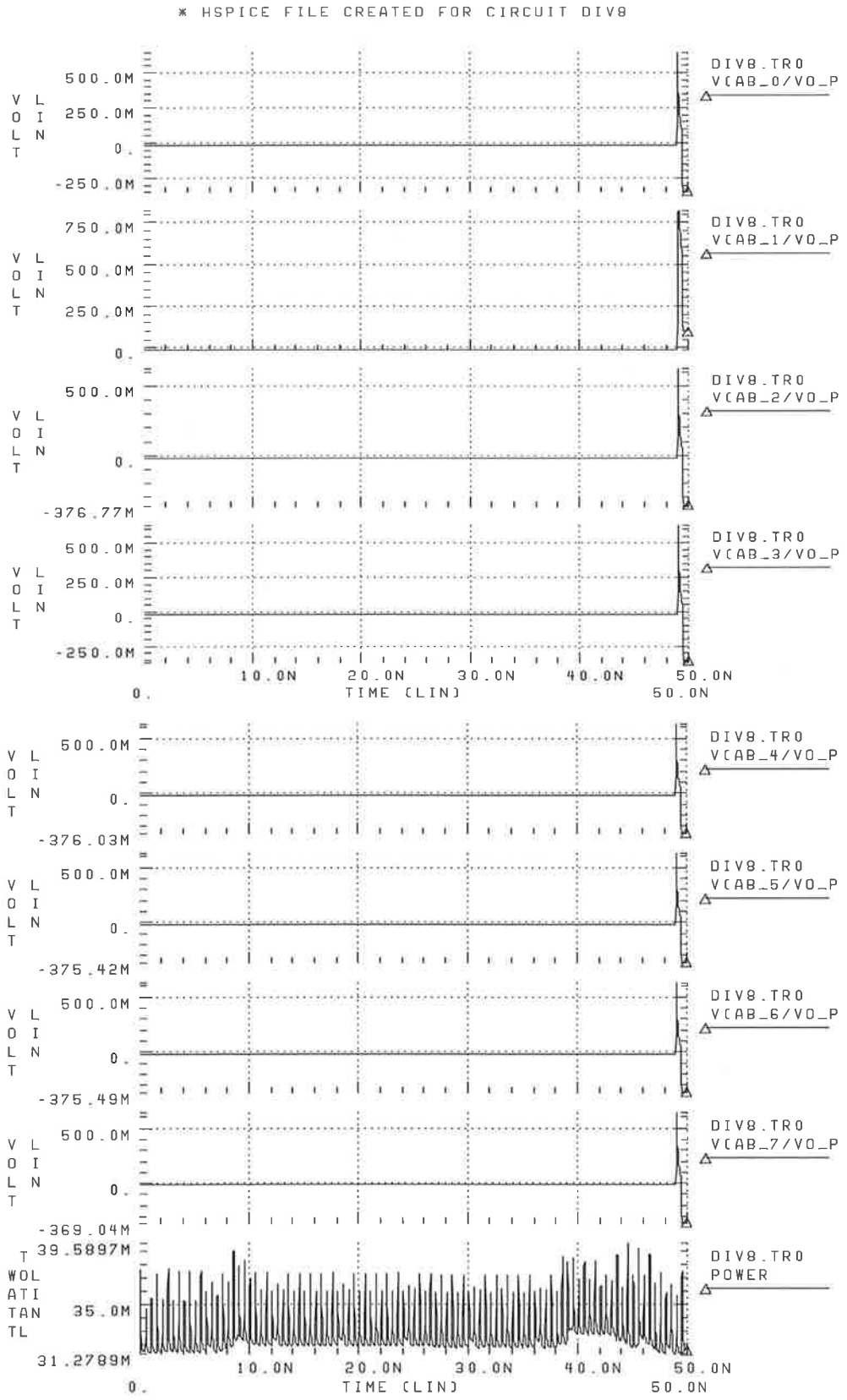


Figure 3.34 The HSPICE simulated results of the 8-bit divider.

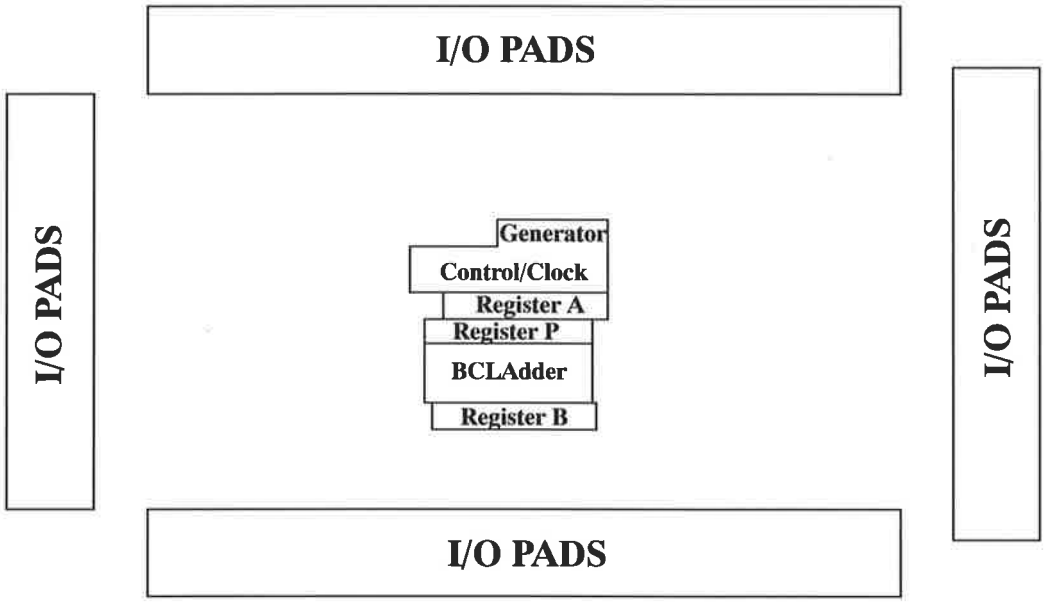


Figure 3.35 The 8-bit divider floor plan.

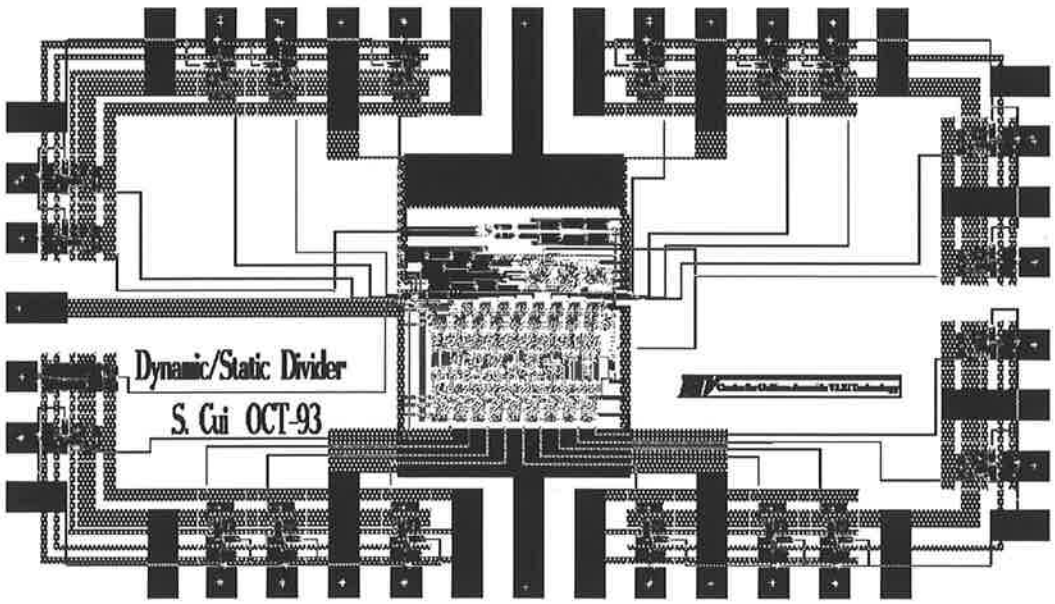


Figure 3.36 The 8-bit divider layout.

3.3.6 Process Spread

From Figure 3.34 it can be seen that the divider works as expected at *typical* process parameters. However, it is good design practice to ensure that a circuit will work for process spreads ranging from *ss2* to *ff2*, corresponding to approximately *95%*

of cases on average. That is, if a manufactured circuit is randomly selected from a large sample, there is *95%* chance that the circuit will work.

The divider was tested over a range of process spreads: *ss2, ss1, tt, ff1, ff2*. It was found that both fast modes performed correctly, but both the slow modes failed. When the slow modes failed, it was the TDFL registers that lost information. The stages appeared to be impaired when it came to discharging an output to ground.

In order to understand the means by which the divider circuit failed for *slow-slow* process spreads, it is necessary to consider the operation of an isolated TDFL shift register stage under these conditions.

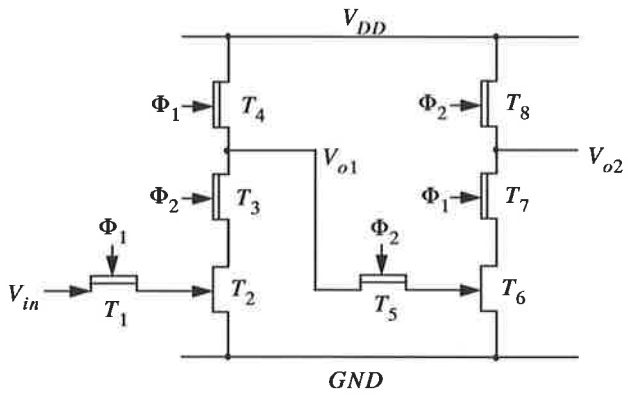


Figure 3.37 Schematic of the TDFL shift register.

Under slow-slow conditions, threshold voltages for both enhancement and depletion devices are increased. For example, for *ss2* at 75°C the *efet* threshold voltage has changed from 0.227V to 0.327V, and the *dfet* threshold voltage has changed from $-0.798V$ to $-0.6V$. Referring to Figure 3.37 on the precharge phase, the node *vo1* will be precharged to V_{dd}, minus the *dfet* threshold voltage, $|V_{td}|$. If the input to the gate of transistor T_2 is a logic high, then the transistor will be turned on. On the evaluate phase, the depletion device T_3 will turn on, but later than normal, due to the higher $|V_{td}|$. Once it is turned on, then the node is discharged to ground through T_2 and T_3. However, in this situation, transistor T_2 is in saturation ($V_{ds} > V_{gs} - V_t$), and so Eq.2.10 is applicable: $I_{dse} = \beta_d \cdot \frac{W_e}{L_e} \cdot (V_{in} - V_{te})^2 \cdot (1 + \lambda_e \cdot V_o) \cdot \tanh(\alpha_e \cdot V_o)$. Since V_{te} has increased, then the saturation current through transistor T_2 to ground will be less. Nearing the end

of the evaluate phase, the depletion transistor T_3 turns off earlier as well. Thus, *slow* transistors have effected the circuit in two ways:

- Effectively shortened the evaluate phase (effect on *dfet*s),
- Impaired ability to sink current (through the *efet*).

The result is that node *vo1* may not able to discharge to the logic low level of the next stage and the circuit will fail. This problem would become worse for less ideal clock signals. The greater the rise and fall times of the clocks, the less current is available to discharge node *vo1*.

The effect of process spread can be modelled in HSPICE by the following formula [27]:

$$V_{TO} = V_{to} + (GAMDS \times V_{ds}) + (K_I(V_{BS})) - (TCV \times \Delta T) \quad \text{(Eq3.61)}$$

where

V_{to} = nominal threshold voltage,

GAMDS = multiplication factor to account for bias dependence,

$K_I(V_{BS})$ = functional relationship for the backgating effect,

TCV = temperature coefficient of threshold voltage.

Since TDFL uses a negative voltage to generate negative clocks, therefore the backgating voltage $V_{BS} = 0.6 - V_{ss} = 0.6 - 1.5 = -0.9V$. This negative backgating voltage together with temperature variation will make *slow* transistors even worse.

There are two ways to solve the problem, one is using lower clock frequency, another is increasing the width of transistors. Clearly, to increase the saturation drain to source current of the *efet* during the discharge of node *vo1*, it is necessary to increase the width of the transistor. It may also be necessary to increase the width of the *dfet* in the pull down path, since this increased current also needs to flow through the *dfet*. Note that as a penalty for increasing transistor width, the parasitic capacitance of each transistor is also increased. This extra capacitance tends to slow down the operation of the circuit, requiring an extra current to be supplied to charge them. The final optimized transistor size of register A is illustrated in Figure 3.38 which has similar sizing to reg-

ister P and B. Note that the transistors in the second inverter are slightly wider than those in the first as they must drive the large load of the parallel outputs. The TDFL registers worked correctly at *ss2* with a clock frequency of *750MHz,* while they can work at up to *1.4 GHz* for *ff2* process parameters.

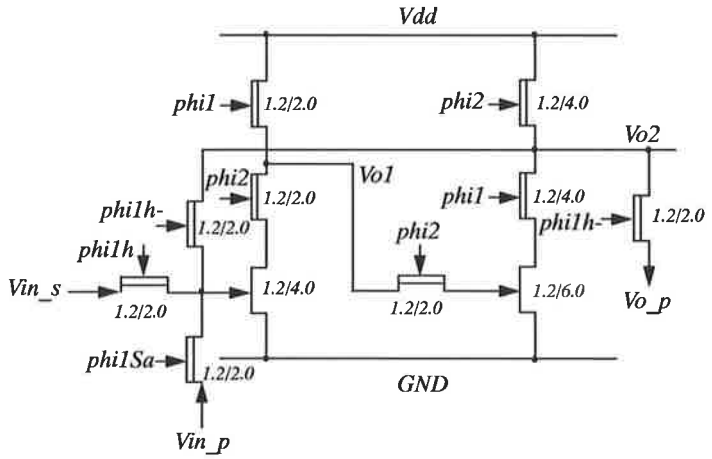


Figure 3.38 The transistor sizes of register A.

3.4 Summary

A variety of dividers and adders were examined, an 8-bit *nonrestoring* divider was implemented based on a mixed dynamic/static approach. The design of the divider had shown that it is possible to accommodate the deficiencies of TDFL, so that the advantages it provides in terms of area saving and low power consumption can be exploited. TDFL is well suited to shift register applications. DCFL allows the implementation of static logic functionality, with the ability to drive moderate loads.

The divider showed to operate at 1GHz clock frequency at *typical typical* processing parameters. The associated power dissipation from V_{DD} at this frequency was 0.35mW on average including the static adder/subtracter part. The work in this chapter showed this mixed dynamic/static approach was very promising for GaAs VLSI circuits. However, the sensitivity of TDFL to variations in process spread and clock skew are of some concern, since they impair the evaluation phase of the circuit. By operating at a lower clock frequency the problem may be circumvented, but further investigation of transistor sizing may yield improved robustness. It is recommended that TDFL only be used localized where process spread and clock skew are under control.

CHAPTER 4

A 32-bit IEEE Floating Point Multiplier

High-speed and high-precision computation are desired for the GaAs Core Processor and many digital signal processing and image processing applications. Floating point computation is most suitable for these applications because it maintains high precision operation over a wide dynamic range.

From Chapter 1, it was seen that a floating point multiplier is the most important element in the GaAs Core Processor. It also requires for single-clock-cycle operation. Furthermore, if a multiplicative divider is used the multiplier will be shared with the divider. In this case, a fast floating point multiplier greatly influences the entire operation of the GaAs Core Processor.

In this chapter, a 32-bit IEEE floating point multiplier is described, which is fast and suitable for implementation in GaAs technology.

4.1 The IEEE Floating Point Standard

4.1.1 Introduction

There are a number of ways to represent nonintergers. One is to use *fixed point* numbers where the number is written as an integer string of digits and the radix point is a function of the interpretation. Addition of two such numbers can be done with an integer addition, whereas multiplication requires some extra shifting. The problem with fixed point arithmetic is the lack of dynamic range. The GaAs Core Processor places the highest demands on computer arithmetic performance, and especially on the large dynamic range of numbers used in Solid Modelling Applications. The dynamic range of a fixed point number system is simply inadequate.

Other representations that have been proposed involve storing the logarithm of a integers (*a, b*) to represent the fraction *a/b* [41]. However, there is only one noninteger representation that has gained widespread use, and that is the *floating point* representation. In this system, a number is divided into two parts, an exponent and a significand. The choice of significand and exponent wordlengths together with sign conversion is entirely arbitrary. There are several different floating point formats [69][71], among them, IEEE standard 754 [72] was an attempt to provide a practical floating point number system that would force floating point calculations performed on different computers to yield the same result. Because of its rapidly increasing acceptance, in this Chapter only IEEE floating point format is discussed.

4.1.2 IEEE Floating Point Format

The IEEE floating point standard specifies the representation of floating point numbers, rounding and exception handling for each of the floating point operations. IEEE floating point numbers consist of three parts: a sign bit, an unsigned exponent with a fixed bias, and a significand which consists of an explicit or implicit leading bit the left of its implied binary point and a fraction field to the right.

The standard specifies four precisions: *single, single extended, double, and double extended.* The properties of these precisions are summarized in Table 4.1. The first row gives the number of bits in the significand. The blank boxes are unspecified parameters. The format for 32-bit single precision numbers is illustrated in Figure 4.1. The value of a normalised number in this format is given by:

$$n = (-1)^s \cdot 2^{e-127} \cdot (1.f) \qquad \text{(Eq4.1)}$$

where s, e and f are the 1-bit sign, the 8-bit exponent biased by 127, and the 23-bit fraction, respectively. $(1.f)$ is the 24-bit *mantissa* consisting of an implied leading "1" and f. So the range of the mantissa is $1 \le (1.f) < 2$. The exponent is a signed number represented using the bias method (as explained in Chapter 3) with a bias of 127. The term called *exponent field* is used to mean the unsigned number contained in bits one through nine and *exponent* to mean the power to which two is to be raised. (In the standard these are called the "biased exponent" and the "unbiased exponent", respectively.) The fraction represents a number less than one, but the *significand* or *mantissa* of the floating point number is one plus the fraction part.

Table 4.1 Format parameters for the IEEE 754 floating point standard.

	Single	Single extended	Double	Double extended
p (bits of precision)	24	≥ 32	53	≥ 64
E_{max}	127	≥ 1023	1023	≥ 16383
E_{min}	-126	≤ –1022	-1022	≤ –16382
Exponent bias	127		1023	

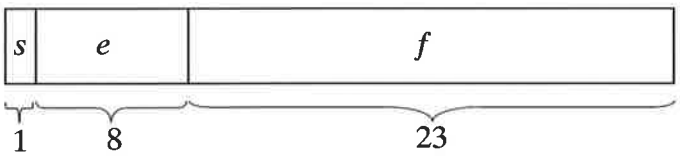


Figure 4.1 Format for IEEE single precision floating point number.

In Table 4.1, the range of exponents for single precision is –126 to 127; accordingly, the exponent field ranges from 1 to 254. The exponent fields of 0 and 255 are used to represent special values. When the exponent is 255, a zero fraction represents infinity, and a nonzero fraction represents a NaN (Not a Number). When the exponent field and fraction are zero, then the number represented is zero. Because ordinary numbers always have a significand greater than or equal to 1—a special conversion such as this is required to represent zero. In IEEE standard, numbers less than $1.0 \times 2^{e_{min}}$ are represented by shifting their fraction part to the right (hidden bit = 0). This is called *gradual underflow*. Thus, as numbers decrease in magnitude below $2^{e_{min}}$, they gradu-

ally lose their significance and are only represented by zero when all their significance has been shifted out.

4.1.3 IEEE Rounding Modes

Another feature of the IEEE standard with implications for hardware is the rounding rule. IEEE standard 754 stipulates four rounding modes, which are *round to nearest*, *round toward 0*, *round toward* $+\infty$ and *round toward* $-\infty$. The default is *round to nearest*, which rounds to an even number in the case of ties. For example, in a floating point system using base 10 and two significant digits, $4.1 \times 0.5 = 2.05$. The result 2.05 should round to 2.0, not 2.1.

4.1.4 Floating Point Multiplication

Floating point multiplication is much like integer multiplication. Because floating point numbers are stored in sign-magnitude form, the multiplier needs only deal with unsigned numbers (although later it can be seen that Booth recoding handles signed two's complement numbers easily).

According to Eq.4.1, the values of a multiplicand X (normalized number) and a multiplier Y are described as:

$$X = (-1)^{s_x} \cdot 2^{e_x - 127} \cdot (1.f_x),$$

$$Y = (-1)^{s_y} \cdot 2^{e_y - 127} \cdot (1.f_y).$$

Then the product $P = X \cdot Y$ becomes

$$P = (-1)^{s_x \oplus s_y} \cdot 2^{(e_x + e_y - 127 + a) - 127} \cdot [(1.f_x)(1.f_y)] \cdot 2^{-a}$$

where "a" results from post-normalization:

$$a = 0 \qquad 0 \le (1.f_x) \cdot (1.f_y) < 2,$$

$$a = 1 \qquad 2 \le (1.f_x) \cdot (1.f_y) < 4.$$

If the fractions are unsigned n-bit numbers, then the production can have as many as $2n$ bits and must be rounded to a n-bit number. *P's* mantissa of $2n$ bits can be divided

into the most significant *n+1* bits and the least significant *n-1* bits. The least significant *n-1* bits act as information for rounding and are represented by *round bit R* and *sticky bit S*. The *R* becomes the *msb* of the least significant *n-1* bits, and *S* is the boolean *or* value of the least significant *n-2* bits.

Besides multiplying the fraction parts, the exponent fields must be added, and bias then subtracted from their sum. However, detecting overflow and underflow is slightly tricky. Consider the case of single precision. The exponent fields must be added together with −127. If the addition is done in a 10-bit adder, −127 = 1110000001_2, and overflow occurs when the high-order bits of the sum are 01 or if the sum is 0011111111. Underflow occurs when the high-order bits are 11 or the sum is 0000000000. Alternatively, the addition can be done using only an 8-bit adder. Simply add both exponents and −127 = 10000001_2. If the high-order bits of the exponent fields are different, no over/underflow is possible. If the high-order bits are both 1, the result has overflowed if it has 0 in the high-order bit or if it is 11111111. If both the exponents have high-order bits of zero, underflow has occurred if the sum has a high-order bit of 1, or if the sum is 00000000.

Since the mantissa typically have much longer wordlengths than the exponents, the main constraint on the speed of the floating point multiplier will be computation of the product's mantissa which consists of an integer multiplier array and a final addition and rounding stage. The following sections will discuss these two important stages in detail.

4.2 Integer Multiplication Algorithms

The most basic form of multiplication consists of forming the product of two positive binary numbers. This may be accomplished through the traditional "*grade-school*" multiplication algorithm also called "*shift and add*" algorithm. Figure 4.2 illustrates the computation of the $2n$-bit product $P = (p_{2n-1}, p_{2n-2}, ..., p_0)$ of two n-bit numbers $X = (x_{n-1}, x_{n-2}, ..., x_0)$ and $Y = (y_{n-1}, y_{n-2}, ..., y_0)$. To perform the multiplication, the bits of Y, from y_0 up to y_{n-1}, need to be examined. For each bit y_i with a value of 1, X is added into the product, but shifted left by i positions. For each bit y_i with a

value of 0, 0 is added into the product. Thus, letting $pp^{(i)} = x \cdot y_i \cdot 2^i$, the product will be:

$$P = X \cdot Y = \sum_{i=0}^{n-1} pp^{(i)} . \qquad \text{(Eq4.2)}$$

where each term $pp^{(i)}$ is called a *partial product*. There are n partial products to sum, with bits in positions 0 to $2n - 2$. The carry-out from the highest bit yields the final bit in position $2n - 1$.

				1	1	1	0	=	X
				1	1	0	1	=	Y
				1	1	1	0	=	$pp^{(0)}$
			0	0	0	0		=	$pp^{(1)}$
		1	1	1	0			=	$pp^{(2)}$
	1	1	1	0				=	$pp^{(3)}$
1	0	1	1	0	1	1	0	=	P

Figure 4.2 The "grade-school" multiplication method.

There are a number of techniques that may be used to perform multiplication, e.g. serial form, serial/parallel form, parallel form or array form. In general, the choice is based on factors such as speed, throughput, numerical accuracy, and area. A variety of multiplier architectures are available for GaAs implementation [83]. The speed advantage of GaAs is best exploited by array architectures [56]. In this section the most widely-used array multiplication techniques will be presented, starting with a simple linear-time array and then develop the multiplier design to cover the modified Booth's algorithm, carry save array, modified carry save array and Wallace tree methods. The section will conclude with examination of speed and circuitry needed for IEEE standard floating point number multiplication.

4.2.1 Simple array multiplier

The simplest VLSI multiplier is the shift-and add form which consists of an array of cells comprising an *and* gate and a full adder as illustrated in Figure 4.3. The final

product is the sum of a number of partial products as shown in Eq.4.2, each partial products is formed on individual rows of the multiplier.

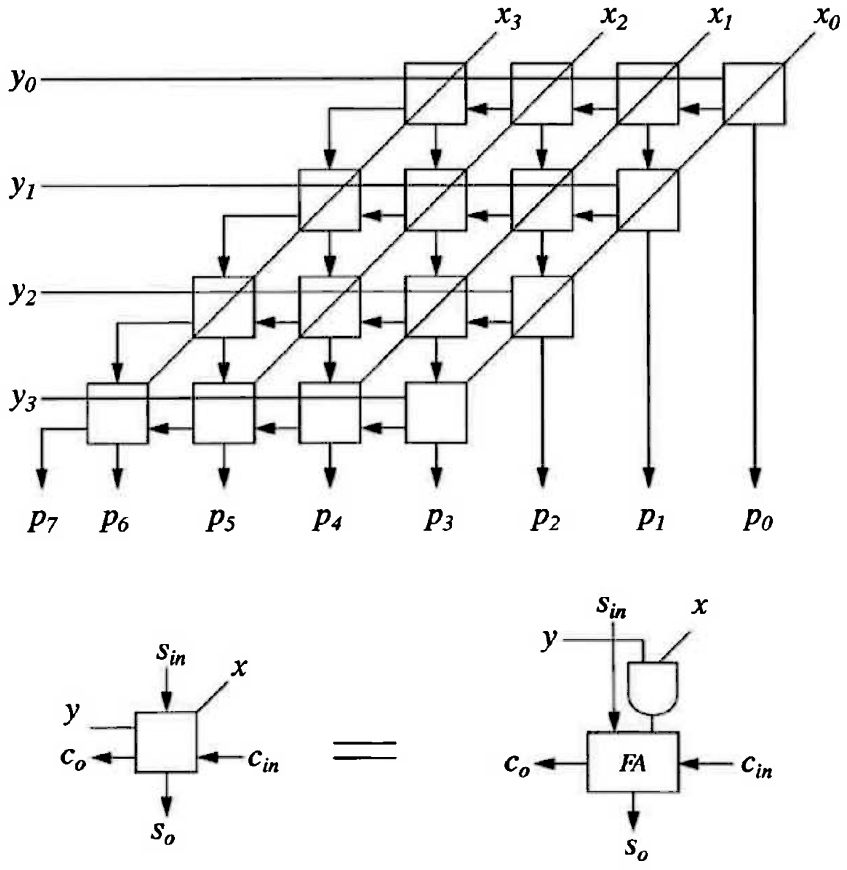


Figure 4.3 A simple array multiplier architecture.

Figure 4.3 illustrates a 4 × 4 array multiplier. An $n \times n$ multiplier requires n^2 full adders and *and* gates. The critical path through the multiplier, and hence the worst case delay is $2n$ cell's delay.

4.2.2 The carry save multiplier

The speed of the simple array multiplier may be increased considerably by means of minor alteration to the cell's interconnection: if the carry signals were to propagate diagonally through the array rather than horizontally, the implicit ripple carry adders embedded in the simple array multiplier could be replaced by an explicit accelerated carry adder at the foot of the array as illustrated in Figure 4.4.

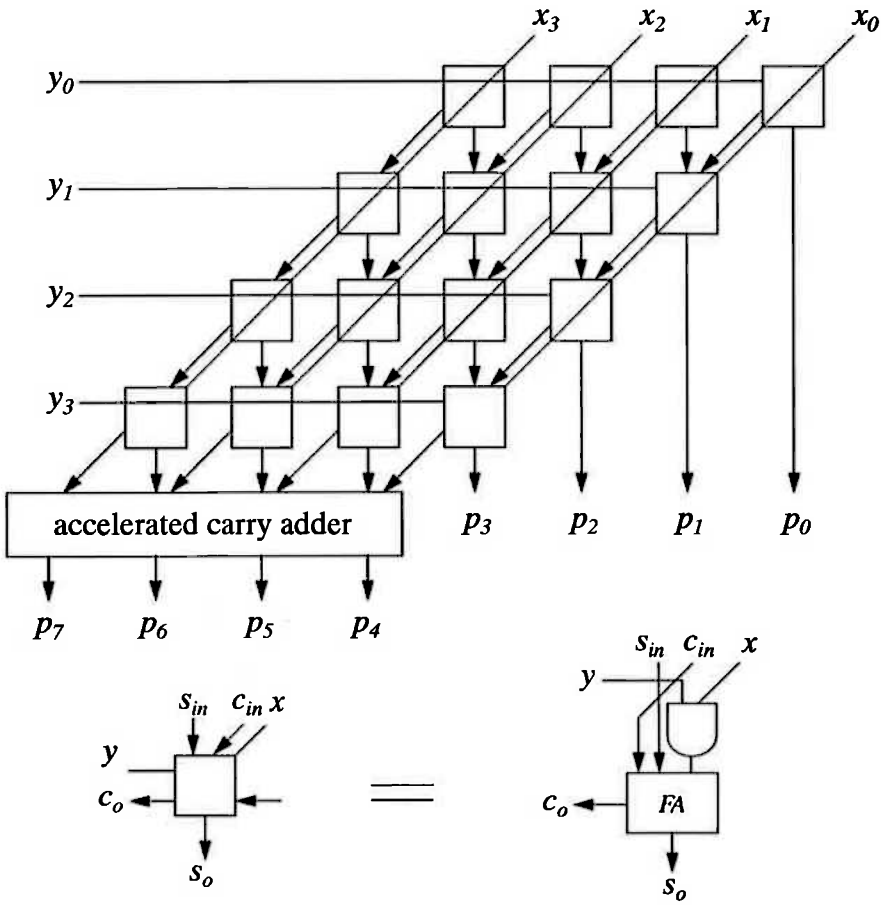


Figure 4.4 Carry save multiplier architecture.

An $n \times n$ carry save multiplier requires n^2 cells (each cell consists of one full adder and one *and* gate), plus an accelerated carry adder. The accelerated carry adder could be any accelerated carry adder like a carry look-ahead adder, carry select adder, or carry-skip adder. The multiplication time of the carry save multiplier is n cell's delay plus the accelerated carry adder delay.

4.2.3 Radix-4 Booth's algorithm

The structures shown in Figures 4.3 and 4.4 compute the partial products in a *radix-2* manner, that is by observing one bit of the multiplicand at a time. Higher radix multipliers may be designed to reduce the number of adders and hence the delay required to compute the partial sums. The best know method is called *Booth recoding* or *Booth's algorithm*, which is a *radix-4* multiplication scheme.

Booth's algorithm may be understood by considering a two's complement number. A useful formula for the value of a two's complement number $Y = (y_{n-1}, y_{n-2}, \ldots, y_0)$ is [41]:

$$Y = -y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + \ldots + y_1 2^1 + y_0 \qquad \text{(Eq4.3)}$$

By adding a dummy term y_{-1} (equal to 0), Eq.4.3 can be rewritten as:

$$\begin{aligned} Y = {} & -y_{n-1}2^{n-1} + \left(y_{n-2}2^{n-1} - y_{n-2}2^{n-2}\right) + \left(y_{n-3}2^{n-2} - y_{n-3}2^{n-3}\right) + \\ & \ldots + \left(y_1 2^2 - y_1 2^1\right) + \left(y_0 2^1 - y_0 2^0\right) + y_{-1}2^0 \end{aligned} \qquad \text{(Eq4.4)}$$

Rearranging Eq.4.4, the word Y will be:

$$\begin{aligned} Y = {} & (y_{n-2} - y_{n-1})\,2^{n-1} + (y_{n-3} - y_{n-2})\,2^{n-2} + \ldots + (y_0 - y_1)\,2^1 + (y_{-1} - y_0)\,2^0 \\ = {} & \sum_{i=1}^{n} (y_{n-i-1} - y_{n-i})\,2^{n-i} = \sum_{i=1}^{n} \alpha_{n-i}2^{n-i} \end{aligned} \qquad \text{(Eq4.5)}$$

where α_{n-i} can be any of the values $\{-1, 0, 1\}$. Therefore, the product P of X and Y can be written as:

$$P = \sum_{i=1}^{n} \alpha_{n-i}X2^{n-i} = \sum_{i=1}^{n} pp^{(i)}, \qquad \text{(Eq4.6)}$$

and the partial products $pp^{(i)} = \alpha_{n-i}X2^{n-i}$ have to be computed in relationship with the value of α_{n-i}.

This algorithm was modified by Mac Sorley [87] in following way. The number Y from Eq.4.3 can be rewritten as:

$$\begin{aligned} Y = {} & -y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + \left(y_{n-3}2^{n-2} - y_{n-3}2^{n-3}\right) + y_{n-4}2^{n-4} + \\ & \left(y_{n-5}2^{n-4} - y_{n-5}2^{n-5}\right) + y_{n-6}2^{n-6} + \left(y_{n-7}2^{n-6} - y_{n-7}2^{n-7}\right) + \\ & \ldots + \left(y_1 2^2 - y_1 2^1\right) + y_0 2^0 + y_{-1}2^0 \end{aligned} \qquad \text{(Eq4.7)}$$

where is y_{-1} once again a zero dummy variable.

Eq.4.7 can be further presented in the following form:

$$Y = \left(-y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + y_{n-3}2^{n-2}\right) + \left(-y_{n-3}2^{n-3} + y_{n-4}2^{n-4} + y_{n-5}2^{n-4}\right) + \left(-y_{n-5}2^{n-5} + y_{n-6}2^{n-6} + y_{n-7}2^{n-6}\right) + \ldots + \left(-y_1 2^1 + y_0 2^0 + y_{-1}2^0\right)$$

Factorising all the powers of 2 yields:

$$Y = (y_{n-3} + y_{n-2} - 2y_{n-1})\,2^{n-2} + (y_{n-5} + y_{n-4} - 2y_{n-3})\,2^{n-4} + (y_{n-7} + y_{n-6} - 2y_{n-5})\,2^{n-6} + \ldots + (y_{-1} + y_0 - 2y_1)\,2^0 \qquad \text{(Eq4.8)}$$

Thus, the number Y can be expressed as:

$$Y = \sum_{\substack{i=1 \\ odd}}^{n-1} (y_{n-i-2} + y_{n-i-1} - 2y_{n-i})\,2^{n-i-1} = \sum_{\substack{i=1 \\ odd}}^{n-1} \alpha_i 2^{n-i-1}, \qquad \text{(Eq4.9)}$$

where α_i can be any of the values {-2, −1, 0, 1, 2} and i is odd. Eq.4.9 is called modified Booth's algorithm.

A modified Booth's algorithm examines three bits of the multiplier $Y = (y_{n-1}, y_{n-2}, \ldots, y_0)$ at a time to determine whether to add $\frac{n}{2}$ sum of 0, 1, −1, 2 or −2 of that rank of the multiplicand. Hence, the number of digits in the multiplier word is halved compared with the original Booth recoding Eq.4.5. Consequently, the number of partial products is reduced by a factor of 2. However the value range of the coefficients α_i has almost doubled (from 3 to 5) at the cost of increased complexity of encoders and multiplexers.

In summary, the steps in the modified Booth's algorithm are as follows [88]:

1. Append a zero y_{-1} to the least significant bit of the multiplier *Y*.
2. Examine the three least significant bits of the multiplier *Y* (called *triplets*) and decode the multiplicand *X* according to Table 4.2.
3. Add or subtract the resulting multiple of the multiplicand from the previous partial product to form the new partial product.

4. Shift the new partial product and the multiplier Y right two places arithmetically, that is, the bit shifted into the high order bit of product P should be the sign bit of P.

5. Repeat above operations $n/2$ times (n ≥ 0 and even). After the final addition or subtraction, the partial product is shifted right one place to form the final product.

Table 4.2 Modified Booth's algorithm of bit triplets.

Multiplier Bits			Operation
y_{i+1}	y_i	y_{i-1}	
0	0	0	+0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	−0

Example:

Consider 16 × 16-bit modified Booth's algorithm multiplier. The multiplicand X = 14 = 0000 0000 0000 1110, the multiplier Y = −157 = 1111 1111 0110 0011. Appending a zero to the least significant bit of the multiplier Y yields:

Y = 1111 1111 0110 0011.0

Then examining the triplets and decoding the multiplicand X according to Table 4.2:

$Y = \overline{1}\underline{11}\,\overline{1}\underline{11}\,\overline{0}\underline{11}\,\overline{0}\underline{01}\overline{1}.\underline{0}$

110 ⇒ first partial product = −1 = 1111 1111 1111 0010

001 ⇒ second partial product = +1 = 0000 0000 0000 1110

100 ⇒ third partial product = −2 = 1111 1111 1110 0100

011 ⇒ fourth partial product = +2 = 0000 0000 0001 1100

110 ⇒ fifth partial product = −1 = 1111 1111 1111 0010

111 ⇒ sixth partial product = –0 = 0000 0000 0000 0000

111 ⇒ seventh partial product = –0 = 0000 0000 0000 0000

111 ⇒ eightth partial product = –0 = 0000 0000 0000 0000

Shifting the new partial product right two places arithmetically, and adding the partial products give:

					–X	1←	1←	1	1	1	1	0	0	1	0
					+X	0	0	0	0	1	1	1	0		
				0 ←	0 ←	0	0	0	0	1	0	1	0	1	0
			–2X	1	1	1	0	0	1	0	0				
		1←	1←	1	1	1	0	0	1	1	0	1	0	1	0
	+2X	0	0	0	1	1	1	0	0						
0 ←	0 ←	0	0	0	1	0	1	0	1	1	0	1	0	1	0
–X	1	1	1	1	0	0	1	0							
P	1	1	1	1	0	1	1	1	0	1	1	0	1	0	1	0

where P = 111 0111 0110 1010 is equivalent to the decimal number –2198, which is the correct result of (14) × (–157).

There are two ways of designing the partial product generation circuits in a modified Booth's algorithm multiplier [95]. One is to use three-to-one multiplexer to manipulated (shifted, inverted etc.) the multiplicand directly. Method 1 requires more logic to produce the partial product bits, but simpler control signals and circuitry. In fact, the control circuits are related since the "direct" control circuit may be used to produce the multiplexer select signals. Figure 4.5 illustrates a typical partial product generator circuit for the second method and Figure 4.6 illustrates a typical control circuit that derives *shift*, *invert* and *zero* signals according to Table 4.3.

Table 4.3 Booth's recoding of bit triplets for method 1.

Y_{i+1}	Y_i	Y_{i-1}	S_1	S_2	S_3	Operation	*pp*
0	0	0	0	1	0	+0	0
0	0	1	0	0	1	+1	X_i
0	1	0	0	0	1	+1	X_i
0	1	1	0	0	0	+2	X_{i-1}
1	0	0	1	0	0	−2	$\overline{X}_{i-1}$
1	0	1	1	0	1	−1	$\overline{X}_i$
1	1	0	1	0	1	−1	$\overline{X}_i$
1	1	1	1	1	0	−0	0

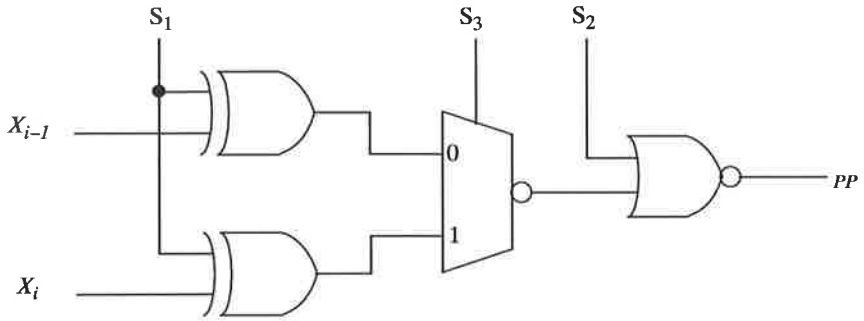


Figure 4.5 Partial product bit generator for method 1.

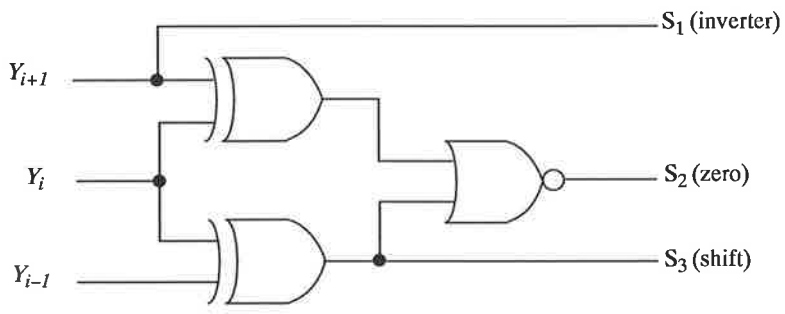


Figure 4.6 "Direct" control circuit for a modified Booth's algorithm coder.

Since each *shift*, *invert* and *zero* signals would drive nearly double loads compared to the first method, heavier buffering is needed and the speed would suffer.

The other way to generate the partial products in a modified Booth's algorithm multiplier is to employ a five-to-one multiplexer for each partial product bit—the five data inputs to the multiplexer would be bit i of 0, X and 2X, derived in advance from the multiplicand word, and the multiplexer's select signals would be derived from the multiplier bit triplets according to Booth encoders.

The Booth encoders examine the bits in the multiplier and activate the appropriate lines in the complement/select (five-to-one multiplexer) logic circuitry. The latter subsequently produces the inputs to the full adders so that the proper partial product terms are evaluated. The function of each encoder is defined in Table 4.4.

Table 4.4 The functions of the encoders of the five-to-one multiplexer method.

Multiplier bits			Operation	Partial product (*pp*) Selectors				
Y_{i+1}	Y_i	Y_{i-1}		S_1	S_2	S_3	S_4	S_5
1	0	1	−1	0	1	1	1	1
1	1	0	−1	0	1	1	1	1
0	0	1	+1	1	0	1	1	1
0	1	0	+1	1	0	1	1	1
0	0	0	0	1	1	0	1	1
1	1	1	0	1	1	0	1	1
0	1	1	+2	1	1	1	0	1
1	0	0	−2	1	1	1	1	0

From Table 4.4 the following equations can be obtained:

$$S_1 = \overline{\left(\overline{\overline{y_{i-1} + y_i + y_{i+1}}}\right) + \left(\overline{y_{i-1} + \overline{y_i} + \overline{y_{i+1}}}\right)}, \quad \text{(Eq4.10)}$$

$$S_2 = \overline{\left(\overline{\overline{y_{i-1} + y_i + y_{i+1}}}\right) + \left(\overline{y_{i-1} + \overline{y_i} + y_{i+1}}\right)}, \quad \text{(Eq4.11)}$$

$$S_3 = \overline{\left(\overline{y_{i-1} + y_i + y_{i+1}}\right) + \left(\overline{\overline{y_{i-1} + \overline{y_i} + \overline{y_{i+1}}}}\right)}, \quad \text{(Eq4.12)}$$

$$S_4 = \overline{\left(\overline{\overline{y_{i-1} + \overline{y_i} + y_{i+1}}}\right)}, \quad \text{(Eq4.13)}$$

$$S_5 = \overline{\left(\overline{y_{i-1} + y_i + \overline{y_{i+1}}}\right)}. \quad \text{(Eq4.14)}$$

Figure 4.7 illustrates the logic diagram of the encoder implemented by *nor* gates and *inverter*s. The last stages of the encoder (thick *nor* gates and *inverter*s) are SDCFL gates which act as buffers to drive large loads.

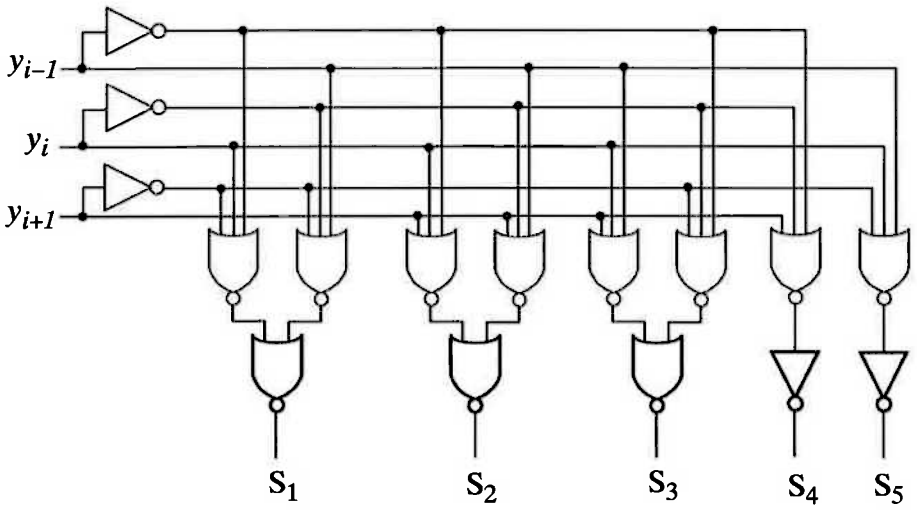


Figure 4.7 Logic diagram of the encoder circuit.

Table 4.5 shows the function of the complement/select (five-to-one multiplexer) logic. It uses the output of the encoder to select the appropriate bits of the multiplicand so as to form the partial products. The output of the multiplexer is directly connected to the input of the full adders. The logic diagram of the multiplexer is illustrated in Figure 4.8.

Table 4.5 The functions of the five-to-one multiplexer.

Inputs						Operation	Output
S_1	S_2	S_3	S_4	S_5	multiplicand bits		(*pp*)
0	1	1	1	1	$x_i, \bar{x}_i, x_{i+1}, \bar{x}_{i+1}$	−1	$\bar{x}_{i+1}$
1	0	1	1	1	$x_i, \bar{x}_i, x_{i+1}, \bar{x}_{i+1}$	+1	x_{i+1}
1	1	0	1	1	$x_i, \bar{x}_i, x_{i+1}, \bar{x}_{i+1}$	0	0
1	1	1	0	1	$x_i, \bar{x}_i, x_{i+1}, \bar{x}_{i+1}$	+2	x_i
1	1	1	1	0	$x_i, \bar{x}_i, x_{i+1}, \bar{x}_{i+1}$	−2	$\bar{x}_i$

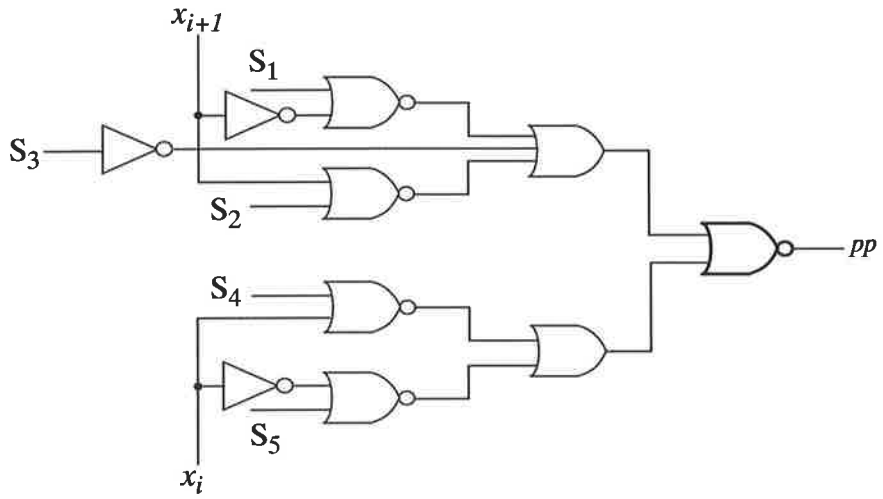


Figure 4.8 Logic diagram of the five-to-one multiplexer.

Noting that in two's complement representation a "1" must also be added to the partial product for –1 and at a "10_2" or two "1" has to be added for –2, for +2 a "0" has to be appended to the partial product. Thus the least significant bit multiplexer is connected with $x_{i+1} = x_0$, $x_i = 0$ and the output of $S_1 \oplus S_5$ is fed to the inputs of the least significant bit full adder. In this way, when $S_1 = 0$, $S_5 = 1$ (–1 case), a "1" from the output of $S_1 \oplus S_5$ is added to the least significant bit of partial product $pp_0 = \bar{x}_0$; when $S_1 = 1$, $S_5 = 0$ (–2 case), a "1" from the output of $S_1 \oplus S_5$ is added to the least significant bit partial product $pp_0 = 1$. Figure 4.9 illustrates a overall schematic of a 4 × 4-bit carry save multiplier with modified Booth's recoding. It can be seen that the 4 partial products are reduced to 2. In this case a ripple carry adder is used as final adder. To further speed up the multiplier, other accelerated carry propagate adders can be used.

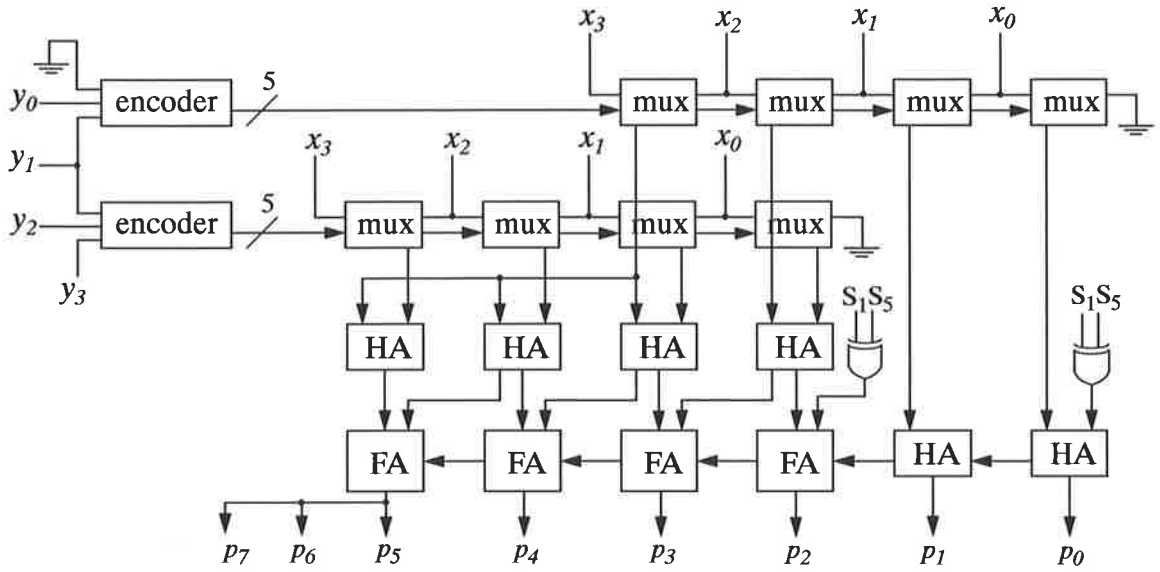


Figure 4.9 The overall schematic of a 4 × 4-bit modified Booth's multiplier.

4.2.4 The Wallace tree multipliers

The *Wallace tree* method multipliers use circuits called *counter*s or *compressor*s to reduce the partial product tree height from n to 2. The final result is then computed by using a carry-lookahead adder or other fast adders mentioned previously on the two remaining partial product rows. A (n, m) compressor is a combinational logic circuit with n inputs and m outputs. The outputs are binary encoding of the sum of the input bits. Such tree architectures are known as Wallace trees (if $(3, 2)$ compressors only are used) or Dadda trees (if some combination of $(3, 2)$, $(4, 2)$, $(2, 2)$, $(5, 3)$, and $(7, 3)$ compressors are used).

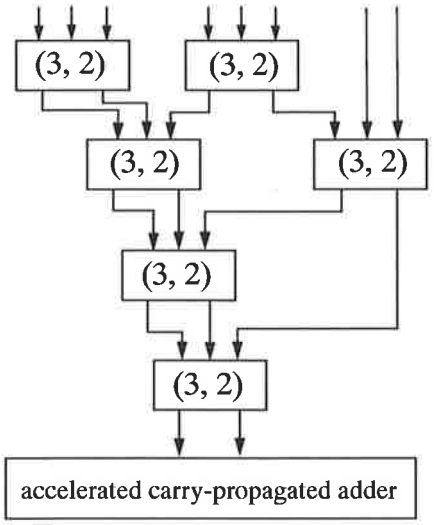


Figure 4.10 A Wallace tree for n = 8 partial products.

Among these methods, Wallace tree reduction is the fastest [90]. Figure 4.10 shows a Wallace tree that adds 8 partial product products. It takes 4 (3, 2) compressors (full adder) to reduce 8 partial product products to 2, then an accelerated carry-propagate adder is used to add the remainder 2 partial products. In general, the number of stages required to reduce n partial products to 2 is:

$$T = \log_{1.5}(n/2) = 2.4664 \ln (n/2) . \qquad \text{(Eq4.15)}$$

For 8 partial product products, n = 8, T ≈ 3.4. Therefore, at least 4 stages are needed as shown in Figure 4.10. For a 25-bit multiplier with modified Booth's recoding, the partial product tree height will be 13. Putting n = 13 to Eq.4.15, $T \approx 4.62$ can be obtained. This means that for IEEE single precision format with modified Booth's recoding and Wallace tree reduction, only 5 stages i.e. 5 full adders deep are needed, while using an array multiplier with modified Booth's encoding, it needs 13 stages. However, the total delay depends not only on the number of stages T, but also on the delay associated with the interconnection wiring capacitance. It can be seen from Figure 4.10, a Wallace tree is not as regular as an array multiplier, the delay of the interconnection is high and it is not suitable for VLSI implementation.

The irregular layout can be improved by using other reduction schemes such as (4, 2) and (7, 3) compressors. Figure 4.11 illustrates a 25×25-bit modified Booth multiplier using (4, 2) compressors. The (4, 2) compressor is formed from two (3, 2) i.e. full adder cells. It can be seen that it needs 3 stages i.e. it is 6 full adders deep. However, the layout regularity is still compromised, which is a disadvantage in high-speed GaAs technology. Figure 4.11 illustrates the connections of such (4, 2) compressor tree.

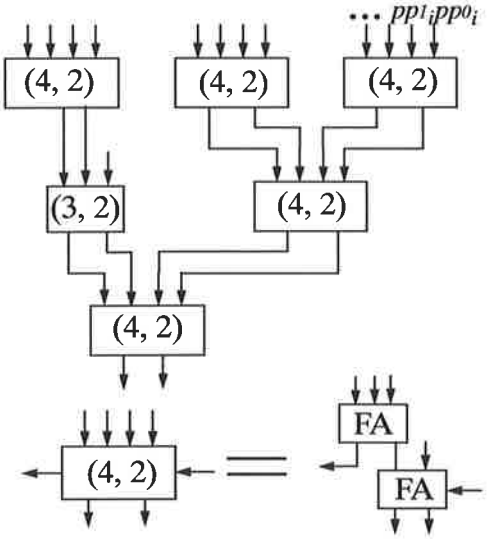


Figure 4.11 A 25×25-bit multiplier using modified Booth and (4, 2) compressors

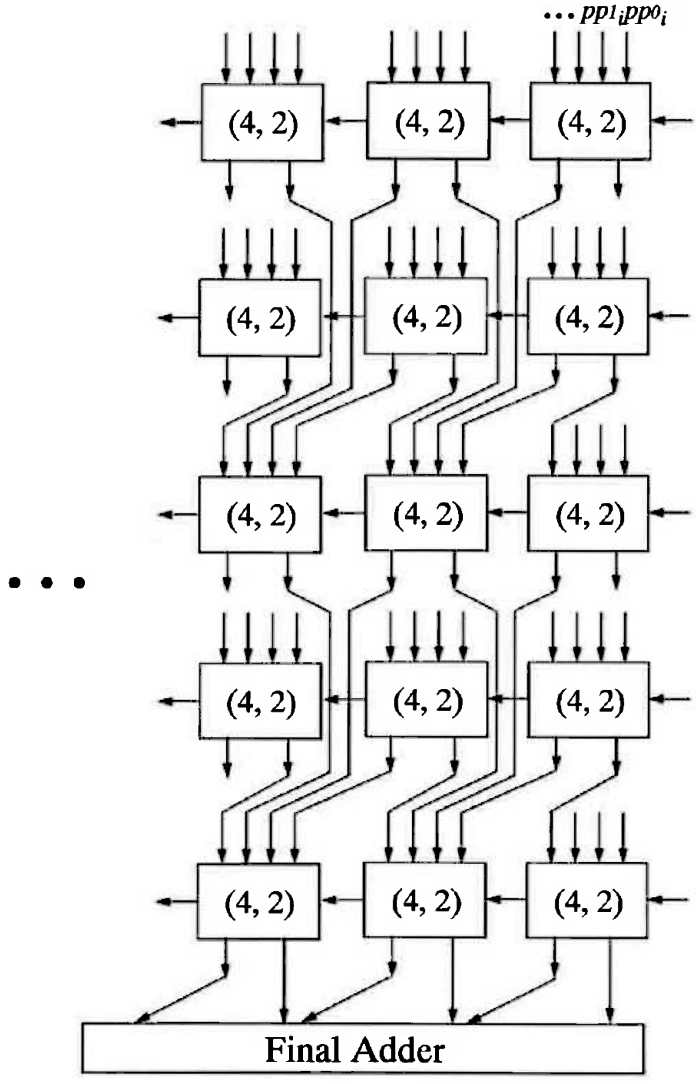


Figure 4.12 The connections of the (4, 2) compressor multiplier.

4.3 Rounding Algorithms

Multiplication of $n \times n$-bits will produce a $2n$-bit product and a significant part of the delay in a floating point multiplier may be caused by rounding the product in accordance with the IEEE standard. As mentioned earlier, the IEEE standard 754 default rounding mode is *round to nearest*. In this chapter only this default rounding mode is considered.

Rounding to nearest as defined in IEEE 754 is actually *round to nearest/even*. This means always round to nearest, and in the case of a tie round to even. A convention rounding system, *round to nearest/up*, adds 1/2 to the least significant bit (*lsb*) of the desired result and then truncates by removing the bits to the right of the *lsb*. Rounding to nearest/up produces exactly the same result as round to nearest/even in all cases except when a tie occurs. If the even result were the smaller value, round to nearest/up would incorrectly round up. Dealing with the tie case before rounding makes round to nearest/even more complex and slower than round to nearest/up. Thus, this section will discuss a general rounding algorithm to produce round to nearest/up result [94], then using the *sticky bit* to get the correct round to nearest/even result.

4.3.1 A simple round to nearest/up algorithm

Supposing A and B are n-bit mantissae of two IEEE single precision numbers. After using array or tree reduction multipliers, $A \times B$ will become the sum of two $2n$-bit partial products in carry save form as shown in Figure 4.13. These two numbers are then added in the final addition section to produce a complete $2n$-bit product. There are two possible rounding operations, depending on the most significant bit (*msb*) of this product. If the resulting product is in the range $2 \leq$ product < 4, which means an *overflow* occurs, the constant $2^{(-n+1)}$ is added to the product and the result is truncated to $n-2$ bits to the right of the decimal point. A *normalization* shift of 1 to the right is then necessary to restore the rounded product to the range $1 \leq$ rounded product < 2, with an appropriate adjustment of the exponent. If the original $2n$-bit product is in the range $1 \leq$ product < 2, which means an no *overflow* occurs, the constant $2^{(-n)}$ is added to the product. In most cases this rounded product will be less than 2 and the rounding operation is finished. However, it is possible that the addition of $2^{(-n)}$ could cause the rounded product to be equal to 2 which means overflow, in this case a normalization shift of 1 bit and an exponent adjustment is necessary.

It can be observed from Figure 4.13 that this rounding algorithm is simple, but requires two carry propagate additions in series. In order to significantly increase the performance, these additions need to be computed in parallel.

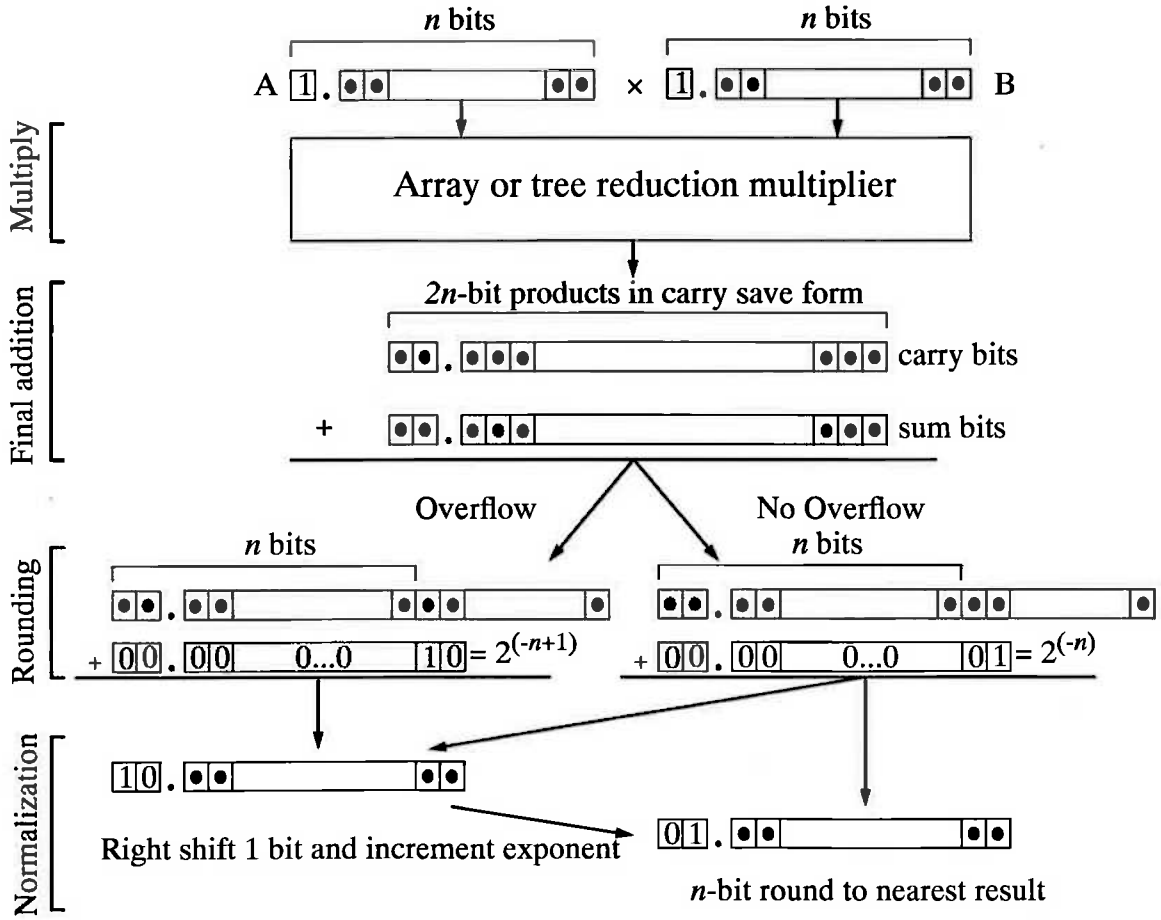


Figure 4.13 A simple round to nearest/up algorithm.

4.3.2 A parallel rounding algorithm

If an n+2 bits carry propagate adder is used in the **Final addition** section of Figure 4.13, then the carry from the low order n-2 bits (C_{in}) will be added at the $2^{(-n)}$ position. In the case of no overflow, an additional $2^{(-n)}$ will be added to the result in the **Rounding** section. Thus, this $2^{(-n)}$ position is called the *rounding bit* position or R bit position. A "1" is always added to the R bit position for rounding, and called R_{in}. If no overflow occurs, adding C_{in} and R_{in} to the R bit position will produce the correct round to nearest/up result.

In the case of overflow, the *msb* of the product, known as the overflow bit (V), is a 1. Then $2^{(-n-1)}$ needs to be added for rounding. This can be done by adding another

$2^{(-n)}$ to the R bit position. This new $2^{(-n)}$ is defined as the *overflow rounding* bit (R_v). Thus, the correct rounding can be obtained by simply adding the carry from the lower order bits (C_{in}), the rounding bit (R_{in}), and the overflow rounding bit (R_v), to the R bit position as illustrated in Figure 4.14.

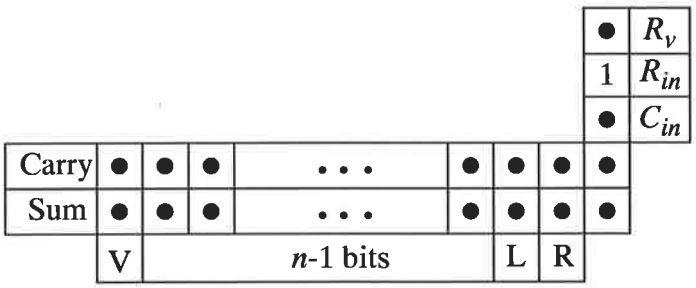


Figure 4.14 Bits to be added for correct round to nearest/up.

It can be seen that there are five slots to be added at *lsb* and R_v is not known until the sum of all of the other bits have been computed. Santoro [94] proposed two algorithms to solve these problems. One is using a row of $n+2$ bit half adders and an empty slot in the array/tree multipliers; the other one is using carry select adders at the two least significant bits instead of using two half adders. The former is useless if no empty slots are left in array/tree multiplier, the latter is a little bit complicated.

Here a modified version is proposed which uses the same idea as Santoro [94] but is much simpler. This algorithm is illustrated in Figure 4.15. One full adder at *lsb* is used to provide a slot for R_{in}. This full adder together with a row of $n+1$ half adders are used to partly sum the carry and sum bits. This leaves a hole in the carry propagate increment adder at the R bit. The C_{in} from the lower order bits can be placed into this hole. A carry select adder is used, one adding 0 representing $R_v = 0$, the other one adding 1 for $R_v = 1$. Once R_{in} and C_{in} have been added to the R bit position and the carry select adder has completed, the correct result can be picked based on the overflow bit from result of adding 0. In this case, the result must be normalized and the exponent adjusted.

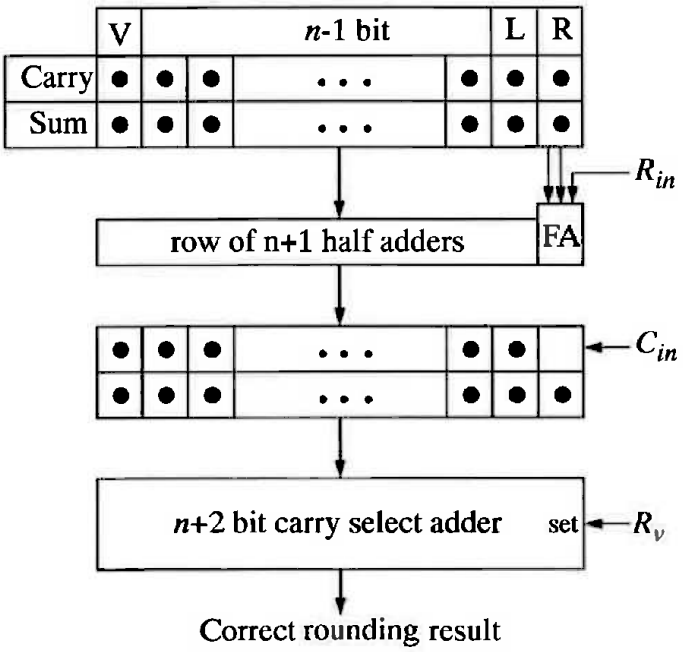


Figure 4.15 A parallel addition and rounding algorithm.

4.3.3 Obtaining round to nearest/even by round to nearest/up

Round to nearest/up produces exactly the same result as round to nearest/even except when a tie happens. A tie can only happen when the result is exactly halfway between two numbers of representable precision. For example:

14.65XXX	Raw number to be rounded
+ 0.05	Add 0.05 to round to nearest/up
14.70	Sum
14.7	Truncated—Final rounded result

If the "X"s are all zeros then 14.65 is exactly half way between 14.6 and 14.7. In this case round to nearest/up produces a different result than round to nearest/even. There are only two cases to be considered. Either all of the "X"s are 0, or they are not. The bit which distinguishes between these cases is referred to as the *sticky* bit. This bit is a 0 if all of the "X"s are 0, and 1 if any of the "X"s is 1.

The difference between correct round to nearest/even and correct round to nearest/up according to the least significant bit L, the rounding bit R, and the sticky bit S is shown in Table 4.6 [94].

Table 4.6 Round to nearest/even versus rounding to nearest/up.

Before Rounding			Adder to R		L After Rounding	
L	R	S	E	U	L_E	L_U
X	0	0	D	1	X	X
X	0	1	D	1	X	X
0	1	0	0	1	0	1
1	1	0	1	1	0	0
X	1	1	1	1	$\overline{X}$	$\overline{X}$

where:

E = Bit added for correct round to nearest/even.

U = Bit added for correct round to nearest/up.

L_E = The L bit after round to nearest/even.

L_U = The L bit after round to nearest/up.

D = Don't care. E can not affect L_E.

It can be seen that the only case where the round to nearest/up bit (U) will produce a different result from the round to nearest/even bit (E) is in row 3 of Table, where $E = 0$, and $U = 1$. In this case round to nearest/up changed the L bit from a 0 ($L = 0$) to a 1 ($L_U = 1$), while round to nearest/even left the L bit unchanged ($L_E = 0$). The should be noticed that when round to nearest/up changed the L bit to a 1, the 1 was not propagated. As such, only the L bit was effected. This means that the correct round to nearest/even result can be obtained from the round to nearest/up result by restoring the L bit to a 0.

By assuming that the round bit will be a 1 ($R_{in} = 1$), the round to nearest/up algorithms have the advantage over the round to nearest/even methods in that the carry propagate addition can take place before the sticky bit has been computed. This means that the round to nearest/up result can be obtained using the rounding algorithms described in previous section. The correct IEEE round to nearest/even result can be obtained by observing only the L, R and *sticky* bits, and forcing the L bit to 0 if required. Figure 4.16 illustrates such a circuit implemented by DCFL gates. The sticky bit is the *or* of all of the bits to the right of the R bit.

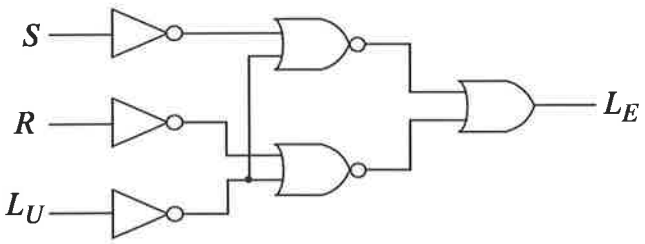


Figure 4.16 The circuit diagram for obtaining round to nearest/even from round to nearest/up.

4.4 A 32-bit IEEE Floating Point Multiplier

The previous sections have described various techniques for floating point multipliers. This section will use the best of these techniques to present a complete 32-bit IEEE floating point multiplier. A modified carry save array is used in conjunction with modified Booth's algorithm to reduce the partial product addition and interconnection. A optimized final adder consisting of a number of carry select adders and a special rounding technique called Trailing-1's Predictor are presented to speed up the final addition and rounding. Finally, the exponent block evaluates the sign and exponent in IEEE single precision format, and detects *overflow/underflow*.The architecture of this floating point multiplier is illustrated in Figure 4.17.

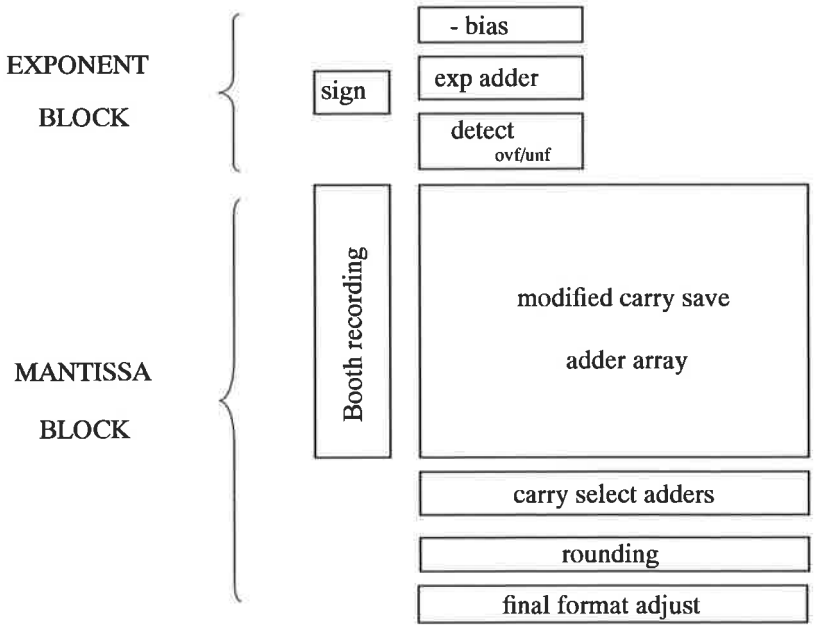


Figure 4.17 The architecture of this floating point multiplier.

4.4.1 A modified carry save array

To summarize integer multiplication algorithms: Multiplication arithmetic is usually realized as additions of partial products. The multiplication time is proportional to the number of additions, since the generation and propagation delay times for sum and carry signals dominate the operation time. Therefore, one of the most important design constraints is to reduce the number of additions to increase the multiplier speed. There are two ways to meet this requirement. One is to reduce the number of partial products to be added like modified Booth's algorithm, and the other is to reduce the number of sum and adding stages like a Wallace tree. A Wallace tree incorporated with modified Booth's algorithm has been prevailing in high-speed multipliers [90][91][98]. However, the Wallace tree suffers from poor structure regularity. Regularity is necessary not only to enhance the design productivity and multiplier bit-width extendability but also to enhance the performance, by allowing less complicated and shorter interconnection wiring.

In order to maintain both regularity and high speed, a *modified carry save array* [97] is used in conjunction with modified Booth's algorithm for the 32-bit floating point multiplier. Figure 4.18 illustrates the structure of this modified carry save array. The array consists of odd rows (shaded rows) and even rows where sum and carry signals generated in an odd row are fed to the next odd row and signals generated in an even row are concurrently transferred to the next even row. In this way, the array has two signal streams in a column in parallel. In the last stage, the sum of odd rows and the sum of even rows are added together. Using this modified array with Booth's algorithm, the maximum number of adder stages is further reduced by half i.e. the maximum number of adder stages $T = n/4$ full adders. However, implementing full adders in the first row of the array is a little difficult, because three partial products from three multiplexers has to be connected together. The connection is complicated and less regular than the rest of the array. Therefore, half adders are used in the first row instead of full adders to get regular structure at cost of one more half adder delay. Therefore, the maximum number of adder stages is $T = \frac{1}{2} + \frac{n}{4}$ full adders.

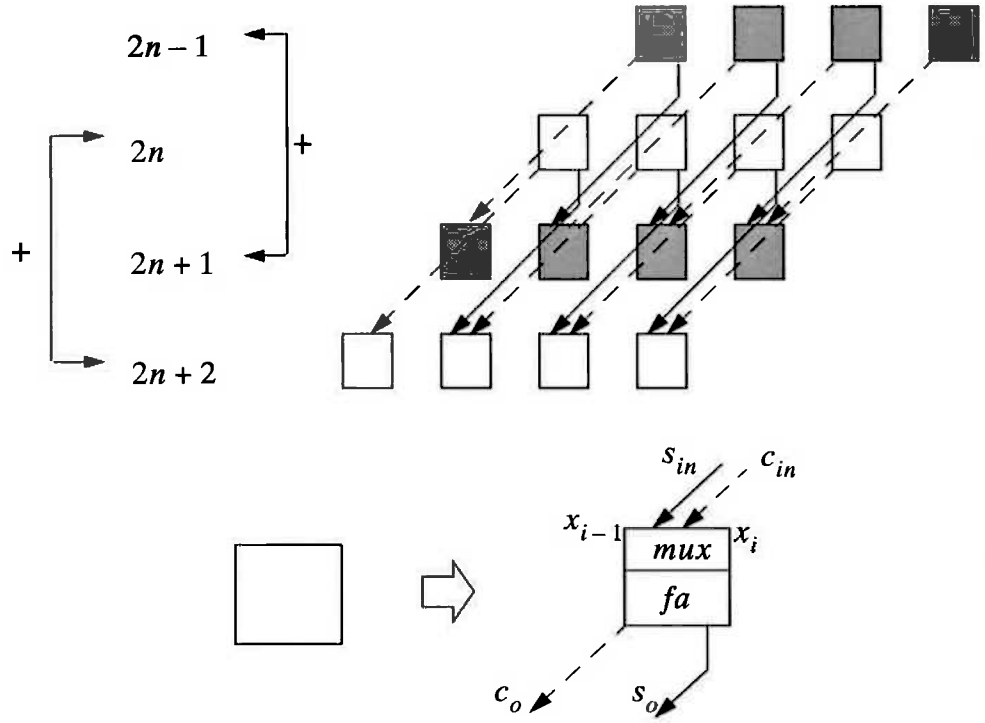


Figure 4.18 A modified carry save array with modified Booth's algorithm.

For IEEE single precision format mantissa n = 25, therefore T = 7 full adders deep, which is comparable to a Booth multiplier using (4, 2) compressors (6 full adders deep) while using a simple and regular structure, necessary for minimizing delays due to interconnect in GaAs technology.

4.4.2 The final adder

To obtain maximum speed, both the array and the final adder part must operate at highest possible speed. From **3.2.6** it is known that for adders exceeding 8 bits in width, the carry select adder is a fast adder with moderate area. Since the final adder width for 25×25-bit mantissa multiplication is $2n$ = 2×25 = 50 bits, a carry select adder is used. It was found that the carry select adder can be further speeded up by the structure called *multi-carry select adder* as shown in Figure 4.20(a). Each block is a pair of ripple carry adders and each block should be one bit wider than the next. Therefore, the best design involves variable-sized blocks. However the size of *m-bit* ripple carry adders must be chosen carefully in order to get minimum delay. For *n-bit* addition, the total delay of the adder of Figure 4.20(a) can be calculated roughly as follows:

$$d_{total} = m \cdot d_{carry} + \left(\frac{n}{m} - 1\right) \cdot d_{mux} \qquad \text{(Eq4.16)}$$

where d_{carry} is the carry in to carry out delay of a full adder, d_{mux} is the delay of one block multiplexer. The HSPICE simulated results show that $d_{carry} = 0.25ns$ and $d_{mux} = 0.3ns$. For 25×25-bit multiplication, $n = 50$. m may be found by differentiating Eq.4.18 with respect to m to find the minimum d_{total}:

$$\frac{\partial}{\partial m}(d_{total}) = d_{carry} - \frac{n}{m^2} \cdot d_{mux} = 0 \qquad \text{(Eq4.17)}$$

Hence

$$m = \sqrt{\frac{d_{mux}}{d_{carry}} \cdot n} \approx 7.7 \qquad \text{(Eq4.18)}$$

$$d_{total} = d_{min} = 3.5ns \qquad \text{(Eq4.19)}$$

The relationship between d_{total} and m when $n = 50$ is shown in Figure 4.19. It can be seen that the delay is minimum when m is around 8 bits. Figure 4.20(b) shows the final adder structure used in the 32-bit floating point multiplier. This structure is chosen also by considering rounding.

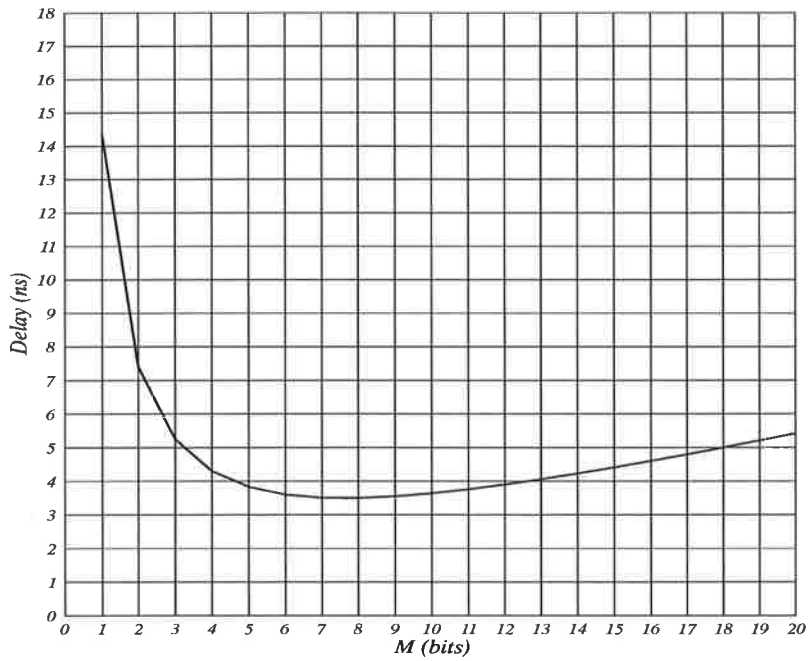


Figure 4.19 The total delay vs. m bit.

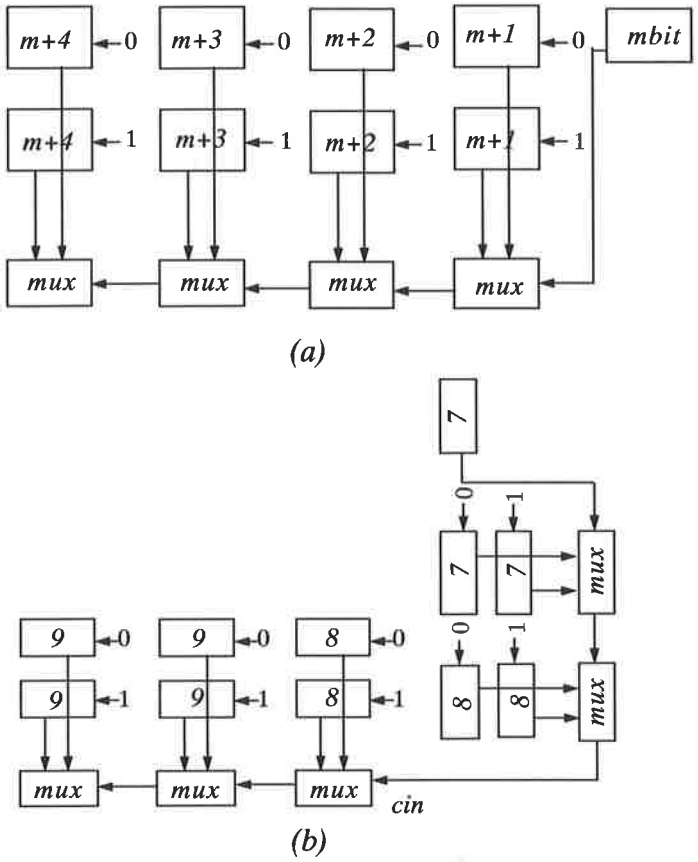


Figure 4.20 The structure of the final adder.

4.4.3 Rounding

Two new rounding techniques are developed for the floating point multiplier. One is using the parallel rounding algorithm described in **4.3.2** combined with the multi-carry select adder. The other is called *Trailing-1's Predictor* (T1P). Both methods perform addition and IEEE rounding in parallel.

1. The CSA method

To implement the multi-carry select adder into the parallel addition and rounding algorithm shown in Figure 4.15, C_{in} has to be combined into the multi-carry select adder, otherwise parallel addition can not be fully achieved. This can be done by using an additional carry select adder, but there are more economical techniques. Consider using two sets of carry select adders, one for C_{in}, one for R_v. Figure 4.14 can be alternatively illustrated as in Figure 4.21. "0" and "1" represent carry select adder, the correct results are selected by C_{in} and R_v, respectively. To simplify the circuit, "0"s are deleted because any X + 0 = X. By combining Figure 4.21 with Figure 4.15, there are two CSA

parallel addition and rounding methods as illustrated in Figure 4.22. Method 1 (Figure 4.22(a)) uses one CSAdder (two rows of full adders in parallel) plus two rows of half adders. The critical path is one CSAdder delay plus two half-adder delay. Method 2 (Figure 4.22(b)) uses one row of full adders plus three rows of half adders. The critical path is one CSAdder delay plus three half-adder delay. Since one full adder equals two half adders, method 1 has one more row of half adders cost but at one half adder faster speed. Thus, method 1 is preferred.

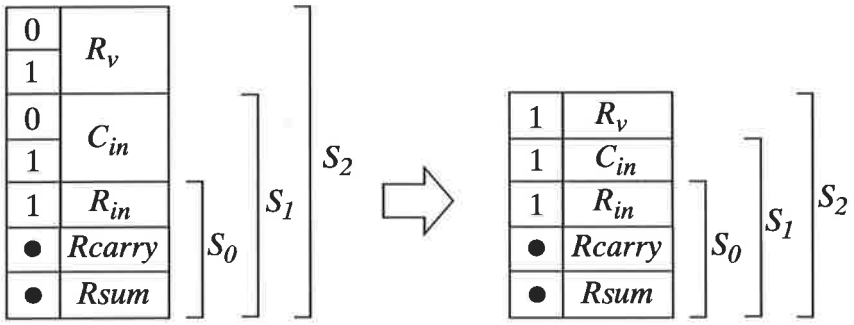
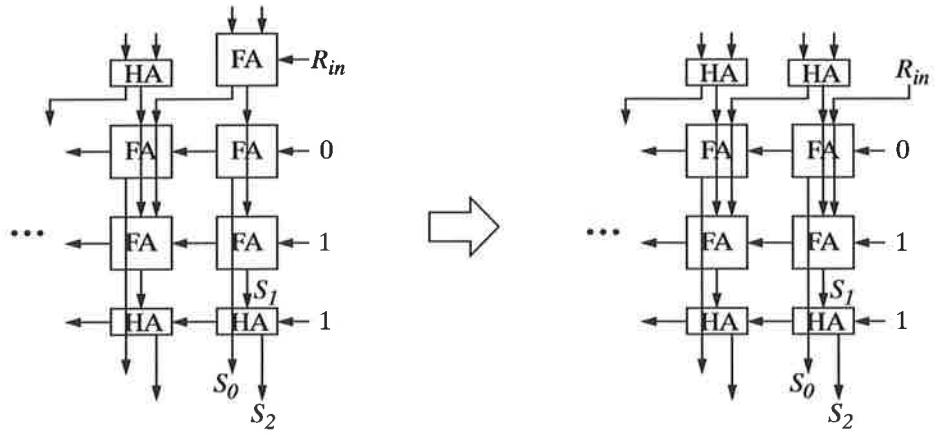
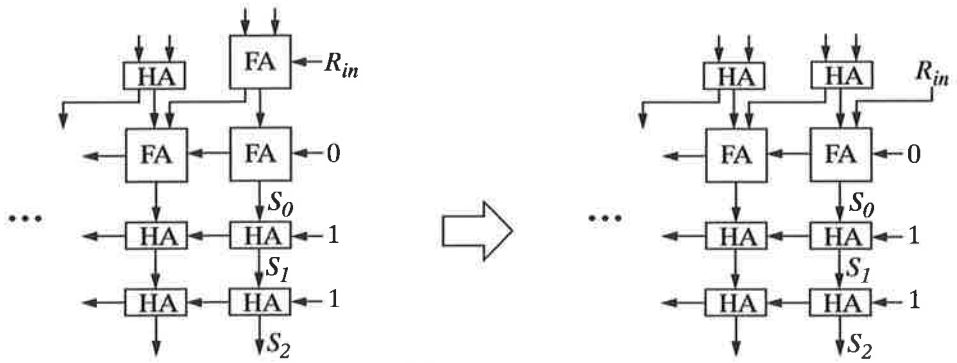


Figure 4.21 Bits to be added at *R* position for CSA method.



(a) Method 1



(a) Method 2

Figure 4.22 The two CSA methods.

The structure of the final adder and rounder for the IEEE round to nearest/even is illustrated in Figure 4.23. Besides the final adder, this parallel rounding method only requires an additional two rows of half adders and two rows of multiplexers. The first half adder row provides a slot for $R_{in.}$ The second half adder row provides a "1" for S_2. The three *mux* rows are for C_{in}, R_v and *renormalization,* respectively.

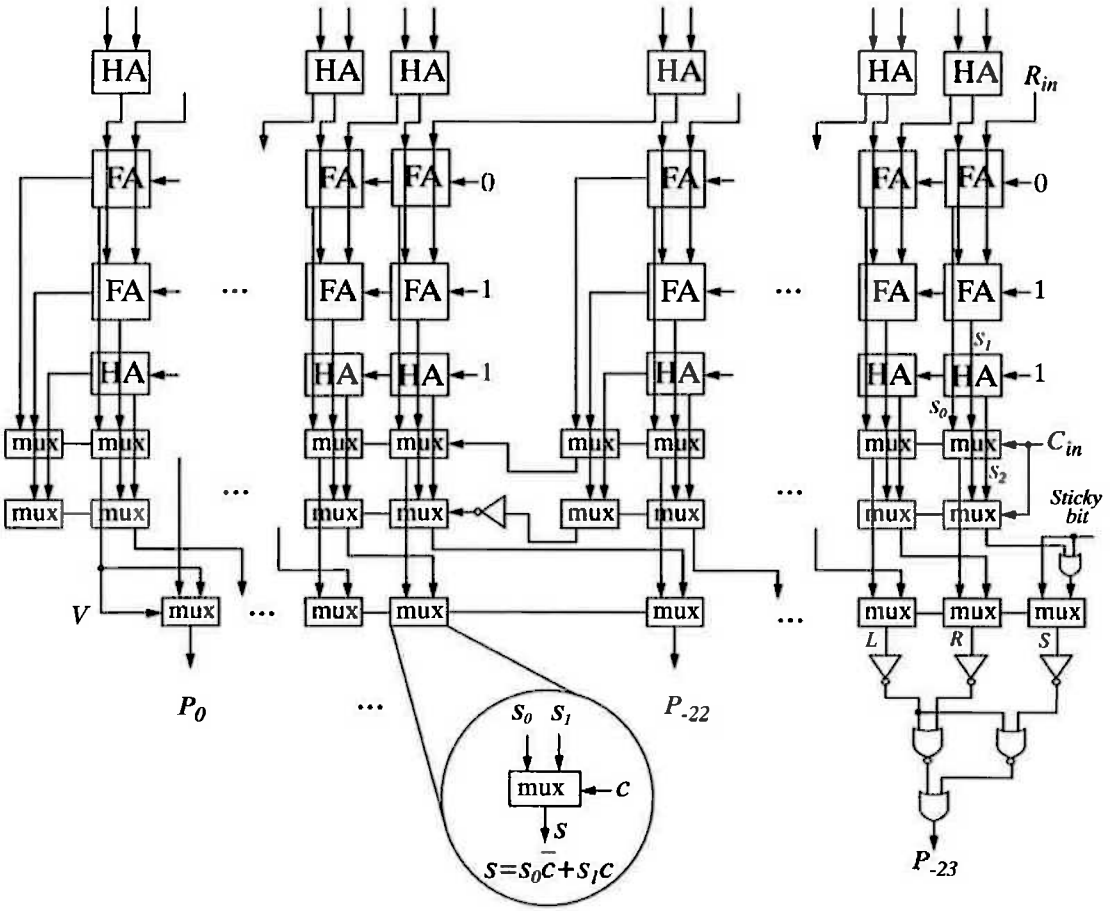


Figure 4.23 The structure of CSA method.

The delay of this CSA parallel addition and rounding can be estimated as the sum of two half adder delay, $2n$-bit CSAdder delay and one row of n-bit mux delay, i.e. $d_{total} = 2d_{ha} + d_{csa\text{-}2n} + d_{mux\text{-}n} + d_{gate\text{-}3}$. Among them, $d_{csa\text{-}2n}$ is taken by the multi-CSAdder and $d_{gate\text{-}3}$ are the three gate delays. Thus, only two extra half-adder delays, n-bit *mux* delay and three gate delays for converting of round to nearest/even are required.

2. The T1P method

The *Trailing-1's Predictor* rounding technique is based on this observation: if there is a large number of "*trailing ones*" in the final addition result *P*, the process of rounding can be quite slow due to the carry being *rippled through* until the first zero in the result is reached.

Instead of employing a second adder to add 1 at rounding bit, a Trailing-1's Predictor would examine the pair of addends to generate a set of flag bits indicating those bits that should be inverted in order to increment P.

Example:

T1P rounding method

	0.1101010	A
+	0.1110101	B
	1.1011111	Sum S_i
+	0 0111111	flag bits $R_i = R_{i-1} \cdot (A_i \oplus B_i)$ $R_{-1} = 1$
	1.1100000	Rounding Result $P_i = S_i \oplus R_i$

It can be seen that at the same time doing final addition, T1P examines the pair of addends A_i and B_i. If they are different, T1P will set the flag bit R_i to 1 indicating sum bit S_i should be inverted to get correct product bit P_i; if A_i and B_i are the same, the flag bit R_i will be 0 and so are the rest of the flag bits, indicating the correct product bit P_i will be the same as sum bit S_i.

A direct application of $R_i = R_{i-1} \cdot (A_i \oplus B_i)$ leads to the simplest design based on a carry-ripple principle as illustrated in Figure 4.24. Other faster, more sophisticated T1P's can be made based on other accelerated addition techniques. However, a carry select version of the T1P is preferred in GaAs technology for the same reasons as were discussed in relation to the choice of adder. This structure is illustrated in Figure 4.25.

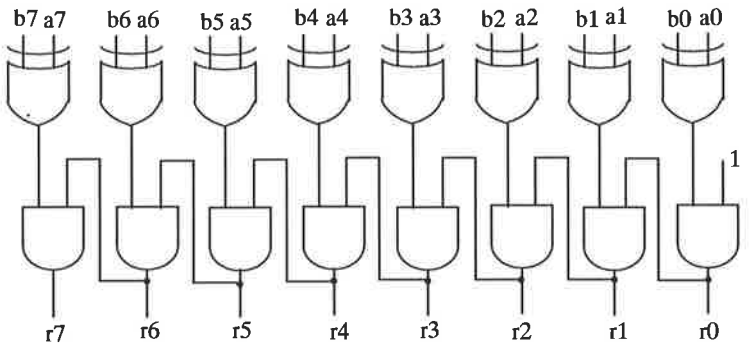


Figure 4.24 A simple 8-bit ripple through T1P.

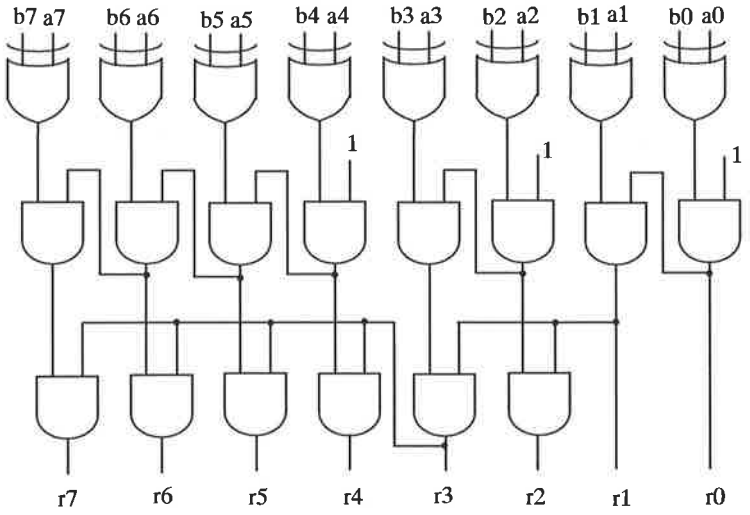


Figure 4.25 An 8-bit carry select T1P.

In order to discuss the rounding procedure clearly, Figure 4.26 shows the top 26 bits of the product, produced in this multiplier by the three most significant blocks of the multi-carry-select adder. The square brackets "[]" indicate the bits to be occupied by the correctly-rounded result. Table 4.7 and Table 4.8 list the actions necessary for producing a correctly-rounded product under the rounding-to-nearest (even) option of the IEEE floating-point standard as a function of: *v*, the overflow bit at the *msb* end of *P*; *l*, the *lsb* of *P*; *r*, the round bit; and *s*, the *sticky* bit. (The round and sticky bits are produced by the three least significant blocks of the final adder and are not included in

Figure 4.26.) Table 4.7 assumes $v = 0$ and Table 4.8 $v = 1$. In Table 4.8, $P^{\wedge}$ indicates P right-shifted 1 place.

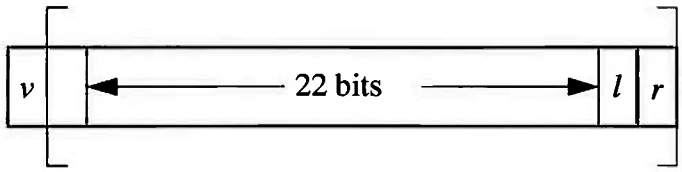


Figure 4.26 Labelling of bits in product *P*.

Table 4.7 Rounding actions for IEEE round-to-nearest (even) for $1 \le P < 2$

l	*r*	*s*	rounded result
0	0	0	P
0	0	1	P
0	1	0	P
0	1	1	$P+2^{-24}$
1	0	0	P
1	0	1	P
1	1	0	$P+2^{-24}$
1	1	1	$P+2^{-24}$

Table 4.8 Rounding actions for IEEE round-to-nearest (even) for $2 \le P < 4$

k	*l*	$(r+s) = s^*$	rounded result
0	0	0	$P^{\wedge}$
0	0	1	$P^{\wedge}$
0	1	0	$P^{\wedge}$
0	1	1	$P+2^{-23\wedge}$
1	0	0	$P^{\wedge}$
1	0	1	$P^{\wedge}$
1	1	0	$P+2^{-23\wedge}$
1	1	1	$P+2^{-23\wedge}$

A further complication is that the T1P must accommodate a carry-in signal ("C_{in}") as well as the pair of addends. The multi-carry-select adder at the foot of the multiplier array is partitioned 7-7-8-8-9-9 as shown in Figure 4.20(b), with the bottom

3 blocks producing the round and sticky bits and the carry-out signal for the top 3 blocks to operate on. In other words, the prediction must accommodate a carry signal being produced that could increment the top 23 bits of the product independently of the rounding.

However, it turns out that this is not a major complication since either the rounding action or the carry signal could cause a carry-propagation in the top 25 bits of the product, but not both. Hence, the carry-in signal is ignored at first and incorporated at a later stage. Table 4.9 summarises the operations to be performed in rounding a product including the event of a carry-in but in the absence of an overflow. A carry-in signal of 1 is accommodated by adding 2^{-24} to P and then determining whether a further 2^{-24} needs to be added for rounding purposes. Consequently, 2^{-24} is added in some cases, and 2^{-23} in others.

Table 4.9 Rounding actions for IEEE round-to-nearest (even) for $1 \le P < 2$

C_{in}	l	r	s	rounded result
0	0	0	0	P
0	0	0	1	P
0	0	1	0	P
0	0	1	1	$P+2^{-24}$ ($+2^{-24}$ round up)
0	1	0	0	P
0	1	0	1	P
0	1	1	0	$P+2^{-24}$ ($+2^{-24}$ round up)
0	1	1	1	$P+2^{-24}$ ($+2^{-24}$ round up)
1	0	0	0	$P+2^{-24}$ ($+2^{-24}$ for c_{in})
1	0	0	1	$P+2^{-24}$ ($+2^{-24}$ for c_{in})
1	0	1	0	$P+2^{-23}$ ($+2^{-24}$ for $c_{in,}$ $+2^{-24}$ round up)
1	0	1	1	$P+2^{-23}$ ($+2^{-24}$ for $c_{in,}$ $+2^{-24}$ round up)
1	1	0	0	$P+2^{-24}$ ($+2^{-24}$ for c_{in})
1	1	0	1	$P+2^{-24}$ ($+2^{-24}$ for c_{in})
1	1	1	0	$P+2^{-24}$ ($+2^{-24}$ for c_{in})
1	1	1	1	$P+2^{-23}$ ($+2^{-24}$ for $c_{in,}$ $+2^{-24}$ round up)

Table 4.10 gives the actions in the event of overflow and taking account of the possibility of a carry-in. Again, a carry-in signal of 1 is accommodated by effectively adding 1 to r and allowing for a possible carry into l.

Table 4.10 Rounding actions for IEEE round-to-nearest (even) for $2 \le P < 4$

C_{in}	k	l	$(r+s)=s^*$	rounded result
0	0	0	0	$P^\wedge$
0	0	0	1	$P^\wedge$
0	0	1	0	$P^\wedge$
0	0	1	1	$(P+2^{-23})^\wedge$ ($+2^{-23}$ round up)
0	1	0	0	$P^\wedge$
0	1	0	1	$P^\wedge$
0	1	1	0	$(P+2^{-23})^\wedge$ ($+2^{-23}$ round up)
0	1	1	1	$(P+2^{-23})^\wedge$ ($+2^{-23}$ round up)
1	0	0	0	$P^\wedge$
1	0	0	1	$(P+2^{-23})^\wedge$ ($+2^{-23}$ round up)
1	0	1	0	$(P+2^{-23})^\wedge$ ($+2^{-24}$ for carry into l)
1	0	1	1	$(P+2^{-23})^\wedge$ ($+2^{-24}$ for carry into l)
1	1	0	0	$(P+2^{-23})^\wedge$ ($+2^{-23}$ round up)
1	1	0	1	$(P+2^{-23})^\wedge$ ($+2^{-23}$ round up)
1	1	1	0	$(P+2^{-23})^\wedge$ ($+2^{-23}$ for carry into l)
1	1	1	1	$(P+2^{-23})^\wedge$ ($+2^{-23}$ for carry into l)

Taking Table 4.9 and Table 4.10 together, P should be increased by 2^{-24} if $\bar{v}\cdot\{c_{in}\cdot\bar{r} + \bar{c}_{in}\cdot r\cdot s + l\cdot r\cdot\bar{s}\}$ is true, and by 2^{-23} if $\bar{v}\cdot\{c_{in}\cdot r\cdot(\bar{l} + s)\} + v\cdot\{c_{in}\cdot(k + l + s^*) + l\cdot(k + s^*)\}$ is true.These additions cover the necessary rounding to nearest actions as well as adding the carry signal from the three least significant blocks of the carry-select adder to the final product. (In fact, splitting the adder in this way speeds the operation of the multiplier a little because two half-length adders are employed instead of one full-length adder.) Hence, the T1P should be designed to operate from bit position l (at significance 2^{-23}) upwards to accommodate rounding by adding 2^{-23}. Rounding by adding

2^{-24} is achieved by activating the T1P if $\bar{v}\cdot\{c_{in}\cdot\bar{r} + \bar{c}_{in}\cdot r\cdot s + l\cdot r\cdot\bar{s}\}$ and r are both high (which can be simplified as if $\bar{v}\cdot\{\bar{c}_{in}\cdot r\cdot s + l\cdot r\cdot\bar{s}\}$ is high), and inverting r irrespective of its initial value. Finally, the act of rounding up may itself cause an overflow, but this event would be detected by the most significant flag bit produced by the T1P going high, which could then provide a right shift signal in concert with the overflow bit, v. The structure of a final addition and rounding by the T1P method is illustrated in Figure 4.27, where Control Logic detects $\bar{v}\cdot\{\bar{c}_{in}\cdot r\cdot s + l\cdot r\cdot\bar{s}\}$. It has one more row of *muxes* and a multi-carry-select T1P comparing to CSA method. Since the inputs of T1P can share the *xor* gate in CSAdder, the T1P is actually just a string of *nand* gates in serial. A *nand* gate can be made by one inverter connected with a *nor* gate in DCFL, and a *mux* uses fewer gates than a half adder. Thus, the T1P method occupies less hardware than CSA method. Furthermore the T1P method implements round to nearest/even directly, thus no round to nearest/up to round to nearest/even conversion needed only a Control Logic is needed to perform $+2^{-24}$ from $+2^{-23}$. The delay of this scheme can be estimated as $d_{total} = d_{csa\text{-}2n} + d_{mux\text{-}n} + d_{Control}$. Detailed layout and simulation will be discussed in the next chapter to see clearly which method is better in terms of delay, area and power dissipation.

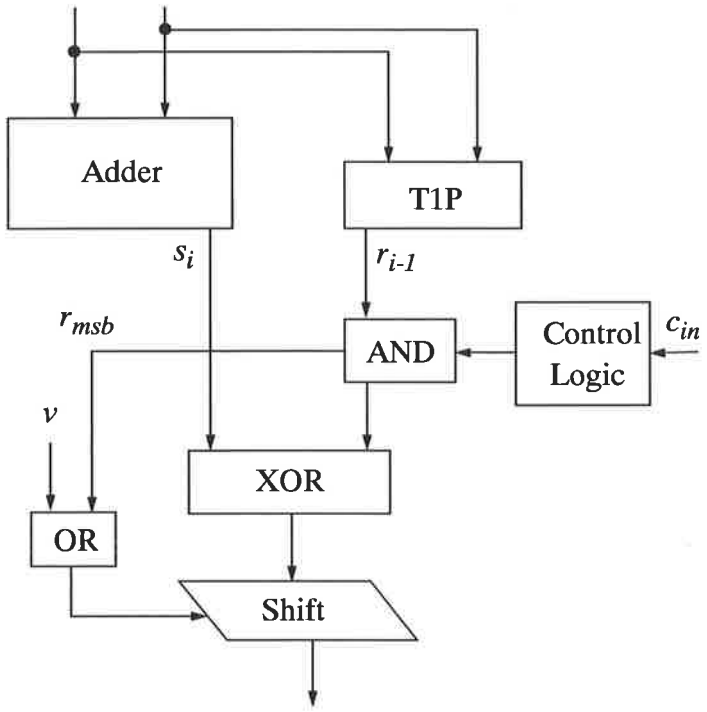


Figure 4.27 The structure of T1P rounding technique.

4.4.4 The exponent block

The exponent block consists of an 8-bit adder, a sign generator and an overflow/ underflow detector. The sign of the product of a pair of IEEE floating point numbers is simply the *xor* of the sign bits of the two operands. This is true even if an infinity or a zero is returned, and is easily determined. The exponent of an IEEE number is biassed by an amount, $\delta = 2^{\varepsilon-1}-1$, where ε is the wordlength of the exponent, so that biassed exponent is always unsigned. Hence, the sum of the two exponent fields will be too large by δ so that $E_p = E_x + E_y - \delta$. For an 8-bit adder, $\delta = 2^{8-1}-1 = 127$. Since subtraction of 127 equals to addition of 10000001_2 in two's complement. Therefore, the exponent adder simply adds both exponents and 10000001_2. In general, the exponent adder can use a simple and slow adder like a ripple carry adder, because 8-bit addition is usually faster than 25×25-bit mantissa multiplication. However, the final addition and rounding stage may cause an overflow ($v = 1$), in this case a normalization shift of 1 bit and an exponent adjustment is necessary. Therefore, a CSAdder is used, one assuming $v = 0$, the other assuming $v = 1$. As soon as v is known, the correct exponent result is picked.

Only overflow and underflow are detected in this floating point multiplier. When Ep ≥ 255, overflow occurs. If Ep ≤ 0 underflow occurs. There are two cases for each flow as shown in Figure 4.28.

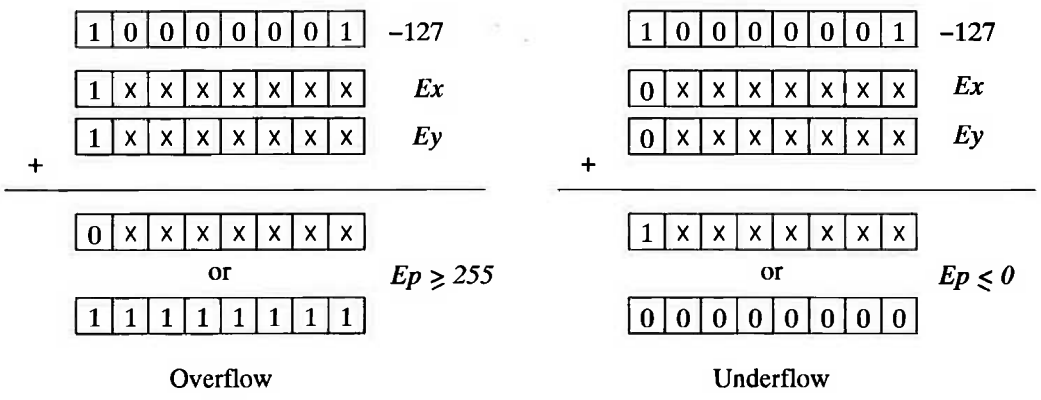


Figure 4.28 The overflow and underflow cases.

It can be seen that if the high-order bits of the exponent fields are different, no over/underflow is possible. If the high-order bits are both 1, the result has overflowed if is has 0 in the high-order bit of the sum or if the sum is 11111111_2. If both the exponents have high-order bits of 0, underflow has occurred if the sum has a high-order bit of 1, or if the sum is 00000000_2.

Examples:

Overflow case 1:

		1	0	0	0	0	0	0	1	= –127
		1	1	0	0	0	0	0	0	E_x = 192
	+	**1**	1	0	0	0	0	0	0	E_y = 192
1	0	**0**	0	0	0	0	0	0	1	E_p = 192 + 192 - 127 = 257 255

Overflow case 2:

		1	0	0	0	0	0	0	1	= –127
		1	0	0	0	0	0	0	0	E_x = 128
	+	**1**	1	1	1	1	1	1	0	E_y = 382
	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	E_p = 128 + 382 - 127 = 255

Underflow case 1:

		1	0	0	0	0	0	0	1	= –127
		0	0	0	0	0	0	0	0	E_x = 0
	+	**0**	0	0	0	0	0	0	0	E_y = 0
		1	0	0	0	0	0	0	1	E_p = 0 + 0 - 127 = -127 ≤ 0

Underflow case 2:

		1	0	0	0	0	0	0	1	= –127
		0	0	0	0	0	0	0	0	E_x = 0
	+	**0**	1	1	1	1	1	1	1	E_y = 127
	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	E_p = 0 + 127 - 127 = 0

In summary, for two IEEE single precision floating point numbers $X = (-1)^{s_x} \cdot 2^{e_x - 127} \cdot (1.f_x)$ and $Y = (-1)^{s_y} \cdot 2^{e_y - 127} \cdot (1.f_y)$, overflow/underflow can be expressed as follow:

$$Overflow = e_{y7} \cdot e_{x7} \cdot \bar{e}_{p7} + e_{p7} \cdot e_{p6} \cdot e_{p5} \cdot e_{p4} \cdot e_{p3} \cdot e_{p2} \cdot e_{p1} \cdot e_{p0}, \quad \text{(Eq4.20)}$$

$$Underflow = \bar{e}_{y7} \cdot \bar{e}_{x7} \cdot e_{p7} + \bar{e}_{p7} \cdot \bar{e}_{p6} \cdot \bar{e}_{p5} \cdot \bar{e}_{p4} \cdot \bar{e}_{p3} \cdot \bar{e}_{p2} \cdot \bar{e}_{p1} \cdot \bar{e}_{p0}. \quad \text{(Eq4.21)}$$

where $e_x = e_{x7}e_{x6}e_{x5}e_{x4}e_{x3}e_{x2}e_{x1}e_{x0}$, $e_y = e_{y7}e_{y6}e_{y5}e_{y4}e_{y3}e_{y2}e_{y1}e_{y0}$ and the sum of the two exponent field and -127 is $e_p = e_{p7}e_{p6}e_{p5}e_{p4}e_{p3}e_{p2}e_{p1}e_{p0}$.

The value of v from the mantissa block could affect overflow/underflow. To speed up the floating point multiplier a CSA-like over/underflow detector can be used at the cost of double the hardware. However, it is observed that the second item of overflow as expressed in Eq.4.20 can be accelerated using the trailing-1's technique (without *xor* gates). The second item in underflow can be sped up using inverters instead of *xor* gates in the T1P shown in Figure 4.25. The inverter plus T1P works like a trailing-0's predictor. In this way, speed and area constrains can be both satisfied. The structure of the exponent block is illustrated in Figure 4.29. Noting that the "1"s of −127 (10000001_2) is combined in the 8-bit CSAdder. This arrangement allows the circuit to perform one addition instead of two additions which will further speed up the exponent block.

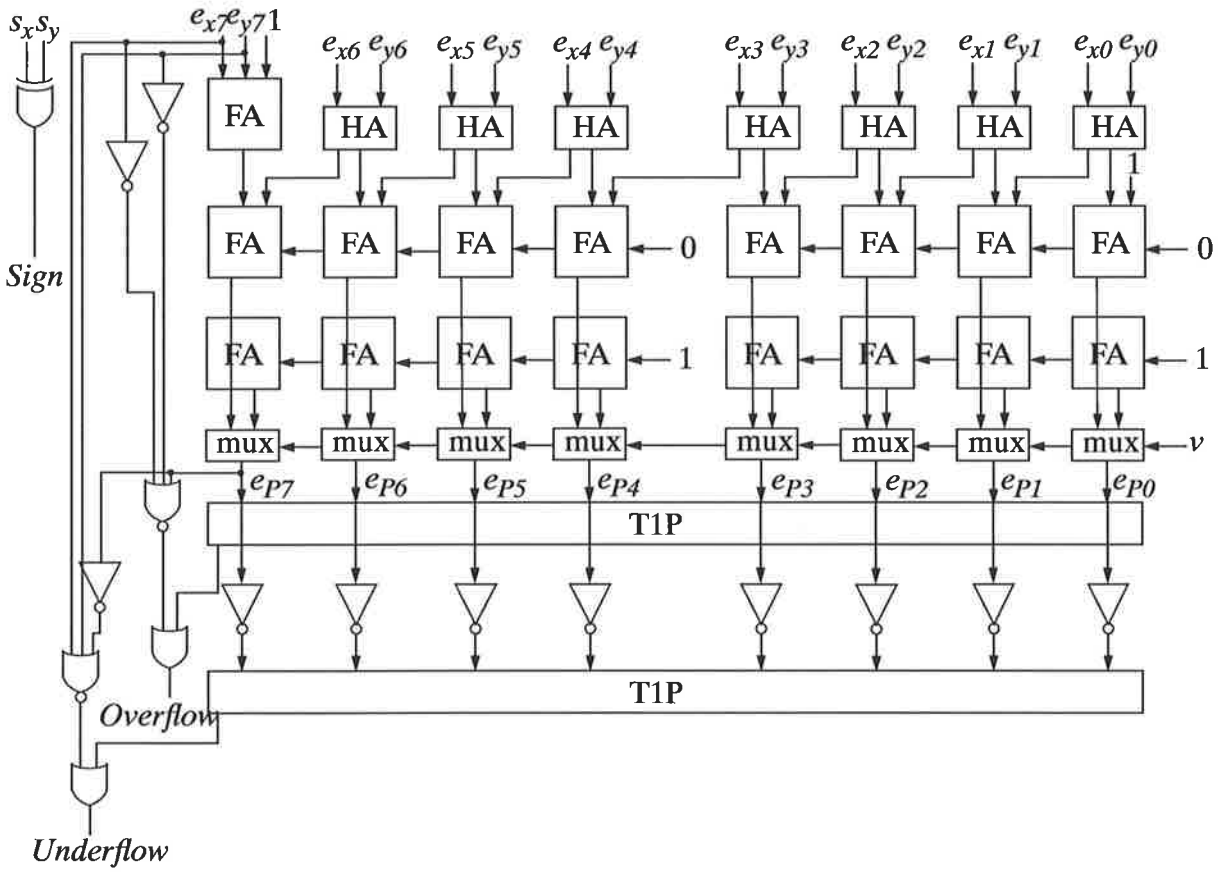


Figure 4.29 The structure of the exponent block.

4.5 Summary

A IEEE single precision floating point multiplier has been described. The analysis of conventional algorithms has been carried out and some new techniques have been developed to choose the best one for GaAs technology. This multiplier is based on a combination of all accelerated parts of a floating point multiplier such as: the modified carry save mantissa array, the multiple carry select final adder and the Trailing-1's Predictor rounding technique, and the exponent block.

The next chapter will describe the implementation of this multiplier, and the significant performance of the circuit can be seen from the layout and the HSPICE simulated results.

CHAPTER

5

Modified Ring Notation Approach—

A New Layout Design Methodology for GaAs VLSI ICs

For CMOS technology there are tools such as COMPASS, OCTTOOLS which have facility to generate mask layout from schematic automatically. However, there are no such tools available for GaAs technology, mainly because that layout style is a critical design parameter for high speed systems.

A Modified Ring Notation approach is presented in this chapter which translates a circuit schematic into mask layout suitable for use in the Vitesse 0.8μm GaAs IC foundry. However, the same idea can be used for other GaAs processes and foundries.

Various primitives have be implemented by different layout styles and compared. The modified Ring Notation approach proves to be the best one to reduce coupling and to improve GaAs technology layout density.

The 32-bit floating point multiplier described in the previous chapter is implemented using this new layout approach. The HSPICE simulated results show that the combination of the fast arithmetic architecture and the new layout style makes the multiplier faster, more compact and lower power.

5.1 The Original Ring Notation Approach

5.1.1 Motivation

The layout style has a major impact on the performance of very high speed VLSI ICs. The main issues of concern are:

1) To minimise interconnect lengths to reduce parasitic capacitance so that the coupling between high speed signals can be minimized.

2) To reduce the inductance and increase the capacitance associated with the power buses so that voltage and current spikes can be reduced.

3) To achieve a high layout density.

4) Pitchmatching of basic blocks.

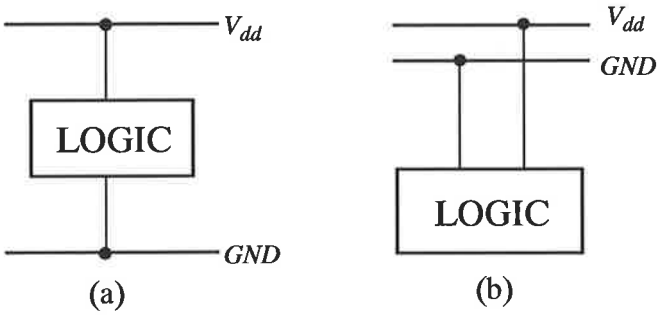


Figure 5.1 (a)The nMOS style and (b) the Ring-Notation style.

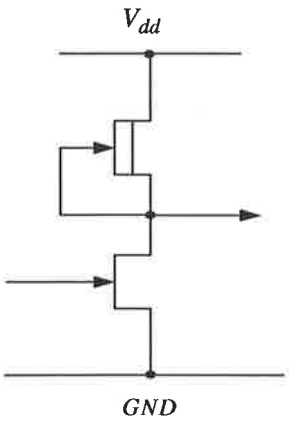
5.1.2 Ring Notation (Ring Diagram)

Eshraghian [24][55] introduced the *Ring Notation* approach explicitly to try to meet the above requirements. In the Ring Notation approach, the *GND* rail is placed between the logic and the power supply V_{dd} rail (Figure 5.1(b)), as compared to the nMOS style layout (Figure 5.1(a)). This reduces the self-inductance of the buses and hence increases immunity to noise induced by signal crosstalk and switching spikes.

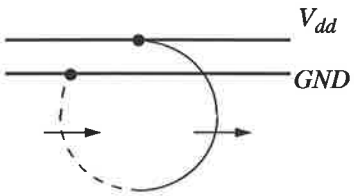
Ring Notation also provides an intermediate method to describe the layout of a circuit design. In the Ring Notation as shown in Figure 5.2(b), the 'dashed' line represents the *efet* while the 'solid' line represents the *dfet*. The two MESFETs are joined together using metal which is implicit in the ring representation for simplicity. However, it should be noted that the missing geometry will appear when the Ring Notation

is translated into mask layout as shown in Figure 5.2(c). Figure 5.3 illustrates the translation from a two input DCFL *nor* gate to a Ring Notation layout.

It can be seen from Figures 5.2 and 5.3 that the translation from a circuit to the layout is straightforward using the Ring Notation approach. Ring Notation allows signal paths to be highlighted and the interconnection strategy to be formulated before the design proceeds to a layout. Complex structures can be readily and systematically mapped thus providing an easy translation method from symbolic notation to mask layout and hence ensuring the portability of designs.



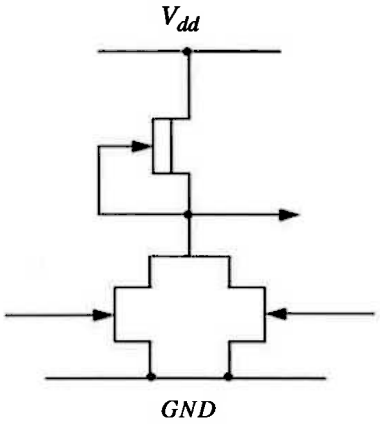
(a) The circuit of a DCFL inverter



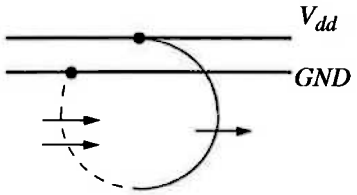
(b) Ring-Notation

(c) Layout

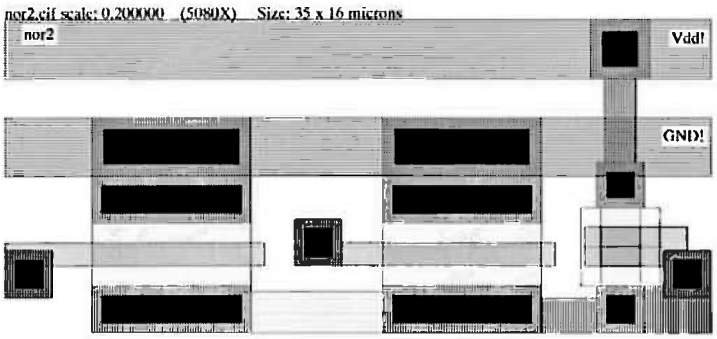
Figure 5.2 Translation from a DCFL inverter to ring style layout.



(a) The circuit of a two input DCFL *nor* gate



(b) Ring-Notation

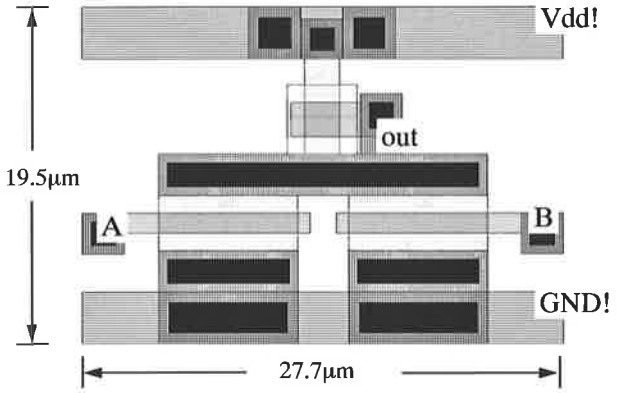


(c) Layout

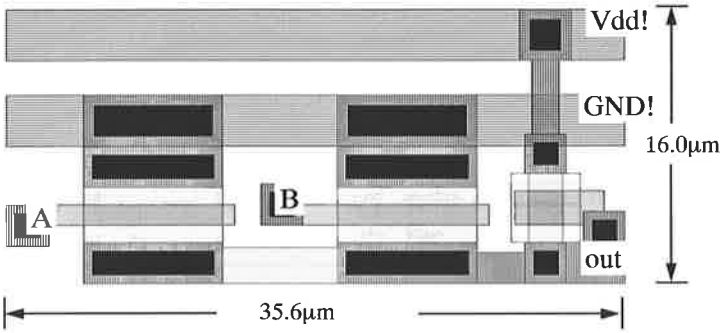
Figure 5.3 Translation from a two input DCFL *nor* gate to ring style layout.

From a designers point of view, ring notation methodology is easier to implement than the nMOS style, allowing a straightforward approach to circuit design. Eshraghian [55] also showed that the Ring Notation technique produces circuits with higher speeds than the nMOS style technique, mainly because of the shorter interconnection length of the path. In this sense, the delay incurred by overcrossing capacitance effects is less in ring style layouts. However, for the same circuit, Ring Notation layouts occupy a little more area than nMOS style layouts ([55] Table 1). This can also be seen from

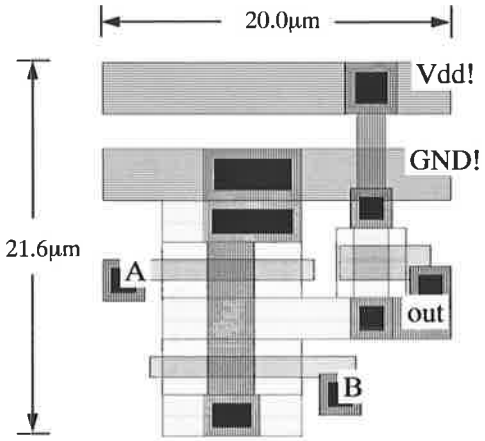
Figure 5.4 which shows a nMOS style two input *nor* gate (a) [25] side by side with a Ring Notation two input *nor* gate (b). In this case, the ring style takes 5% more area than the nMOS style. This situation will get worse for larger circuits, because when the circuit is large and complex, the ring style layout has to use several V_{dd}/GND bus lines instead of extended transistors along one V_{dd}/GND bus line. The extra V_{dd}/GND bus lines occupy extra area.



(a) nMOS style



(b) The Ring Notation



(c) The modified Ring Notation

Figure 5.4 Side-by-side comparison of three styles for a two input *nor* gate.

In order to maintain the advantage of Ring Notation and at the same time reduce the layout area, a *Modified Ring Notation* (MRN) has been developed and will be described in the next section.

5.2 The New Modified Ring Notation Approach

5.2.1 The Modified Ring Notation Layout

The MRN uses the same idea as 'ring notation' or 'ring diagram'. But when translating to the mask layout, the MRN stacks transistors vertically instead of horizontally along the bus lines as shown in Figure 5.4 (c). The layout is more compact, because *e-mode* transistors can share connected vias and no horizontal spacing required between *e-mode* transistors.

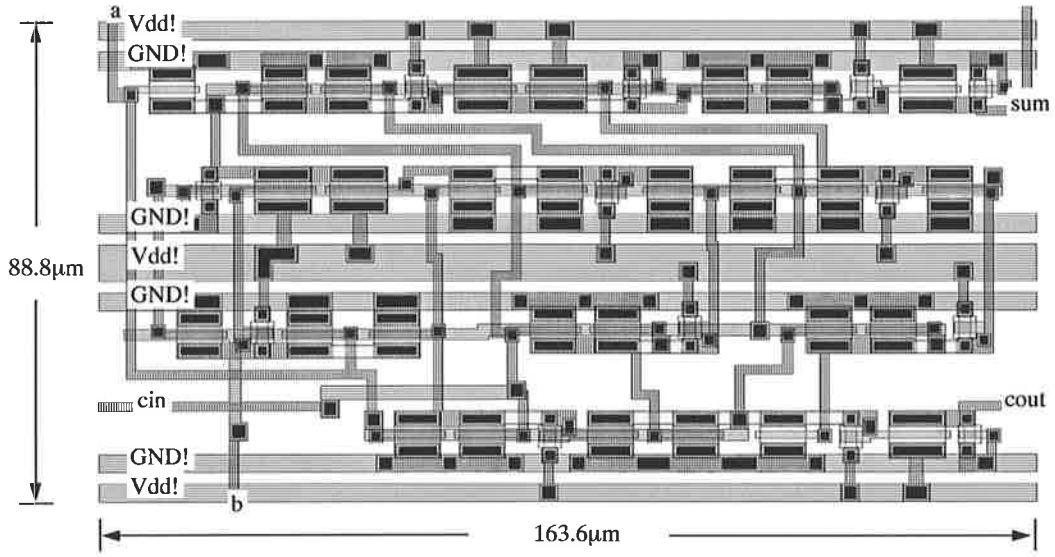
5.2.2 The Improvement in Layout Area

For complex circuits, the advantage of MRN over original Ring Notation in terms of area is more obvious. Figure 5.5 shows a full adder implemented by the two ring styles at same scale. Because original ring style lays-out the transistors horizontally alone the V_{dd}/GND bus lines, it has to use three bus lines to make the layout the desired shape. The extra bus lines and the space between lines occupy a lot of area. In contrast, the modified ring style full adder uses only one V_{dd}/GND bus line, the transistors are stacked vertically along the bus line. There is no wasted space in the layout. The MRN approach reduces the full adder area by a factor of 3 which is very significant.

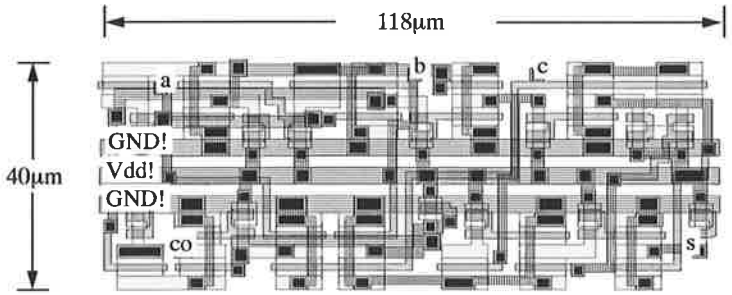
5.2.3 The Improvement in Speed

The MRN not only improves the layout area, but also the speed of the circuit. Table 5.1shows the simulated performance of some basic logic cells implemented by these two Ring Notations with *fan-out = 1*. It can be seen that when the circuit is as simple as *nor* gates, the area of MRN is improved, but the delay is increased. This is because using shared vias reduces the area of the vias, hence the resistance is increased. Consequently, the current through the transistor is decreased, which slows down the speed of the circuit. However, when the circuit is more complex, such as a full adder, the MRN also has a significant improvement in terms of speed. It can be seen that the MRN approach increases the speed of the full adder by a factor of 1.31–1.75. This is

because the more compact layout reduces the length of connections and therefore the capacitive loads.



(a) A full adder using original ring style.



(b) A full adder using modified ring style.

Figure 5.5 A full adder implemented by (a) original ring style and (b) the modified ring style.

Table 5.1 Results for a selection of basic logic blocks

	Original Ring Notation				Modified Ring Notation			
	delay(ps)		area(μm^2)	power(mW)	delay(ps)		area(μm^2)	power(mW)
*nor*2	60.0		569.6	0.250	63.0		432.0	0.250
*nor*3	81.8		801.6	0.278	89.7		541.0	0.278
full adder	co	282	14528	3.467	co	161	4720	3.467
	sum	462			sum	350		
5-to-1 *mux*	371		7873.6	3.03	282		6205.7	3.03

5.3 Global Power Supply and Ground Arrangement

5.3.1 Local Power and Ground Arrangement

The V_{dd} and *GND* bus lines must be placed carefully to reduce their self inductance so that their susceptibility to current transients can be reduced. The impedance of these bus lines can be approximated using the CPW or CPS models depending on their separation. Figure 5.6 show the Characteristic impedance of (a) CPW, (b) CPS as a function of the shape ratio $k = a/b$, using the models described in **2.4.1**. The plots use the normalised substrate thickness h/b as a parameter (noting the difference of a and b between CPW and CPS).

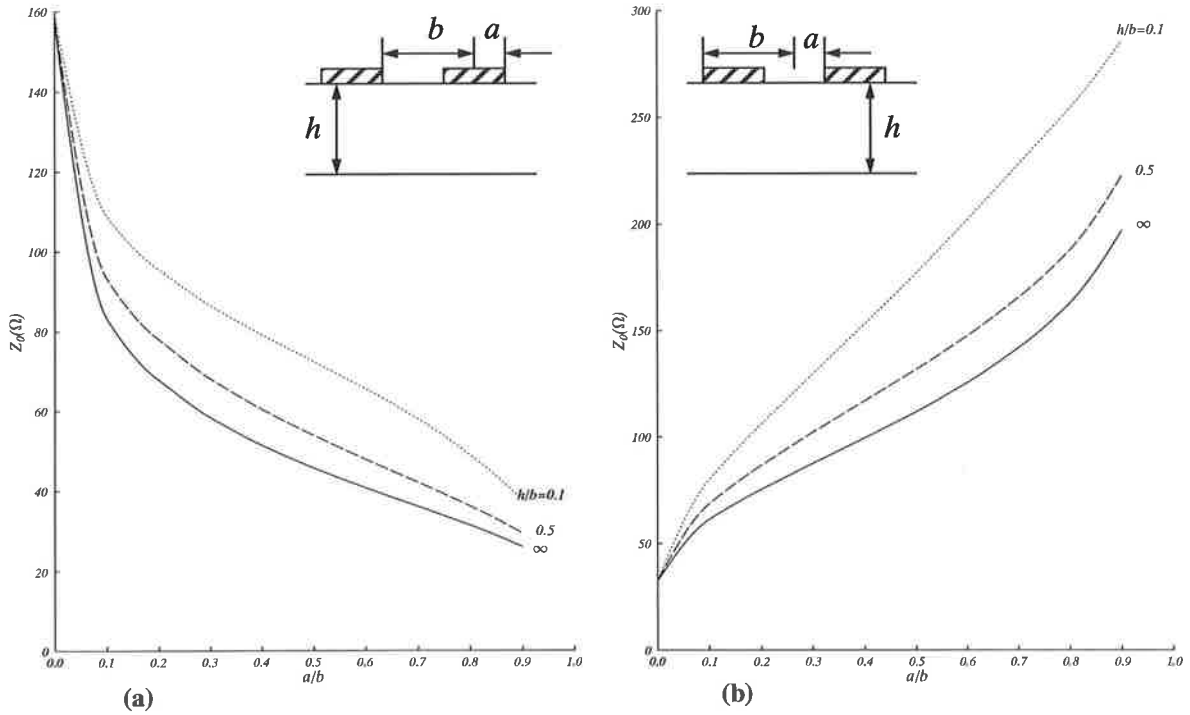


Figure 5.6 Characteristic impedance of (a) CPW (b) CPS as a function of the shape ratio $k = a/b$, taking the normalised substrate thickness h/b as a parameter.

It can be seen that for both CPW and CPS geometries, either the width of the two V_{dd}/ground buses have to be widened or the spacing between the two V_{dd}/ground buses have to be reduced in order to get smaller impedance and therefore smaller inductance.

The MRN, like original Ring Notation, forces the V_{dd} and *GND* bus lines to be placed as close as possible with a minimum distance determined by the design rules. On the other hand, in nMOS layout style, the bus lines are placed far apart on either side of the logic gates.

Figure 5.7(a) and (b) illustrates the normal sizes and the relative distances of the V_{dd}/ground buses for two layout configurations. The nMOS layout shown in Figure 5.7(a) is clearly a CPW model. Referring to Figure 5.6(a) with $h/b = \infty$ (usually $h = 600\ \mu m$ [25]), and typical values given in Figure 5.7(a), the characteristic impedance of a nMOS layout style is about 108Ω. The MRN layout shown in Figure 5.7 is a CPS model. Referring to Figure 5.6(b) with $h/b = \infty$, and typical values given in Figure 5.7(b), the characteristic impedance of a MRN layout style is about 80Ω. This analysis shows that the MRN has a quarter of improvement in the inductance between bus lines even at the leaf cell level. At higher levels of hierarchy this improvement becomes even more significant as the distance between the bus lines is increased.

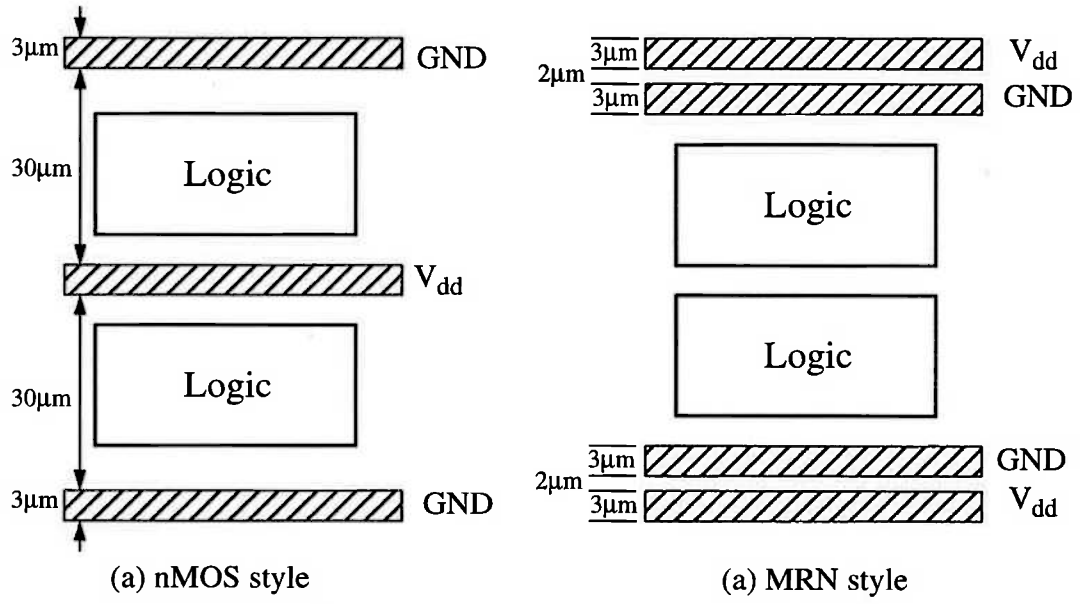


Figure 5.7 The usual sizes and the relative distances of the bus lines of (a) nMOS (b) MRN style.

5.3.2 Global Power and Ground Arrangement

The objective of the power distribution is to provide constant and equal power supply and ground potentials to every device on a chip. Since the metal thickness is constrained by the processing technology, increased line width may be needed to reduce voltage variations. In conflict with this goal, one would also like to minimize the chip area consumed by power buses so that the density of circuits and signal interconnections can be as high as possible. These objectives are further complicated by the problems like resistance (*ir* drop), electromigration (current density limitations) and inductance (*LdI/dt* voltage variations).

In order to solve the above problem, a *net-like* connection for power supply and ground is used in conjunction with the MRN approach as shown in Figure 5.8. Metal 2 (M2) lines are the ring power/ground buses used horizontally as local buses to connect to rows of gates, represented by triangles. Metal 3 (M3) lines are used vertically for global buses. Because M3 has the highest current density and is transparent to the rows of gates, using M3 instead of M1 can save space. As local V_{dd} and *GND* buses, the M3 V_{dd} and *GND* buses pair should be placed as close as possible to reduce themselves inductance. In Figure 5.8 connections between M3 and M2 are represented by X symbols for vias. The width required for the M2 buses depends on the current driven by the transistors on the row between via connections to the M3 buses. The width of the M3 buses depends on bond pad width. The number of bond pads required can be determined by considering the worst case *ir* drop and electromigration current density (J_{max} = 2.8mA/μm for M3 [27]). By using this method, all the inductance, electromigration and the resistance are reduced because of the parallelism, without increasing chip area.

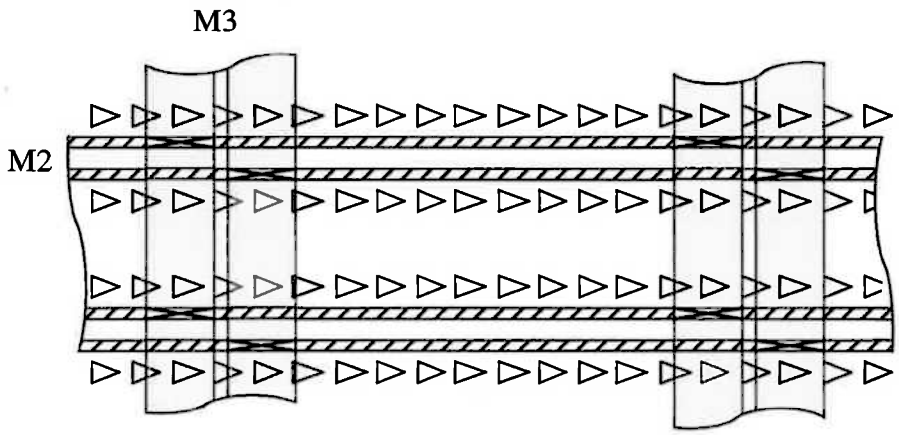


Figure 5.8 Net-like power lines arrangement

The arrangement of M3 power/ground wiring on the chip must be considered. There are typically many options available. If *ir* drop and inductance were not important, these buses could be arbitrarily located. However, for big and high density chips, it is necessary to select a low *ir* drop and low inductance layout. In general, multiple ground and supply pads must be used in order to meet metal migration limits in most chips. This requirement has the additional benefit of reducing both the *ir* and inductive noise in the power and ground buses. Interleaved and distributed ground and supply

connections are much better than grouped multiple connections since the total inductance and resistance are lower.

5.4 Implementing the 32-bit Floating Point Multiplier

After fully considering the algorithm and layout style, the detailed layouts of the 32-bit floating point multiplier can be implemented by following the introduction of the MRN for basic gates.

5.4.1 Mantissa Multiplier

The mantissa multiplier is a 25×25-bit (including hiding bit) modified carry save array. Figure 5.9 illustrates the floor plan of the array which is arranged as a square for convenient implementation and area efficiency.

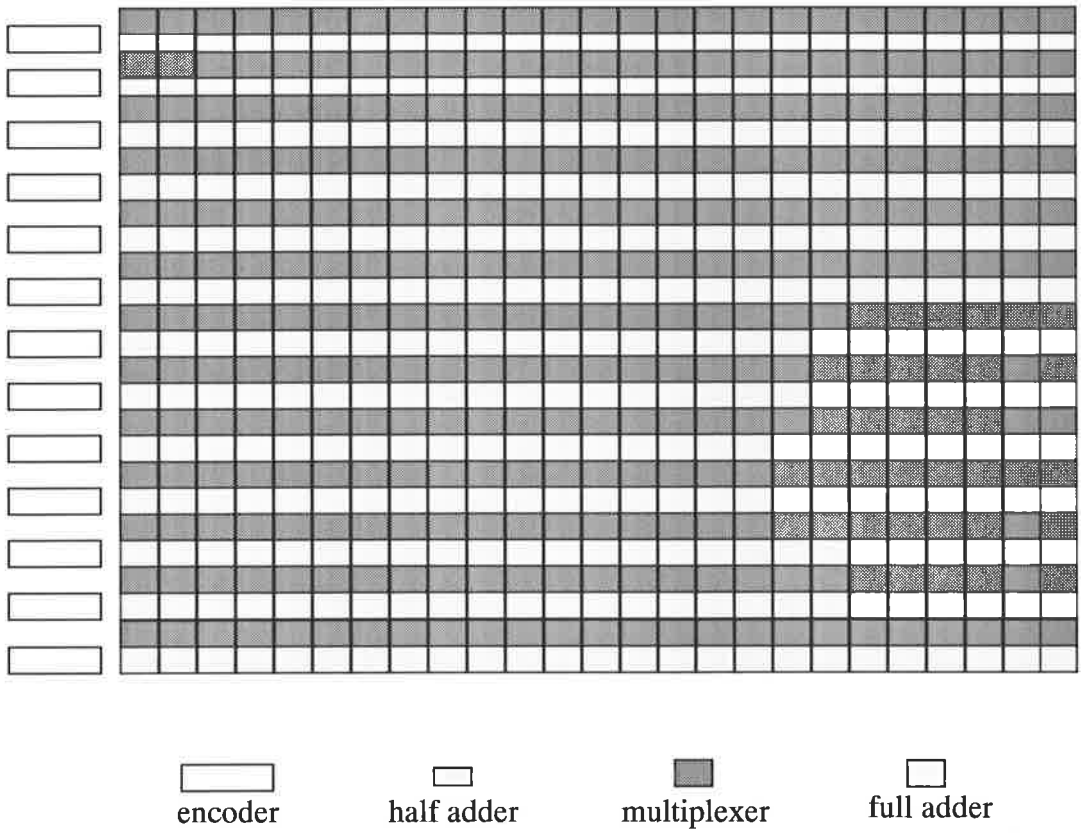
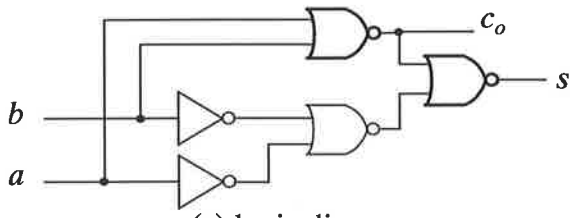
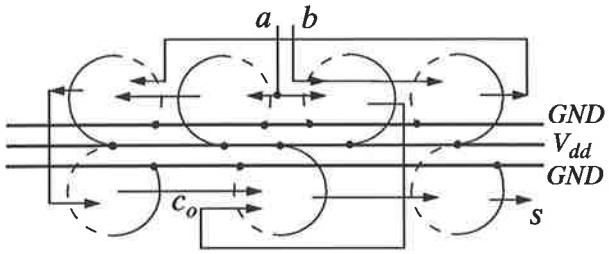


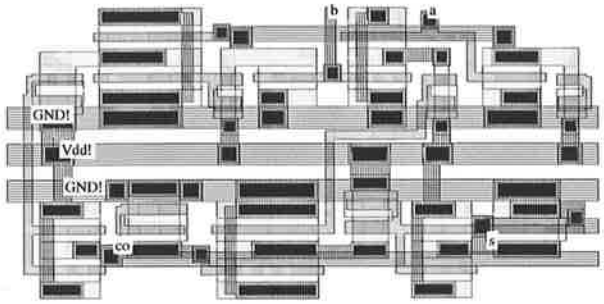
Figure 5.9 The floor plan of the 26-bit mantissa array.



(a) logic diagram



(b) modified ring notation



(c) layout

Figure 5.10 The implementation of the half adder using MRN.

1. Half adder

The first two rows of adders are half adders. Figure 5.10 shows the implementation of the half adder using MRN. Note that the gates drawn in bold lines are implemented by SDCFL to increase driving ability, because the outputs of the half adder need to drive three to four gates plus 150*pF* capacitive load of interconnection wire. Table 5.2 shows the characteristic of the implemented half adder. The area information comes from MAGIC layout, the delay and power dissipation are from the worst case HSPICE simulated results at *tt* parameters and 75°C temperature with *fan-out* = *4* and C_{load} = *150ff* (from layout extraction).

Table 5.2 The characteristics of the implemented half adder.

half adder	delay (*ps*)		area (μm^2)	power dissipation (mW)
	co	240	3200	4.5
	sum	311		

2. Full adder

The full adder is the most important and frequently used cell in the multiplier. The characteristics of the full adder will dominate the performance of the whole floating point multiplier. Therefore, the full adder has to be designed carefully. Figure 5.11 illustrates three full adder designs: (a) is implemented directly from Eq. 3.51 and Eq. 3.52; (b) uses only *nor* gates to reduce the total gates of the full adder; (c) consists of two half adders to perform the full adder function. It seems that design (a) is the fastest, because it has only 3 gate delays. However, it has a 4-input *nor* gate to drive big load, which will seriously degrade the speed of the full adder, even for a 4-input SDCFL *nor* gate. Alternatively, this 4-input nor gate can be replaced by two 2-input *or* gates and one 2-input *nor* gates, but the speed still suffers. Design (b) has 3 gate delays (the *or* gate is no cost in delay according to **2.2.3**) and the least number of gates among the three full adders. However, the five 3-input *nor* gates plus two *or* gates will take more area and longer interconnect wires which slow down the whole speed of the adder. On the other hand, even though design (c) has 6 gate delays and one or two more gates than other designs, it consists of inverters and 2-input *nor* gates, and only one 3-input *nor* gate. This makes it the most compact and the fastest design which is demonstrated by the HSPICE simulated results for these three full adders with four fan-out at typical process parameters as shown in Figure 5.10. Thus, design (c) is used in this multiplier. The layout of the full adder shown in Figure 5.11(c) varies from place to place depending on the place of the input/output required by its adjacent cells. Figure 5.13 illustrates a typical implementation of the full adder. Table 5.3 shows the characteristic of the implemented full adder at *tt* parameters and 75°C temperature with *fan-out* = *4* and C_{load} = *100ff* (from layout extraction)

Table 5.3 The characteristics of the implemented full adder.

full adder	delay (*ps*)		area (μm^2)	power dissipation (mW)
	co	231	4399.7	5.1
	sum	436		

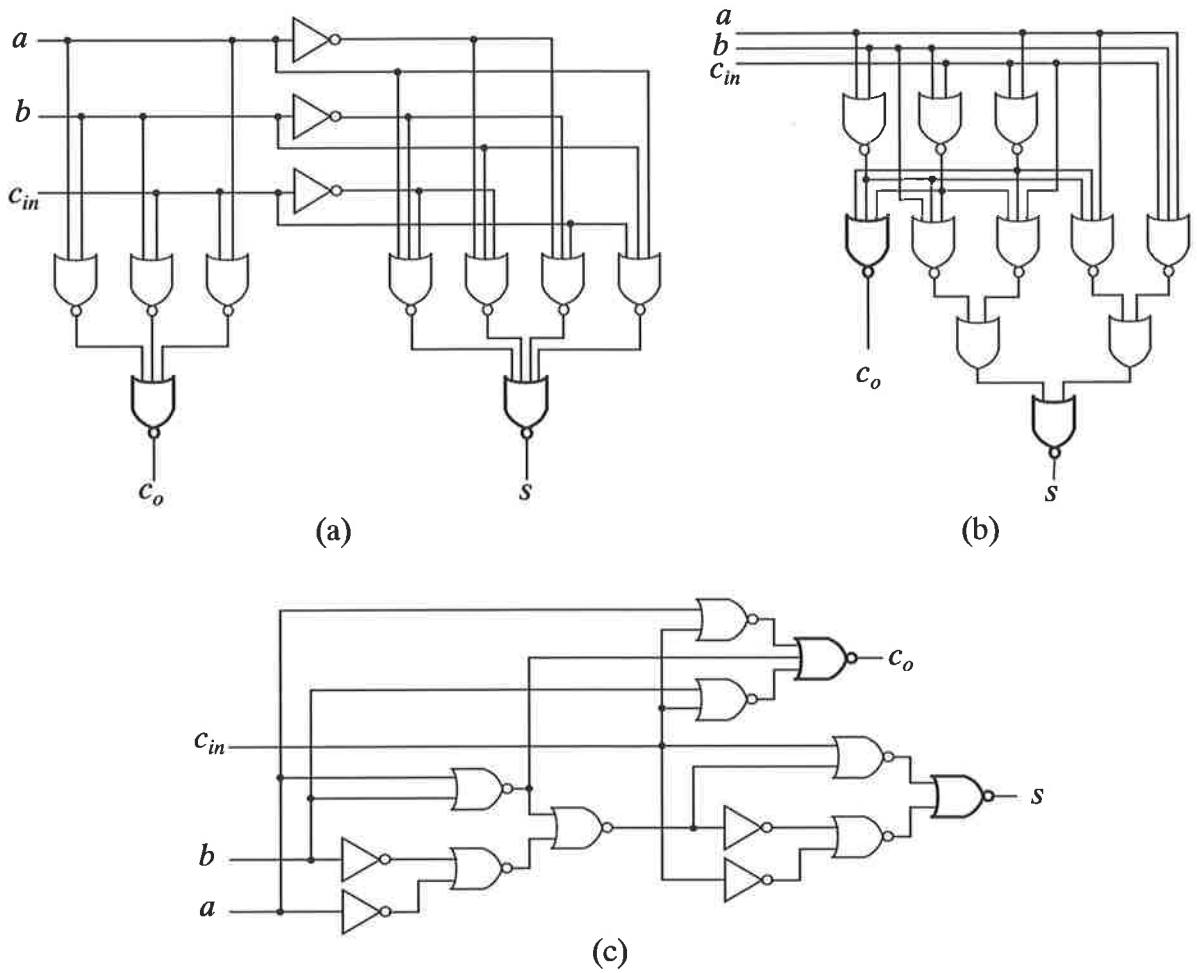


Figure 5.11 Three full adder designs.

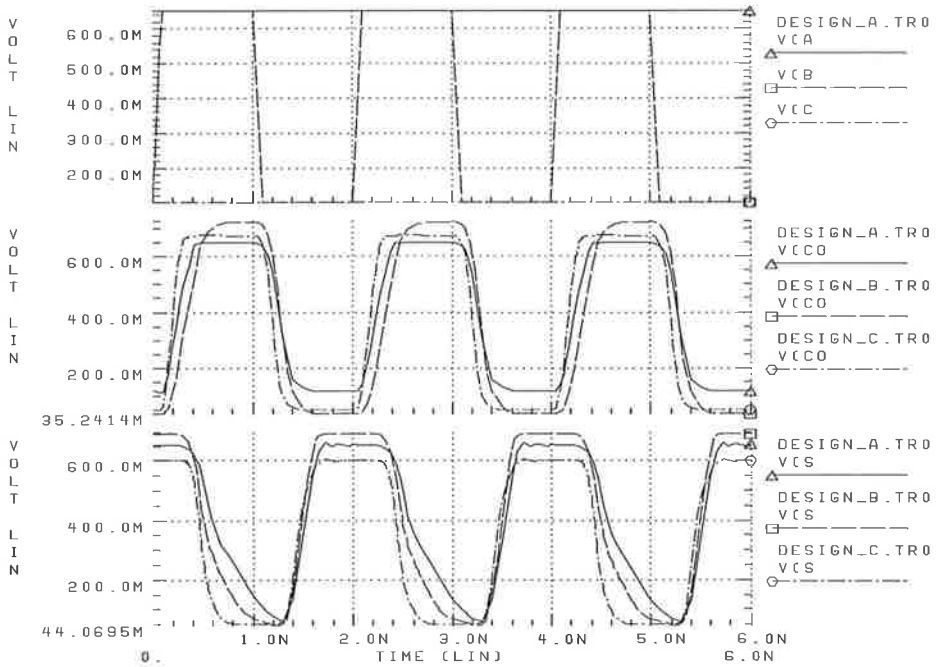
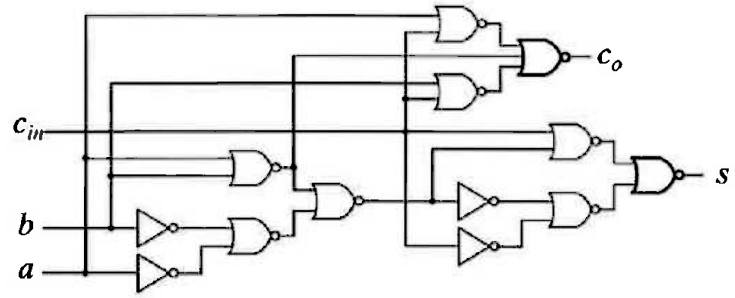
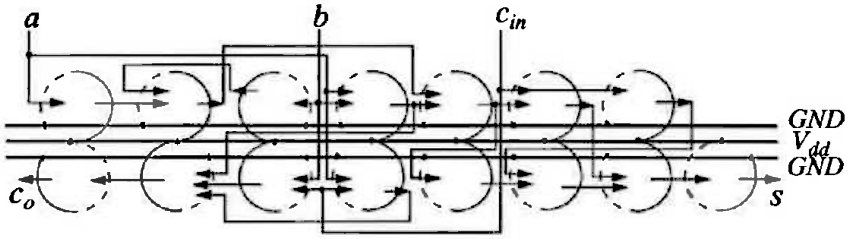


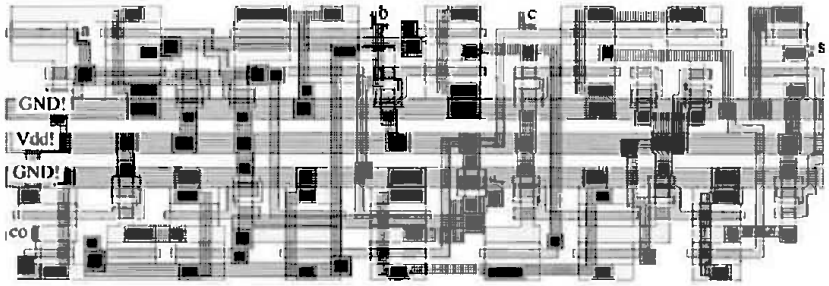
Figure 5.12 The HSPICE simulated results of the three full adders.



(a) logic diagram



(b) modified ring notation



(c) layout

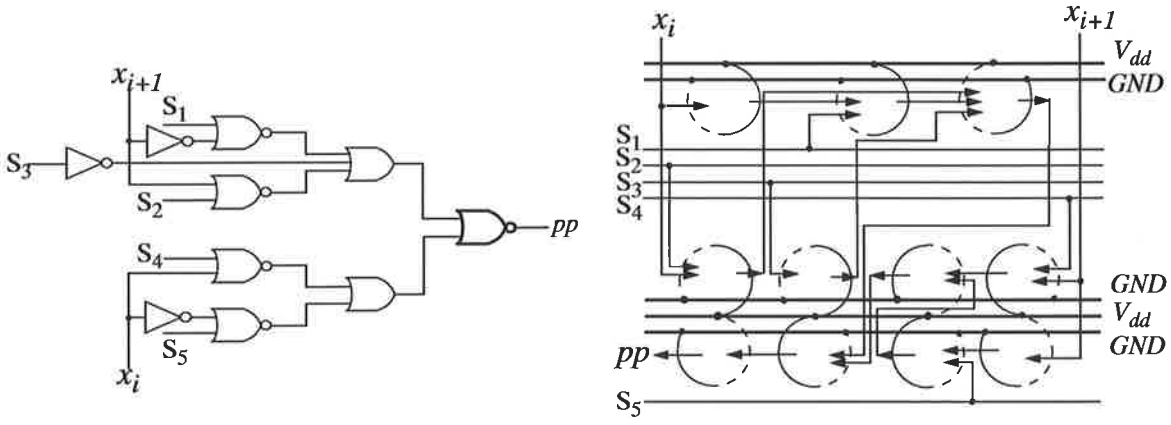
Figure 5.13 The implementation of the full adder using MRN.

3. Multiplexer

For the five-to-one multiplexers, there are five control signal buses along each *mux* row. They are arranged in parallel with the local V_{dd}/GND buses and implemented by Metal 2. This makes the control signal buses have lower *ir* drop, inductance and electromigration problem. Using Metal 2 also allows the control buses to overlap the transistors of the *muxes* as much as possible to save layout area. Figure 5.14 illustrates the implementation of the multiplexer using MRN. The simulated characteristic performance of the multiplexer is shown in Table 5.4 with *fan-out = 4*.

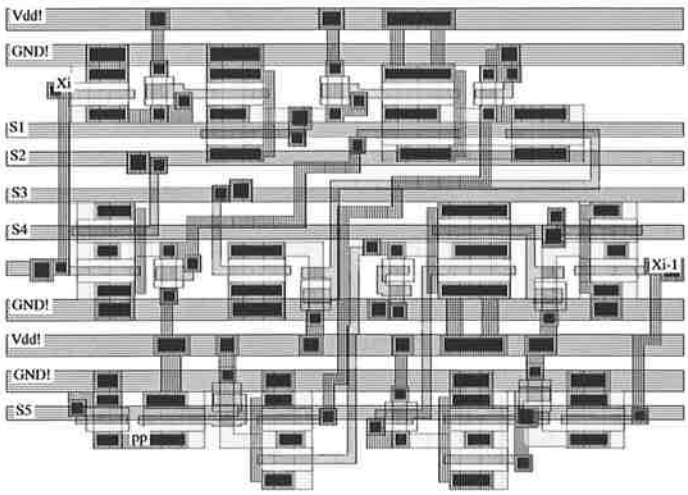
Table 5.4 The characteristics of the implemented multiplexer.

5-to-1 *mux*	delay (*ps*)	area (μm^2)	power dissipation (mW)
	326	3200	3.34



(a) logic diagram

(b) modified ring notation

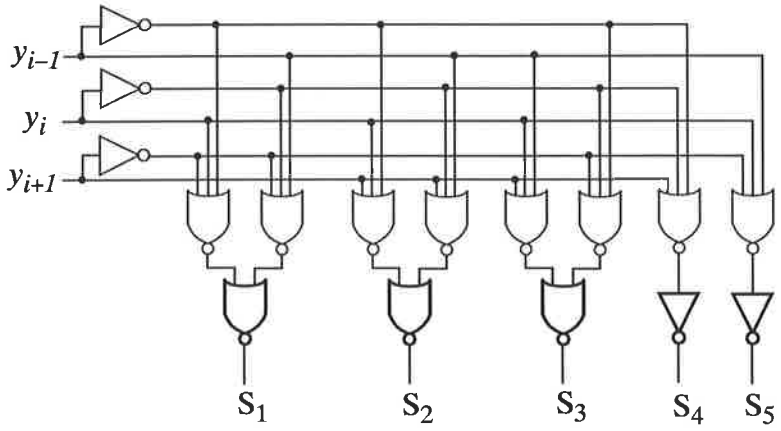


(c) layout

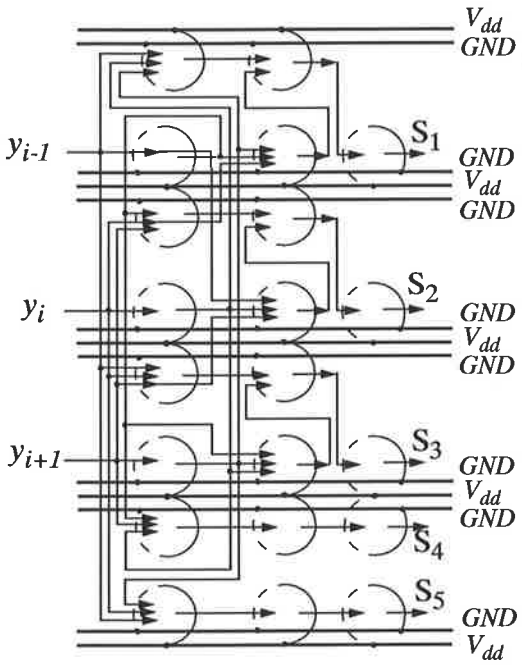
Figure 5.14 The implementation of the multiplexer using MRN.

4. Booth encoder

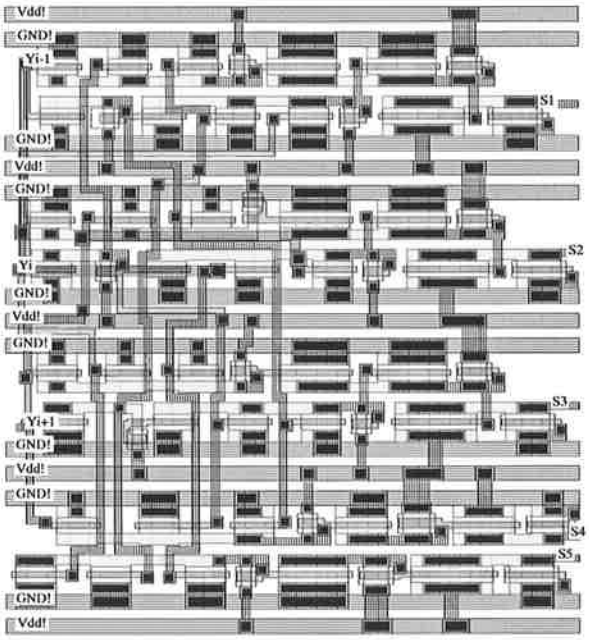
The encoders are placed on the left of the multiplexers. Each encoder generates the five control signals for each multiplexer row. Therefore, five V_{dd}/GND buses are used to correspond to the signals. The outputs of the encoder use larger SBFL gates to drive such heavy loads. The implementation of the multiplexer using MRN is illustrated in Figure 5.15. The characteristic performance of the implemented encoder is shown in Table 5.5 when driving 26 5-to-1 *muxes*.



(a) schematic



(b) MRN



(c) layout

Figure 5.15 The implementation of the encoder using MRN.

Table 5.5 The characteristics of the implemented encoder.

encoder	delay (*ps*)	area (μm^2)	power dissipation (mW)
	385.8	17335.5	17

5.4.2 Sticky Bit Generator

According to its definition, *sticky* bit *S* is the boolean *or* value of the least significant *n-2* bits. In another words, it is the *or* of all of the bits to the right of the *R* bit. There are a total 22-bits which must ored together. Since the *or* gate of an AOI structure (Figure 2.6) has to be inserted between two DCFL gates, and the maximum

number of inputs is restricted to 2, DCFL *nor* gates followed by inverters are used to perform the *or* function. Figure 5.16 illustrates the logic diagram of the sticky bit generator using DCFL 3-input *nor* gates and inverters. It takes 6 gate delays to generate the sticky bit, which is acceptable, because it is not needed until the final addition is finished.

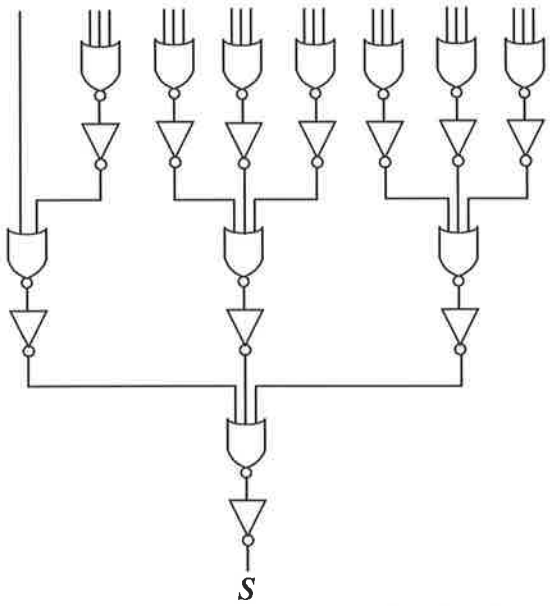


Figure 5.16 The sticky bit generator circuit.

It is not necessary to implement all of the gates of the sticky bit generator in a single block. The *nor* gates and inverters can be placed wherever is convenient in the mantissa array, then connected together.

5.4.3 Final Adder/Rounding

Two rounding algorithms, the CSA method and T1P (Trailing-1's Predictor) method, were presented in Chapter 4. In this section, first, an 8-bit CSA final adder/rounder and an 8-bit T1P final adder/rounder are implemented and compared. Then the final 48-bit adder/rounder for the 32-bit floating point multiplier based on the better method is implemented and simulated.

1. An 8-bit CSA method adder/rounder

Figure 5.17 illustrates the structure of an 8-bit CSA method adder/rounder. It consists of half adders, full adders and two-to-one multiplexers. The half adders and full adder used can be as same as those in the multiplier array. The implementation and the characteristics of the multiplexer using MRN are illustrated in Figure 5.15 and Table 5.6, respectively.

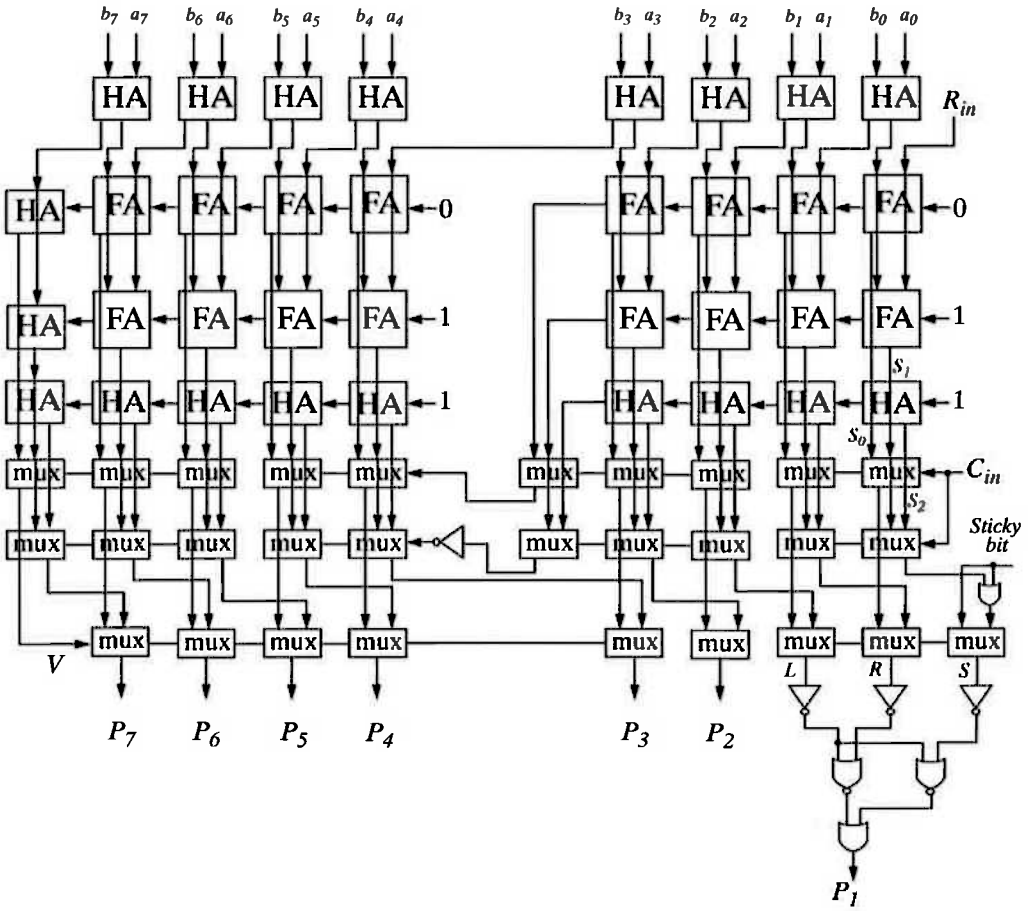
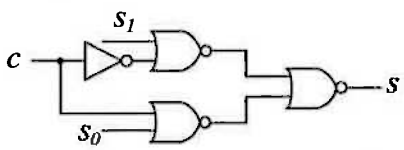


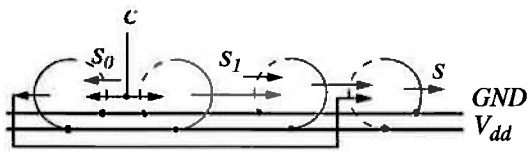
Figure 5.17 The structure of an 8-bit CSA method adder/rounder.

Table 5.6 The characteristics of the implemented two-to-one multiplexer.

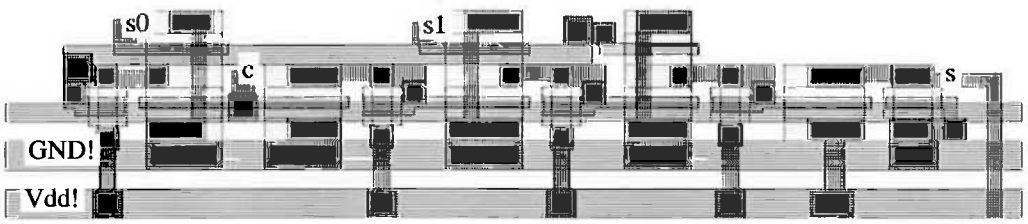
2:1 MUX	delay *ps*	area μm^2	power dissipation mW
	115.7	2052	1.5



(a) logic diagram



(b) modified ring notation



(c) layout

Figure 5.18 The implementation of the multiplexer using MRN.

Figure 5.19 shows the HSPICE simulated results of the 8-bit CSA adder/rounder at the worst case. The worst case occurs when a carry ripples through the adder/ rounder.

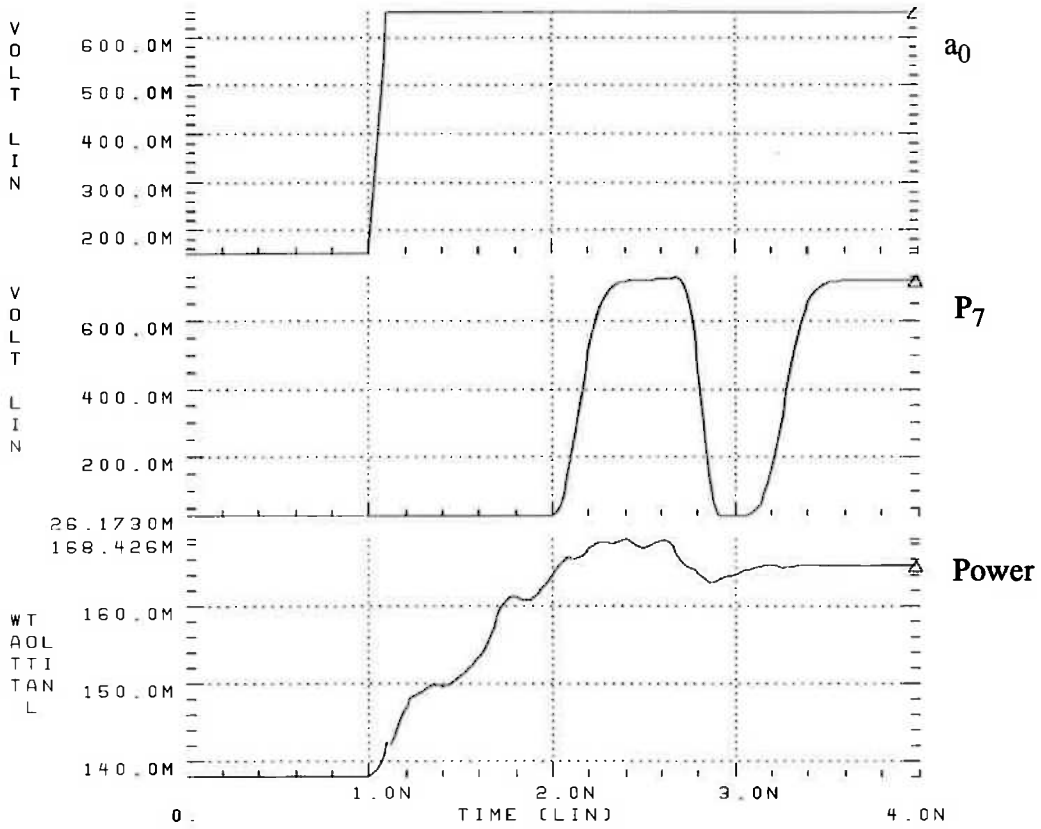


Figure 5.19 The HSPICE simulated results of the 8-bit CSA adder/rounder.

2. An 8-bit T1P method adder/rounder

The schematic diagram of the 8-bit T1P (Trailing-1's Predictor) adder/rounder is illustrated in Figure 5.20 which uses the Carry-Select T1P as shown in Figure 4.25. The T1P starts at the l bit position. The r bit rounding is achieved by activating the T1P if $\bar{v}\cdot\{\bar{c}_{in}\cdot s_0\cdot s + s_1\cdot s_0\cdot\bar{s}\}$ is high. Figure 5.21 illustrates the schematic of the control logic, where $r = s_0$, $l = s_1$, $s = sticky\ bit$

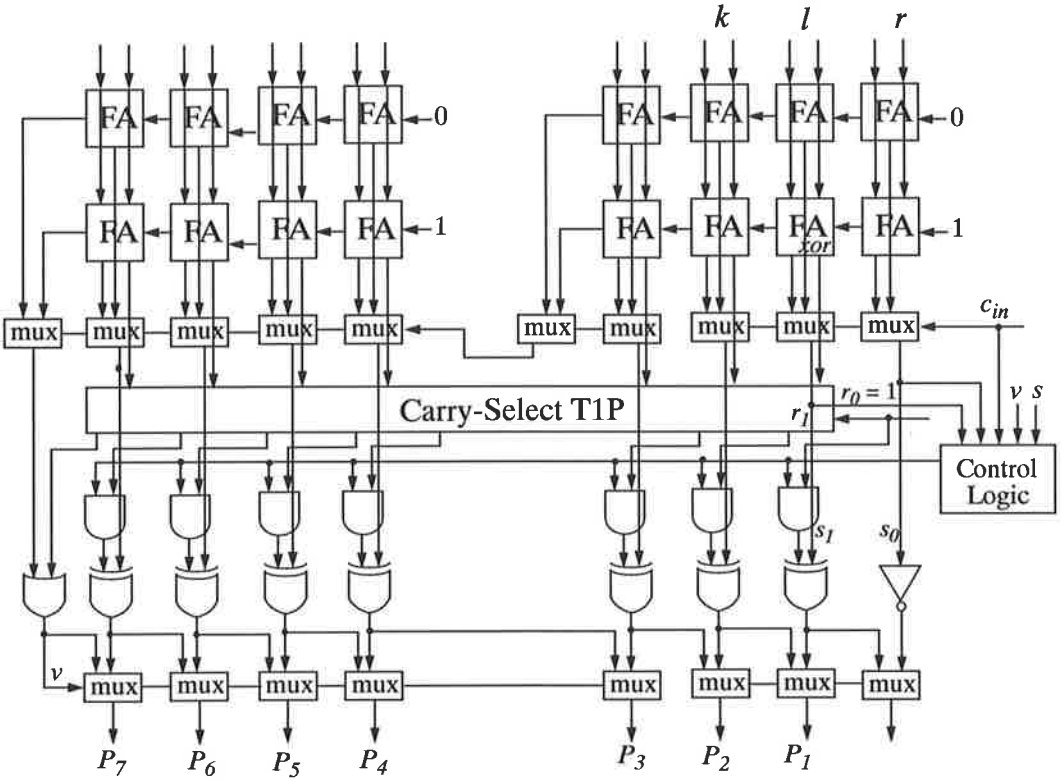


Figure 5.20 The structure of T1P rounding technique.

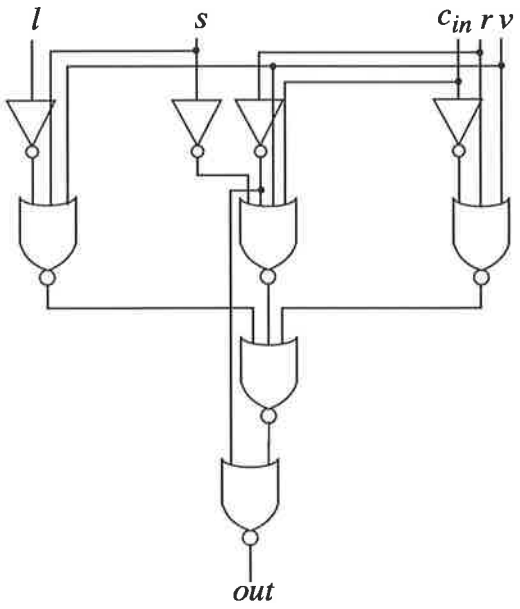


Figure 5.21 The schematic of the control logic.

Example:

		1	0	1	1	0	1	0	1	a
		0	1	1	0	1	0	1	0	b
	1	0	0	0	1	1	1	1	1	sum s_i
		0	0	0	1	1	1	1	1	r flag
	1	0	0	1	0	0	0	0	0	$s_i \oplus r_{i-1}$
shift		1	0	0	1	0	0	0	0	P

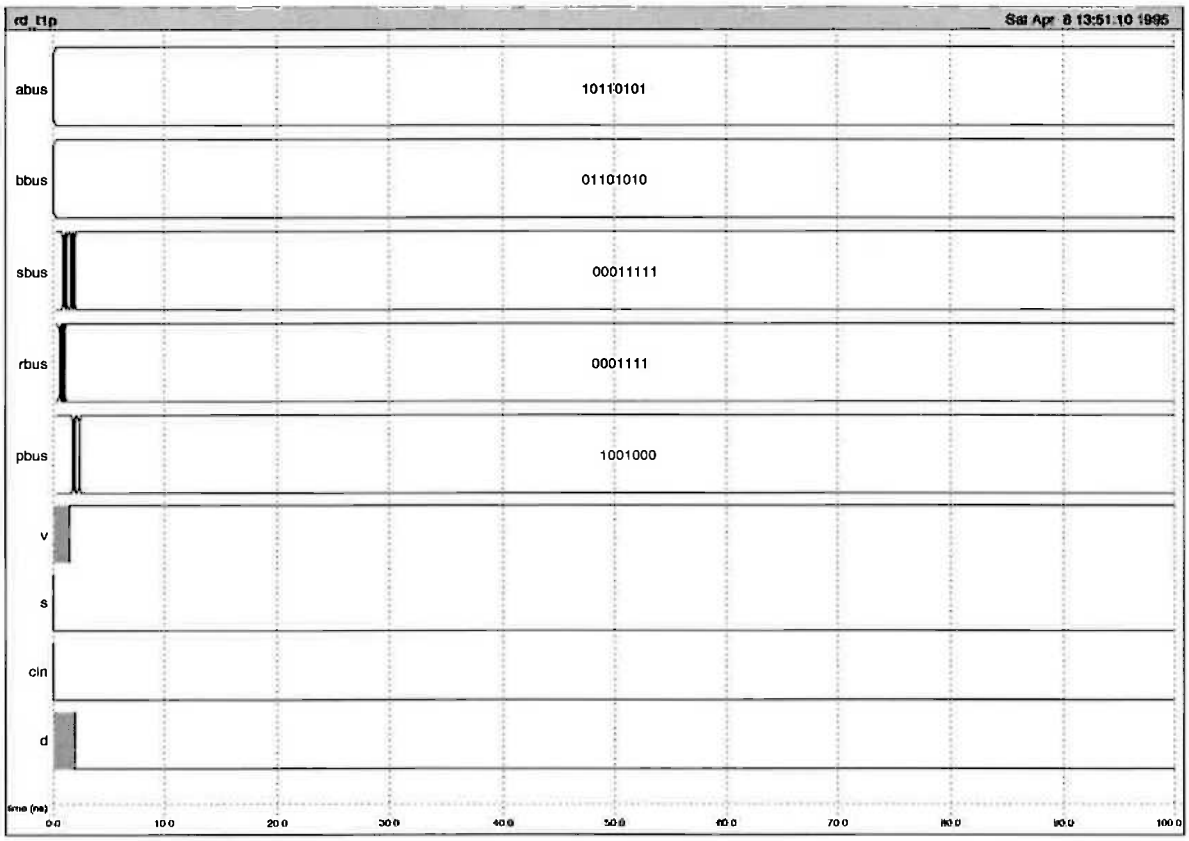


Figure 5.22 The simulated results of the 8-bit T1P adder/round in IRSIM.

Figure 5.22 shows the IRSIM simulated results of this example. The worst case delay can be found in the HSPICE simulated results shown in Figure 5.23. Comparing to Figure 5.19, the T1P method is much faster and uses less power than the CSA method. The detailed comparison of these two methods is shown in Table 5.7. It can be seen that the T1P method is 50% faster and 25% smaller than the CSA method. The T1P method also saves 16% power. Therefore, the T1P rounding technique is adopted in the floating point multiplier. The partitioning of the final adder is 7-7-8-8-9-9 as illustrated in Figure 4.20.

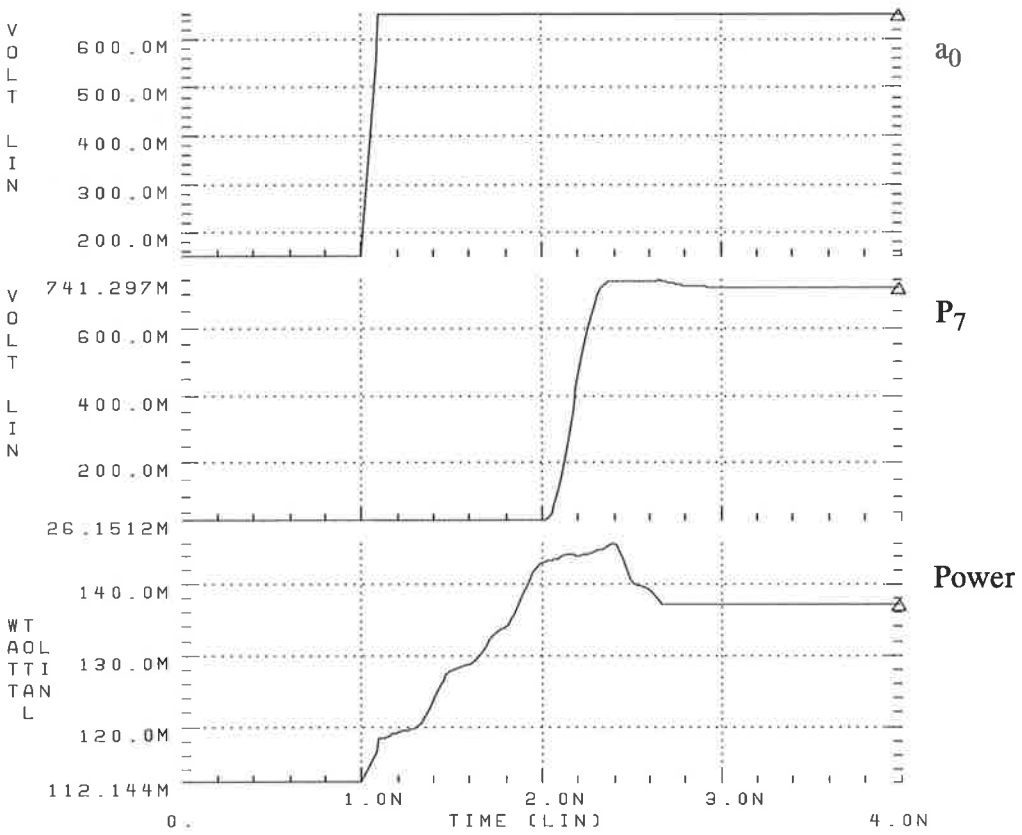


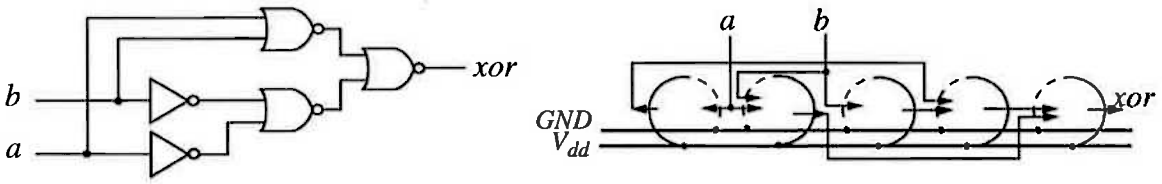
Figure 5.23 The HSPICE simulated results for the worst case of the 8-bit T1P adder/rounder.

Table 5.7 The comparison of the two implemented rounding methods.

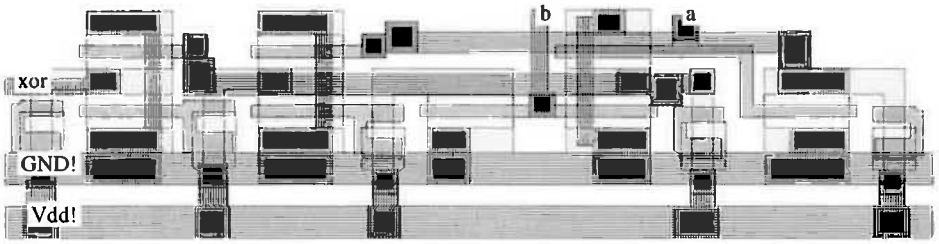
	delay *ps*	**area** mm^2	**power dissipation** mW
CSA	2.22	0.376	153.0
T1P	1.12	0.280	128.5

5.4.4 Exponent Block

The exponent block consists of an 8-bit adder, a sign generator and an overflow/ underflow detector. The sign generator of the IEEE floating point multiplier is simply an *exclusive-or* gate. Figure 5.24 illustrates the logic diagram and the implementation of the *exclusive-or* gate using MRN. The performance of the *xor* gate is shown in Table 5.8.



(a) logic diagram (b) modified ring notation



(c) layout

Figure 5.24 The implementation of the *xor* gate using MRN.

Table 5.8 characteristics of the implemented *xor* gate.

xor	**delay (*ps*)**	**area (μm^2)**	**power dissipation (mW)**
	220	1874.9	1.3

The 8-bit adder is built from a row of half adders and an 8-bit CSAdder. The overflow/underflow detector is sped up using the carry select T1P technique (without *xor* gates). The layout of the exponent block corresponding to Figure 4.29 is shown in Figure 5.25. The worst case delay of the exponent block occurs when the sum of the exponent adder and the overflow *v* from the mantissa block are zero which causes a underflow. Figure 5.23 shows the IRSIM results for various cases including overflow and underflow. Figure 5.24 shows the HSPICE simulated results of the worst case delay. It can be seen that the worst delay is *840ps*.

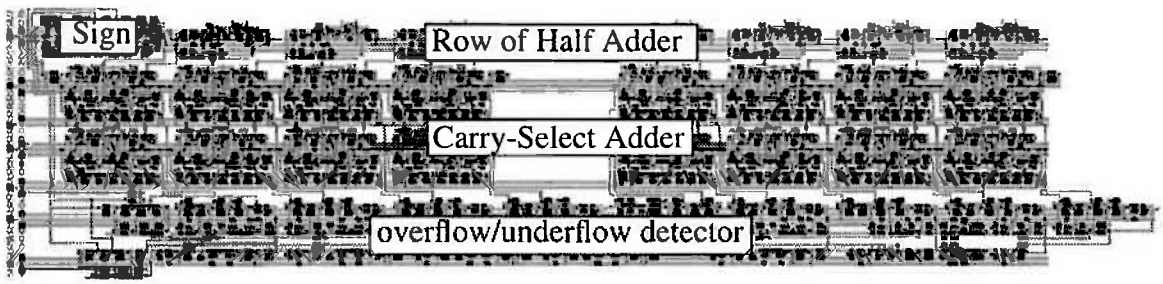


Figure 5.25 The layout of the exponent block.

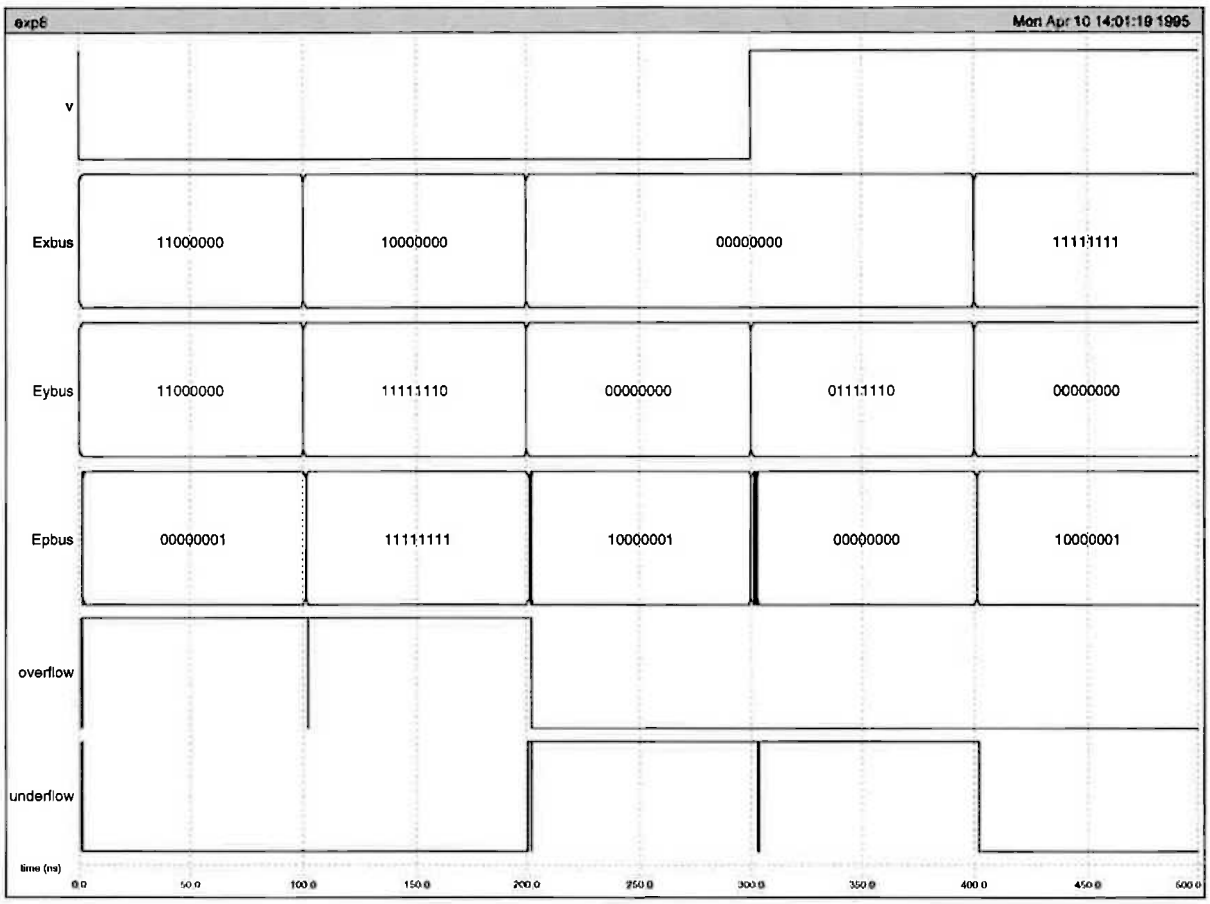


Figure 5.26 The IRSIM results of the exponent block.

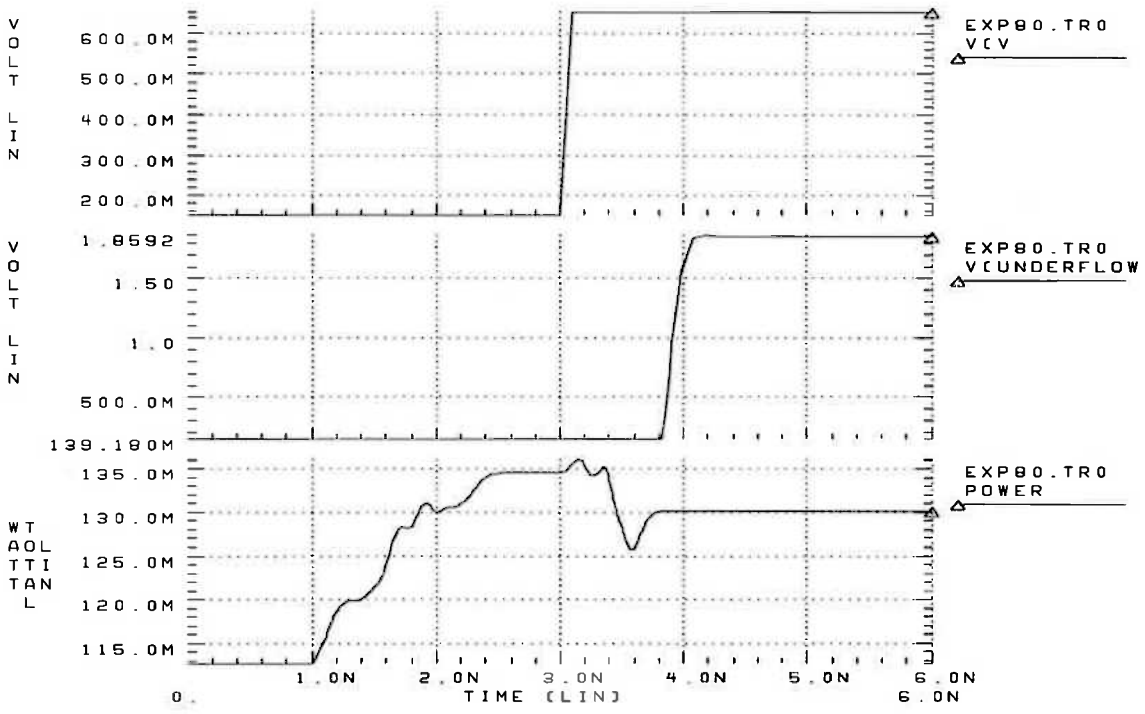


Figure 5.27 The HSPICE simulated results of the exponent block.

5.4.5 The floating point multiplier chip

1. Final Floorplan

Having described the implementation of all modules of the floating point multiplier, it is time to put everything together. The floorplan of the multiplier is illustrated in Figure 5.28. The complete layout, in MAGIC, is shown in Figure 5.29. It corresponds exactly to the floorplan of Figure 5.28. The area is *2.43mm* by *3.77mm* (without pads). The chip uses *28,000* transistors which gives *3956 transistors/mm*^{2} for 0.8 μm GaAs technology.

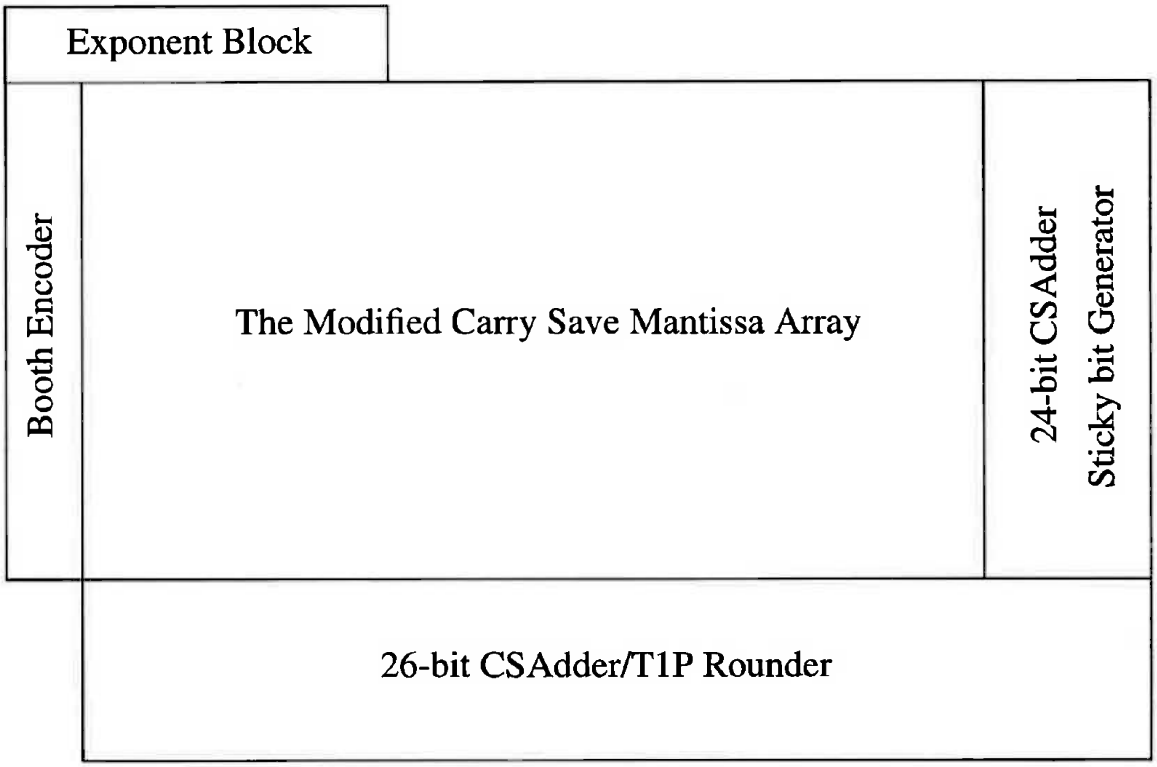


Figure 5.28 The final floorplan of the floating point multiplier.

2. Simulations

The function of the floating point multiplier was checked using IRSIM. 40 different sets of input vectors were applied with correct results. Figure 5.31 shows four IRSIM results including the worst case. The worst case delay of the multiplier equals to the sum of the worst case delays of the mantissa array, the final adder/rounder and the exponent block. Namely, all the Y inputs are high and all the X inputs are low, then X_0 is from high to low, which causes a carry rippling through the final adder/rounder and an overflow v in the mantissa. The latter causes an underflow as mentioned in exponent section. Figure 5.30 shows the HSPICE simulated results for the worst case. The simu-

lated delay of the chip is *4ns* at *typical-typical* (*tt*) parameters with 100fF load. The power dissipation is 3.5W at 75°C with 2V power supply giving 14mW/MHz.

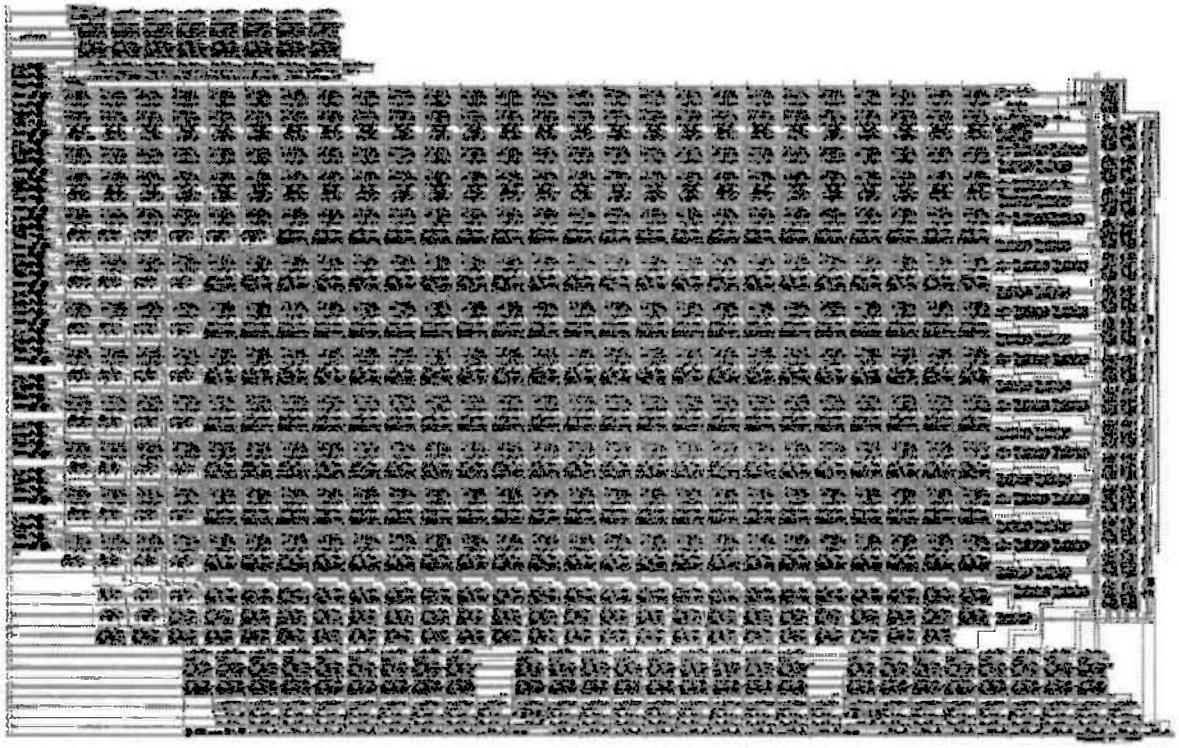


Figure 5.29 The final layout of the floating point multiplier.

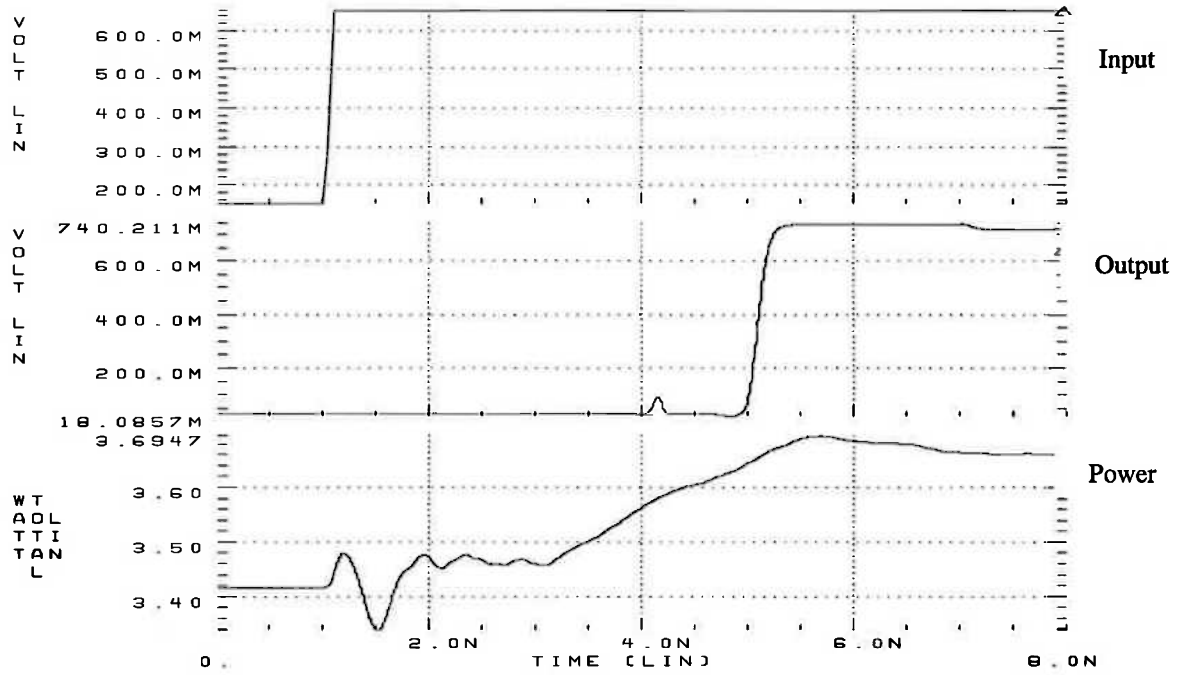


Figure 5.30 The HSPICE simulated results.

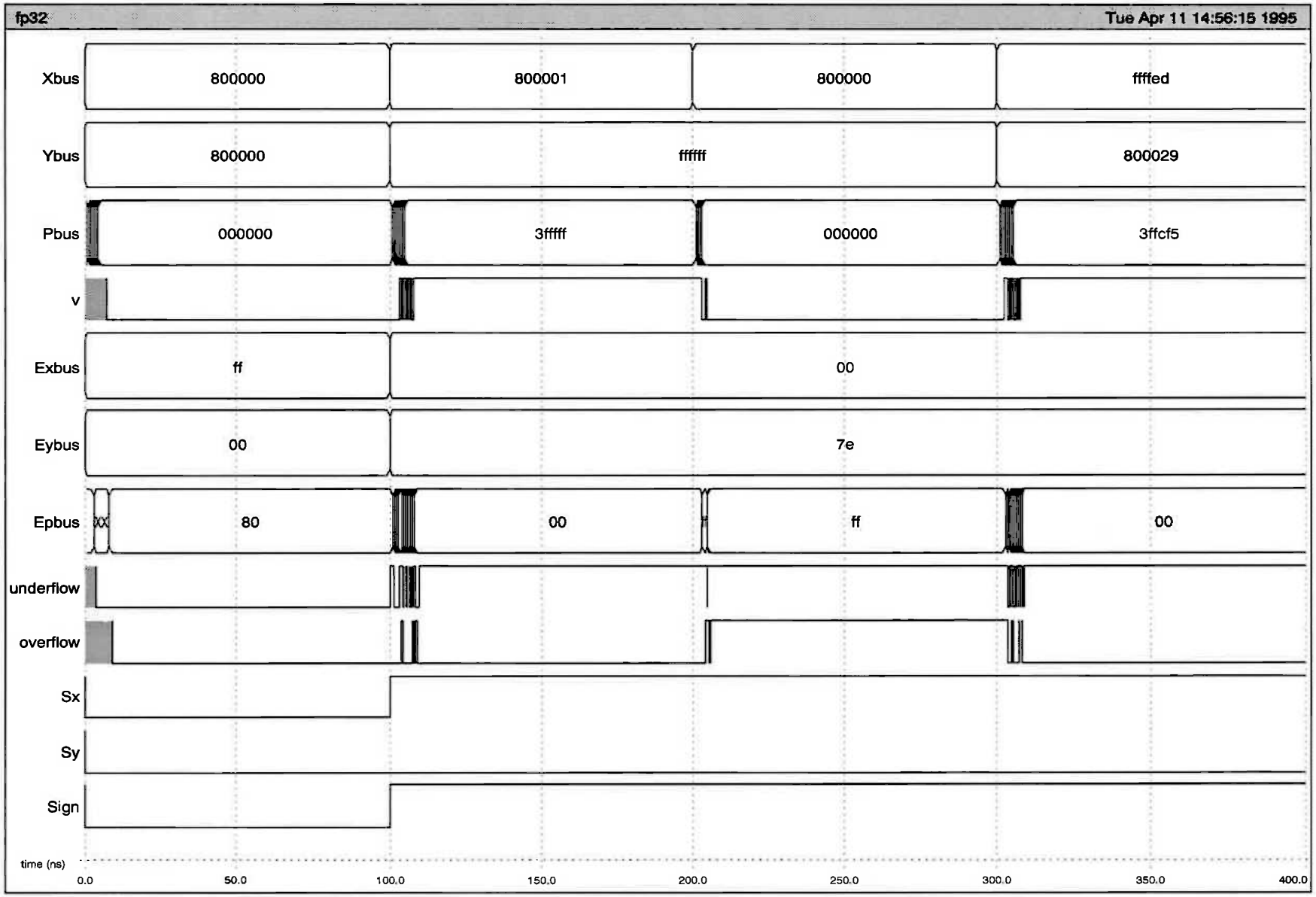


Figure 5.31 The IRSIM results of the floating point multiplier.

3. Performance comparison

Table 5.9 shows the comparison of the 32-bit floating point multipliers (U of A) with the AT&T GaAs heterojunction FET 32-bit multiplier [98]. It can be seen that our chip has less than half delay, uses about 12% area and saves half power comparing to AT&T chip. Since the AT&T chip is the only other 32-bit multiplier reported in this field, a 16×16-bit fixed point multiplier has also been designed and implemented with the same techniques used in the floating point multiplier in order to compare with other GaAs technologies. Table 5.10 shows the comparison of our 16×16-bit multiplier with other 16×16-bit multipliers. It can be seen that our chip has the best performance in terms of delay, area and power dissipation.

Table 5.9 The performance of 32-bit multipliers

	Technology	Format	Method	Delay *ns*	Area mm^2	Transistors	Density $/mm^2$	Power *W*	mW/ MHz
AT&T	GaAsHFET 1μm	IEEE32	(4, 2)	9.25	8×9.5	49,700	654	7	74.75
U of A	GaAsMESFET 0.8μm	IEEE32	m. CS	4	2.43×3.77	28,000	3056	3.5	14

Table 5.10 The Performance of 16×16 Multipliers

Authors	Technology	Method	Delay *ns*	Area mm^2	FETs	Density $/mm^2$	Power *W*	mW/ MHz
Sekiguchi Sumitomo(1990)	GaAs (0.7μm)	m. Booth, WT multi-chip	7.6	4 X 3.75 X 3.75	4 X 3906	278	4 X 1.1	33.4
Singh et al ITT(1990)	GaAs (0.7μm)	CS	4.75	3.2 X 2.8	11708	1307	2.61	12.4
Kajii et al (1988)	GaAs HEMT		4.1	6.3 X 4.8	15000	496	6.2	25.4
Cui et al U of A (1994)	GaAs (0.8μm)	m. Booth & m. CS	2.93	1.4 X 2.5	11562	3300	1.44	4.2

5.5 Summary

A novel GaAs VLSI layout approach, called the Modified Ring-Notation approach, is described. This approach maintains the advantages of the original Ring Notation while reducing layout area and delay.

The floating point multiplier presented in the previous chapter was implemented by the new layout approach. The combination of the fast arithmetic architecture and compact layout style achieves *4ns* multiplication time with *3.5W* power dissipation at *75°C* giving *14 mW/MHz*. The area of the chip is *2.43mm by 3.77mm* (excluding pads)

and uses *28,000* transistors to give a density of *3056 transistors/mm*^{2} for 0.8-μm GaAs technology. It has the best performance amongst the current designs, which suggests that the our floating point architecture and the new layout approach work very well for the GaAs technology.

From the implementation of the 32-bit single precision multiplier, it can be estimated that it will take *7.6ns* to performance 64-bit double precision floating point multiplication if the same architecture is used.

CHAPTER

6

A 16×16-bit Fixed Point Multiplier Chip

A 16×16-bit fixed point multiplier is described in this chapter. This chip was fabricated using VITESSE H-GaAs-II 0.8μm Technology. It consists of a 16×16-bit multiplier, a TDFL test circuit and CMOS compatible pad drivers and receivers. The purpose of the chip is to economically verify the architecture of the multiplier, the Modified Ring Notation layout methodology, the dynamic/static mixed approach and the interfacing between the GaAs/CMOS/BiCMOS unified technology described in the previous chapters of this thesis.

6.1 The 16×16-bit Fixed Point Multiplier

6.1.1 The Architecture of the 16×16-bit Multiplier

The 16×16-bit fixed point multiplier uses the same architecture as the 32-bit floating point multiplier described previously, i.e. a modified carry save array is used in conjunction with Booth's algorithm to reduce the partial product addition and interconnection, and a multi-carry select adder is used to speed up the final addition. For 16×16-bit multiplication, the total number bits of the final addition is *30*. Using Eq.4.18, $m = 6$. Therefore, the multi-carry-select adder at the foot of the multiplier array is partitioned *5-5-6-7-7* as shown Figure 6.1. The area is *1.4mm* by *2.5mm* (excluding pads).

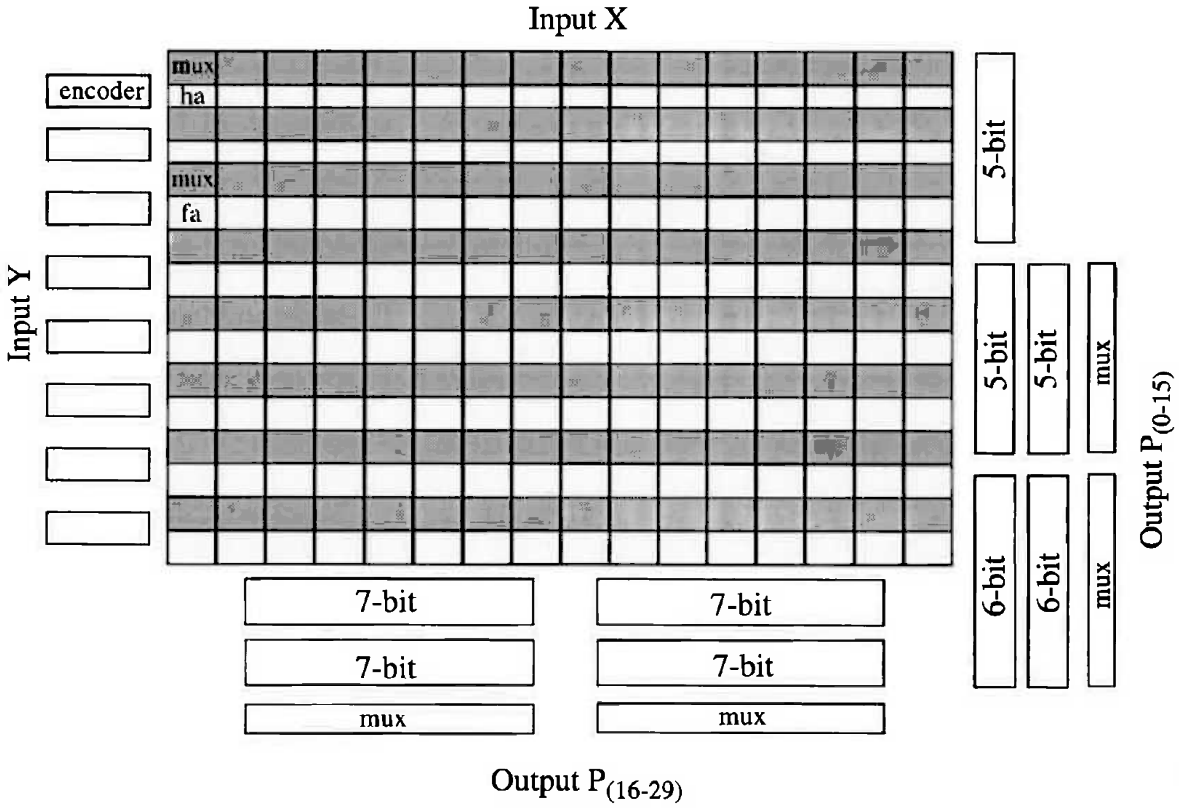


Figure 6.1 The partitioning of the 16×16-bit multiplier.

The HSPICE simulated worst case delay is shown in Figure 6.2. The worst case delay is the time taken for "*P29*" going to high with all the *Y* inputs are high, all the *X* inputs are low, and *X0* is from high to low. The worst case delay is *2.93ns* at *typical-typical* (*tt*) parameters and *75°C* temperature. The power dissipation is *1.44W* with 2V

power supply. The simulations at *slow-slow* (*ss*) and *fast-fast* (*ff*) were also carried out. The worst case delay is *6.5ns* at *ss2*, and is *2.16ns* with *2.14W* maximum power consumption at *ff2*.

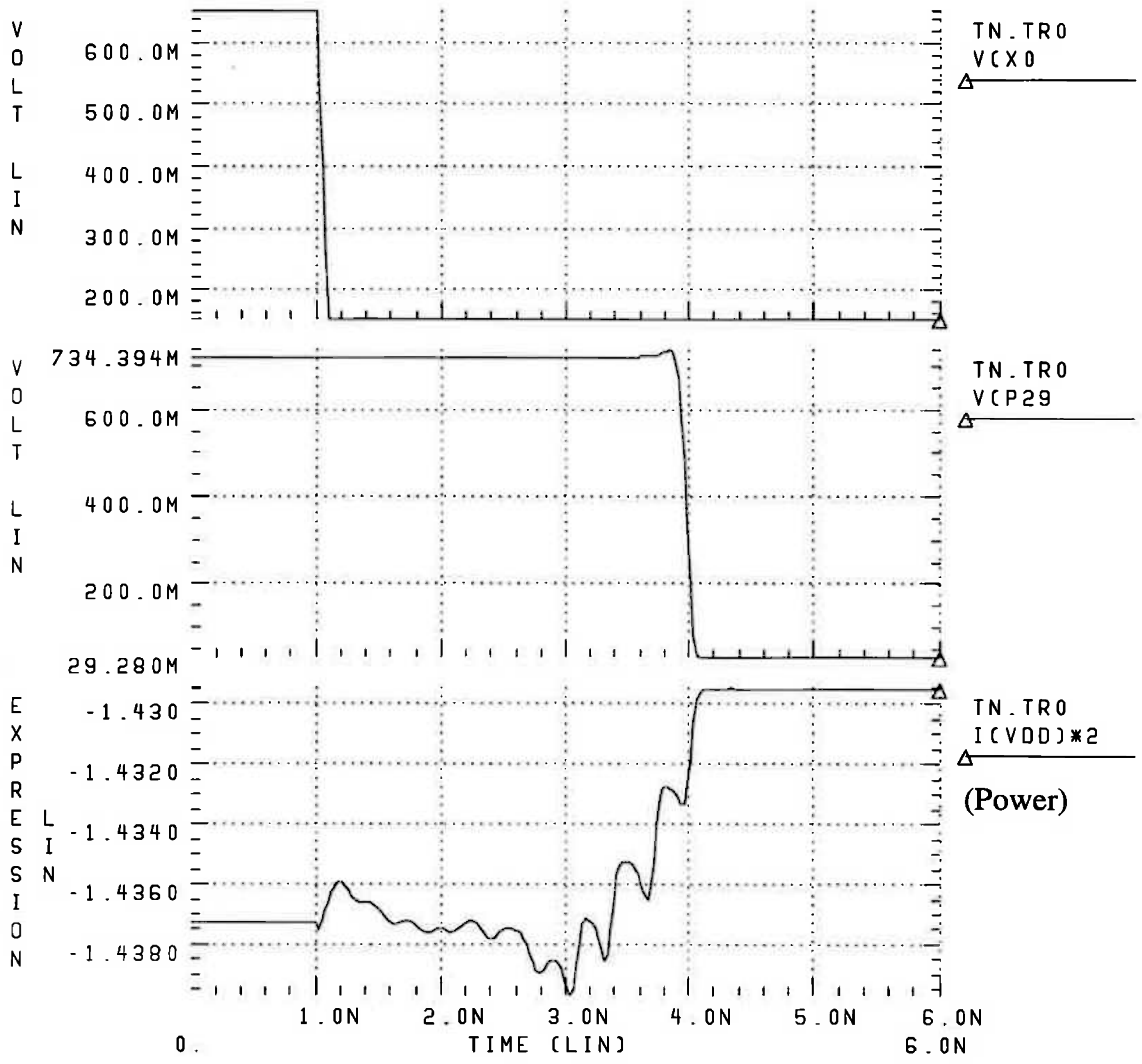


Figure 6.2 The HSPICE simulated delay of the 16×16-bit multiplier.

6.1.2 The Global Power and Ground Buses

In order to deliver clean, constant and equal power supply and ground to every device on the chip, the global power and ground buses must meet the electromigration, *ir* drop and inductance requirements. The *net-like* connection for power supply and ground described in **5.3.2** is used for this chip. The width of the M3 buses depends on

the power pad width. The number of power pads required is determined by considering the electromigration. According to the VITESSE manual, the current density for M3 is $J_{MAX} = 2.8mA/\mu m$ [27]. The bond pad width used in this chip is $96\mu m$. Therefore the maximum current flow in a power bus is $I_{MAX} = 268.8mA$. Since the maximum power dissipation is *2.14W*, and V_{dd} is *2V*, the minimum number of the power pads will be:

$$N = \frac{I_{TOTAL}}{I_{MAX}} = \frac{2.14/2}{268.8\times10^{-3}} = 3.98$$

Therefore, at least four internal power pads and four internal ground pads are needed, to satisfy electromigration requirements.

In order to reduce the *ir* drop and inductance, the placement of the global power and ground buses must be carefully considered. Figure 6.3 illustrates two arrangements of the power/ground buses for the 16×16-bit multiplier chip. Figure 6.3(a) simply uses two groups of power/ground buses as required by the electromigration consideration (**2.4.2**). Figure 6.3(b) uses four pairs of power/ground pads, but uses five groups of power/ground buses connected in parallel.

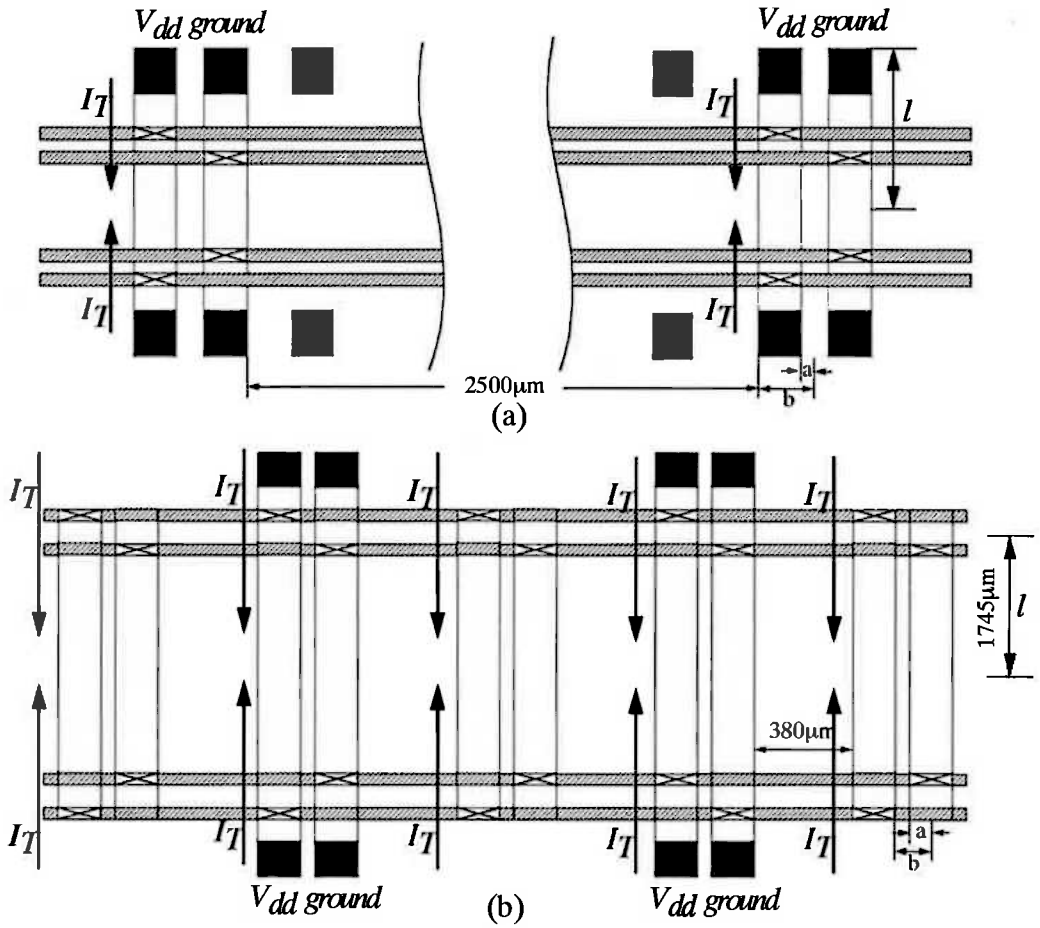


Figure 6.3 The two arrangements of the power/ground buses for the multiplier chip.

We use the interconnection models described in **2.4** to calculate the *ir* drop and inductance of each design.

1. *ir* drop

• Design (a)

According to Eq.2.27 the voltage drop at the far end of the power line is:

$$\Delta V = \frac{\rho I_T l}{2A} = \frac{\rho I_T l}{2Wt} = \frac{R_s I_T l}{2W} .$$

where R_s is layer sheet resistance.

Referring to Figure 6.3(a), current I_T per bus can be calculated as following:

$$I_T = \frac{P_o / V_{dd}}{4} = \frac{2.14/2}{4} = 267.5mA .$$

The VITESSE 132 standard padring is used for this chip which will be described later. The size of the padring is $5590\mu m$ by $3490\mu m$, and the sheet resistance of M3 is $0.025\Omega/\square$ [27]. Therefore,

$$\Delta V = \frac{R_s I_T l}{2W} = \frac{0.025 \times 267.5 \times 3490/2}{2 \times 96} = 60.78mV .$$

•Design (b)

From Figure 6.3(b), the total current I_T per bus is:

$$I_T = \frac{P_o / V_{dd}}{10} = \frac{2.14/2}{10} = 107mA .$$

Therefore, the voltage variation due to *ir* drop is:

$$\Delta V = \frac{R_s I_T L}{2W} = \frac{0.025 \times 107 \times 3490/2}{2 \times 96} = 24.31mV .$$

The voltage drop of the layout shone in Figure 6.3(b) is $24.31mV$ which is significantly better than the $60.78mV$ of Figure 6.3(a).

2. Inductance

- Design (a)

According to the VITESSE manual, the minimum space between two M3 wires is 6μm. Therefore, in Figure 6.3(a) $a = 6/2 = 3\mu m$ and $b = W_{pad} + a = 96 + 3 = 99\mu m$. Because the two power and ground buses in Figure 6.3(a) are completely separated from another two power and ground buses ($2500\mu m > 10b$), the CPS model can be applied.

From Eq.2.14 and Figure 2.22, $k = \frac{a}{b} = \frac{3}{99} = 0.03$, $\frac{K(k)}{K'(k)} = 0.32$. Using Eqs.2.11, 2.16 and 2.16 with $\varepsilon_r = 1$, the effective dielectric constant can be determined as following:

$$\varepsilon_{eff} = 1 + \frac{\varepsilon_r - 1}{2} = 1,$$

$$Z_0 = 120\pi \frac{K(k)}{K'(k)} = 120\pi \times 0.32 = 120.6\Omega,$$

$$C_0 = \frac{1}{Z_0 c} = \frac{1}{120.6 \times 3 \times 10^8 \times 10^6} = 0.0276 fF/\mu m.$$

For minimum design rules, the capacitance of M3 to substrate, $C_{M3,}$ is:

$$C_{M3} = C_{pp} + 2C_f = 5 \times 0.022 + 2 \times 0.035 = 0.18 fF/\mu m \text{ [27].}$$

The ratio of the capacitance gives

$$\varepsilon_{eff} = \frac{C_{M3}}{C_0} = \frac{0.18}{0.0276} = 6.5.$$

Then the actual impedance and inductance can be determined from Eqs.2.11 and 2.10 using $\varepsilon_{eff} = 6.5$:

$$Z_0 = \frac{\sqrt{\varepsilon_{eff}}}{C_{M3} c} = \frac{\sqrt{6.5}}{0.18 \times 10^{-15} \times 3 \times 10^8 \times 10^6} = 47\Omega,$$

$$L_0 = Z_0^2 C_{M3} = 47^2 \times 0.18 \times 10^{-15} = 401 fH/\mu m = 4.01 nH/cm.$$

Because the centre of the power/ground buses is connected to the V_{dd} or ground by two parallel paths, the net inductance to the centre is reduced by half:

$$L = \frac{1}{2} L_0 l = \frac{1}{2}(4.01) \times \left(\frac{3490}{2}\right) \times 10^{-4} = 0.35 nH.$$

If the chip is connected to its circuit board or chip carrier by bond wires, an additional *0.8nH* is added in series with each bond pad [25]. Therefore, the total inductance of one power/ground bus is:

$$L_T = \frac{1}{2}\left[\left(\frac{3490}{2} \times 10^{-4} \times 4.01\right) + 0.8\right] = 0.35 + 0.4 = 0.75 nH.$$

•Design (b)

In Figure 6.3(b), $a = W_{pad}/2 = 96/2 = 48\mu m$ and b = spacing of M3 + $a = 6 + 48 = 54\mu m$. Because the spacing between the two power and ground buses and their neighbours is not significant ($380\mu m < 10b$), the CPW model and Eq.2.12 can be applied as follows:

$k = \frac{a}{b} = \frac{48}{54} = 0.89$ and $\frac{K(k)}{K'(k)} = 1.34$. Therefore, when $\varepsilon_r = 1$

$$Z_0 = 30\pi \frac{K'(k)}{K(k)} = \frac{30\pi}{1.34} = 70.3\Omega,$$

$$C_0 = \frac{1}{Z_0 c} = \frac{1}{70.3 \times 3 \times 10^8 \times 10^6} = 0.0474 fF/\mu m.$$

The ratio of C_{M3} with C_0 gives

$$\varepsilon_{eff} = \frac{C_{M3}}{C_0} = \frac{0.18}{0.0474} = 3.8.$$

Then the actual impedance and inductance can be determined from Eqs.2.11 and 2.10 using $\varepsilon_{eff} = 3.8$:

$$Z_0 = \frac{\sqrt{\varepsilon_{eff}}}{C_{M3}c} = \frac{\sqrt{3.8}}{0.18\times10^{-15}\times3\times10^{8}\times10^{6}} = 36\Omega,$$

$$L_0 = Z_0^2 C_{M3} = 36^2\times0.18\times10^{-15} = 233fH/\mu m = 2.33nH/cm.$$

Because five V_{dd}/ground buses are connected parallel, and they share four V_{dd}/ground bond pads, each bus net inductance to the centre is reduced by one fifth:

$$L = \frac{1}{2}L_0 l = \frac{1}{5}(2.33)\times\left(\frac{3490}{2}\right)\times10^{-4} = 0.08nH$$

The total inductance of one power/ground bus is:

$$L_T = \frac{1}{5}(0.8+L) = \frac{1}{5}(0.8+0.08) = 0.176nH.$$

By comparing design (a) and design (b), it can be seen that the latter has 40% less voltage variation and 23% less inductance than the former. In order to keep the inductive voltage spike below *50mV*, the maximum current slew rate for design (a) is $dI/dt = V/L = 0.05/0.75 = 6.67\times10^{7}A/s$ or $\Delta I = 6.67mA$ out of a total of *100mA* in *100ps*. For design (b) $dI/dt = 0.05/0.176 = 28.4\times10^{7}A/s$ or $\Delta I = 28.4mA$ out of a total of *100mA* in *100ps*. Because DCFL steers current between the switch FET and the input gate-source diode of the next stage, the I_{DD} and ground current is relatively constant. However, a fractional change in current below 6.67 percent is difficult to achieve, while below 28.4% is much easier and possible.

Therefore, while both Figure 6.3(a) and (b) satisfy electromigration requirement, the latter has 40% less voltage variation and 23% lower voltage spike due to *ir* drop and inductance, the design in Figure 6.3(b) is chosen for this multiplier chip.

The currents driven by the transistors on the M2 buses between two M3 buses are from 10mA to 16mA. The electromigration current density for M2 is 2.0mA/μm. Therefore the width of the M2 buses ranges from 5μm to 8 μm. The ir drop and induction are not problems for M2 buses, due to the net parallel structure which reduces the whole resistance and inductance.

6.2 The TDFL Test Circuit

In order to test the dynamic/static mixed approach described in Chapter 3, a TDFL test circuit is implemented in this 16×16-bit multiplier chip. Since the mixed approach depends on dynamic (TDFL) operations, only TDFL test circuits are put on the chip due to restriction on the area and the number of pads. The TDFL test circuit contains a 24-bit TDFL shift register, an on chip clock generator and four probe pads for high speed testing as shown in Figure 6.4.

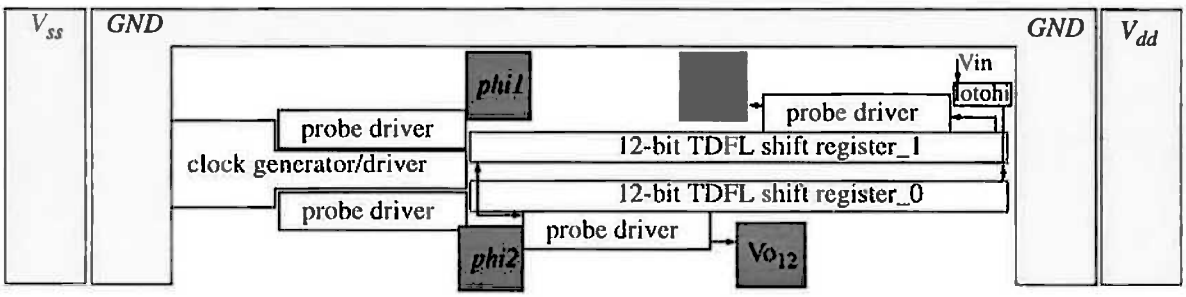


Figure 6.4 The structure of the TDFL test chip.

6.2.1 The 24-bit TDFL Shift Register

A 24-bit simple TDFL shift register designed in **2.3.6** was simulated under a range of conditions to test if the design is robust enough to withstand manufacturing variation.

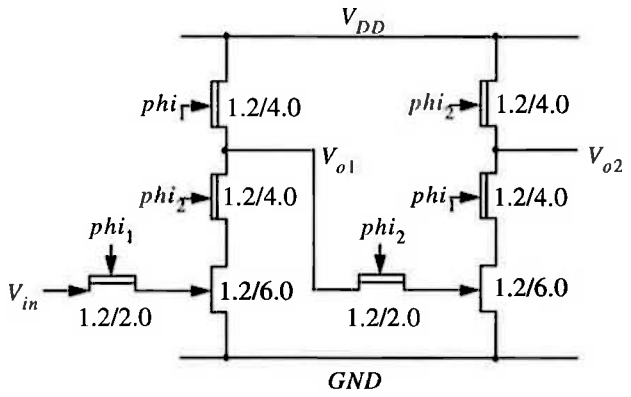


Figure 6.5 TDFL shift register stage.

The 24-bit shift register with TDFL register sizing shown in Figure 6.5 is expected to work properly with process spreads from *ss2* to *ff2* at highest clock frequency around 600MHz.

6.2.2 On Chip Clock Generator

Since the maximum frequency of the 24-bit TDFL shift register is around 600MHz, there are 12 inverters and one *nor* gate connected with output to the input of next stage in the on chip clock generator. There is a reset signal connected to the *nor* gate to start the ring oscillation. The structure of the on chip clock generator is illustrated in Figure 6.6.

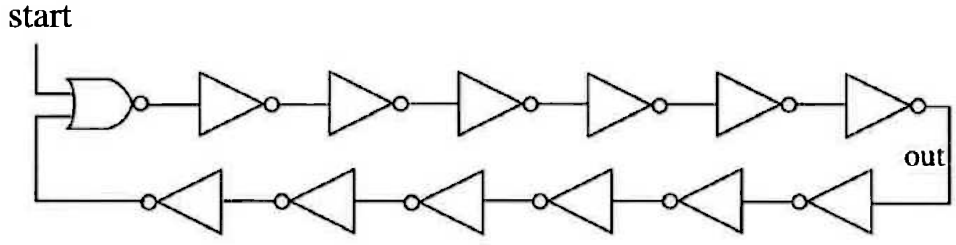


Figure 6.6 The structure of the on chip clock generator.

The output of the clock generator is connected to the two phase non-overlapping clock generator/driver described in **3.2.2** for the 24-bit TDFL register.

6.2.3 Probe Pads

The chip uses CMOS compatible I/O which will be described later. These pads are too slow to test the TDFL circuit. Therefore, probe pads are used for testing the outputs of the TDFL circuit. There are four probe pads, two for the two non-overlapping clocks to test the frequency, another two for the outputs of the shift registers at the 12th register and 24th register to test the function of the TDFL register. The pad driver uses the same clock driver as described in Figure 3.32.

6.2.4 The low_to_high (lotohi) Input

The output level form the CMOS pad receiver is from -1.4V to -0.85V. However, the input level required by the TDFL circuit is from 0V to 0.6V. Therefore, a *lotohi* input buffer is used to convert to the desired voltage level. Such a circuit can be easily achieved by connecting a DCFL inverter with a DFET buffer as illustrated in Figure 6.7.

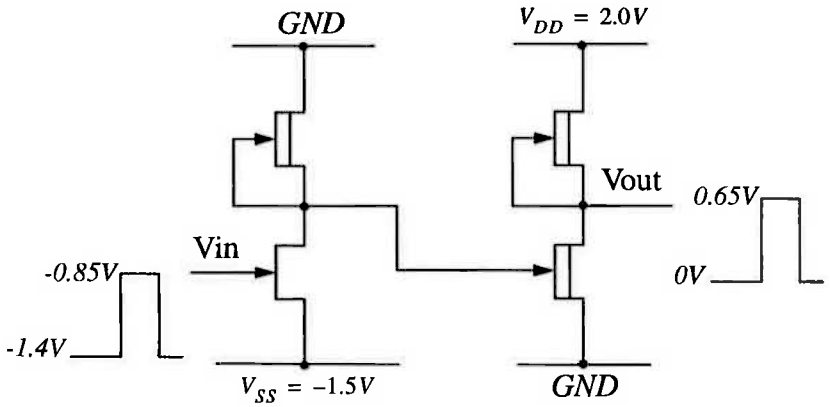


Figure 6.7 The *lotohi* buffer.

6.2.5 The HSPICE Simulated Results

The HSPICE simulated result of the TDFL test circuit is shown in Figure 6.8 at *typical-typical* parameters and 75°C temperature. The clock frequency is 523MHz. The circuit works also from *ss2* to *ff2* with frequency from 330MHz to 600MHz.

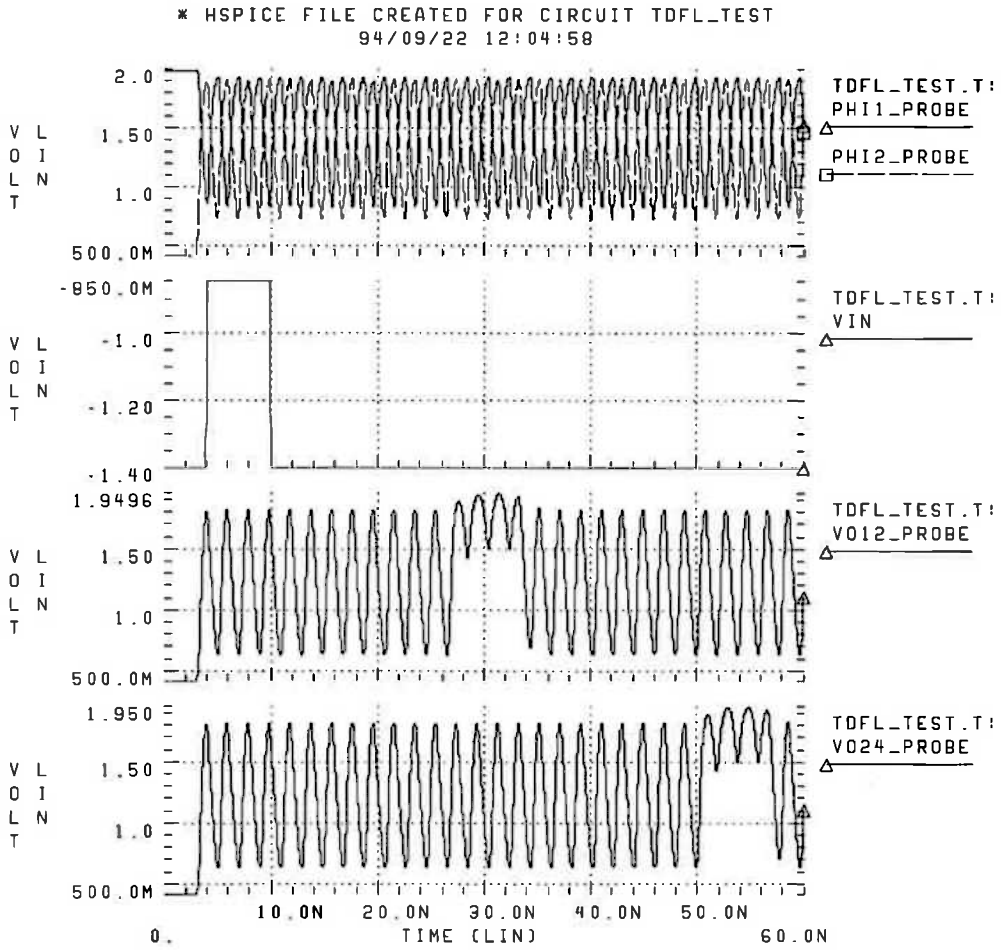


Figure 6.8 The HSPICE simulated results of the TDFL testing circuit.

6.3 The Test Chip

6.3.1 The CMOS Compatible Pad Driver and Receiver

In order to test the interface between GaAs chips and CMOS chips for the solid modelling accelerator, the VITESSE VCB50K Library's CMOS compatible pads *TTL I/O* [116] are used in the 16×16-bit multiplier chip.

The TTL input buffer (IT0QD1) in the VITESSE TCB50K Standard Cell is used as the CMOS-to-GaAs receiver. Figure 6.9 illustrates the schematic of the circuit.

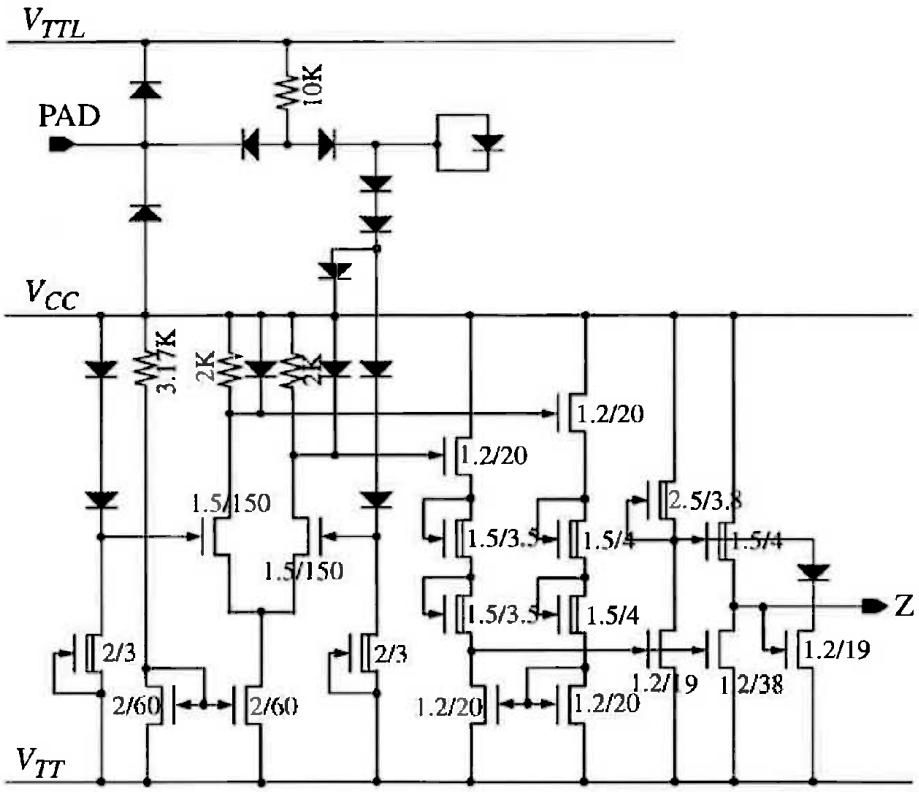


Figure 6.9 CMOS-to-GaAs receiver schematic.

The circuit was designed for $V_{TTL} = 5V$, $V_{CC} = 0V$ and $V_{TT} = -2V$. In this case, the backgating voltage has to be set to -1.6V, which will seriously degrade the speed of the whole chip. Therefore, $V_{TTL} = 7V$, $V_{CC} = 2V$ and $V_{TT} = 0V$ are used instead without affecting the function of the receiver. The HSPICE simulated results of the receiver are shown in Figure 6.10. It has a logic swing of *0.66V* to *0.1V*, which is suitable for DCFL and SDC FL logic families. The rise time is *0.685ns*, and the fall time is *1.06ns*

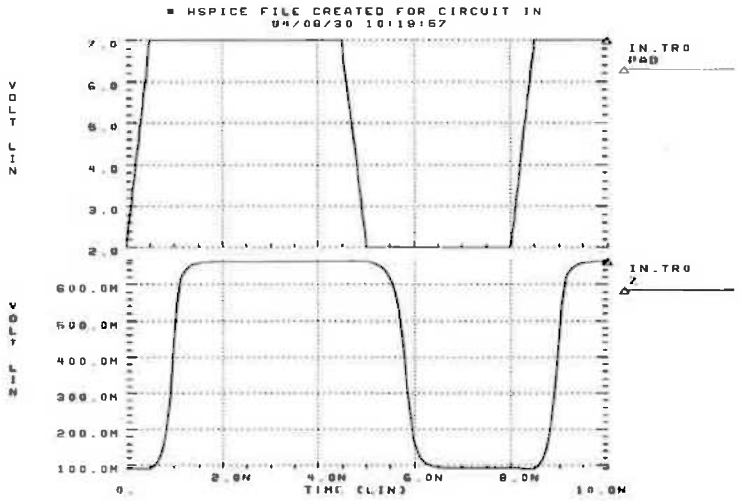


Figure 6.10 The HSPICE simulated results of the CMOS-to-GaAs receiver.

The TTL output buffer (OT00D1) in the VITESSE TCB50K Standard Cell is used as the GaAs-to-CMOS driver. The schematic is illustrated in Figure 6.11. As the receiver, the power supplies are also shifted up 2V to get positive supplies. Figure 6.12 shows the HSPICE simulations. It has a logic swing of *4.55V* to *0V* when driving a single CMOS input. The rise/fall times are *4.18ns* and *2.74ns*, respectively.

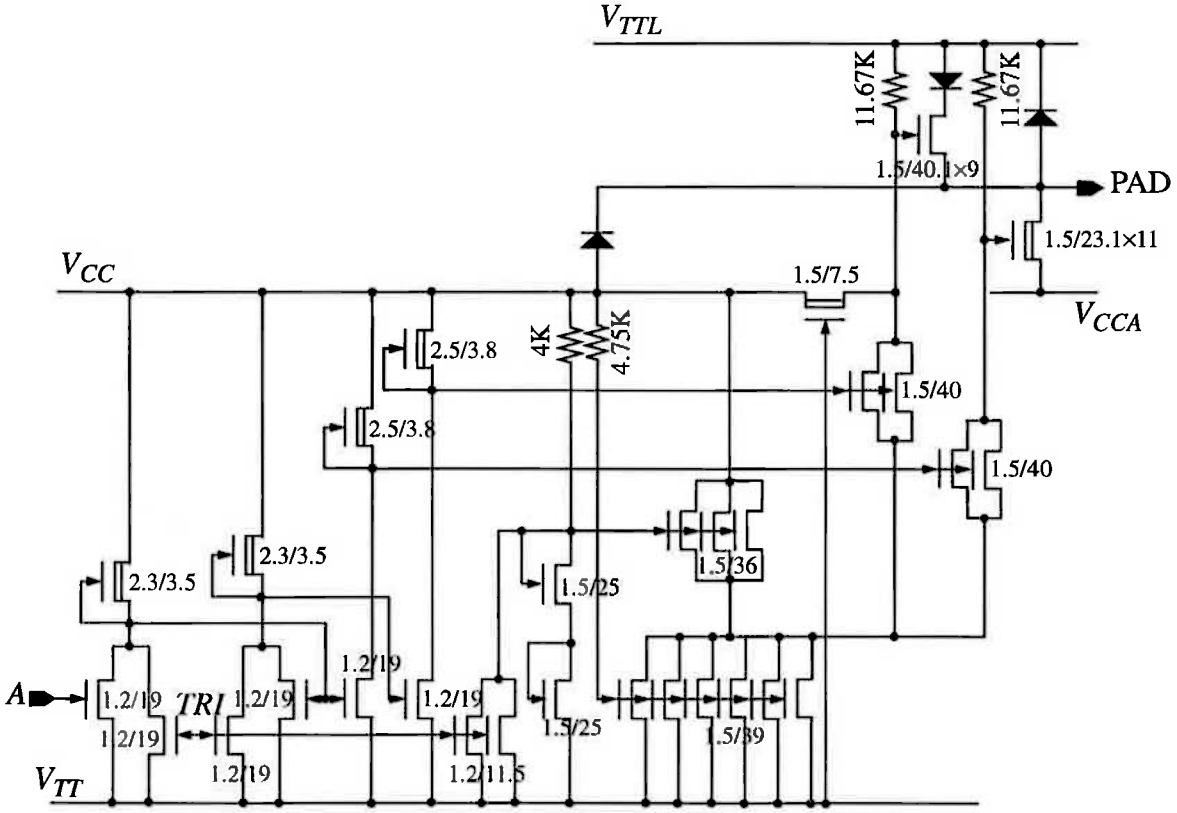


Figure 6.11 GaAs-to-CMOS driver schematic.

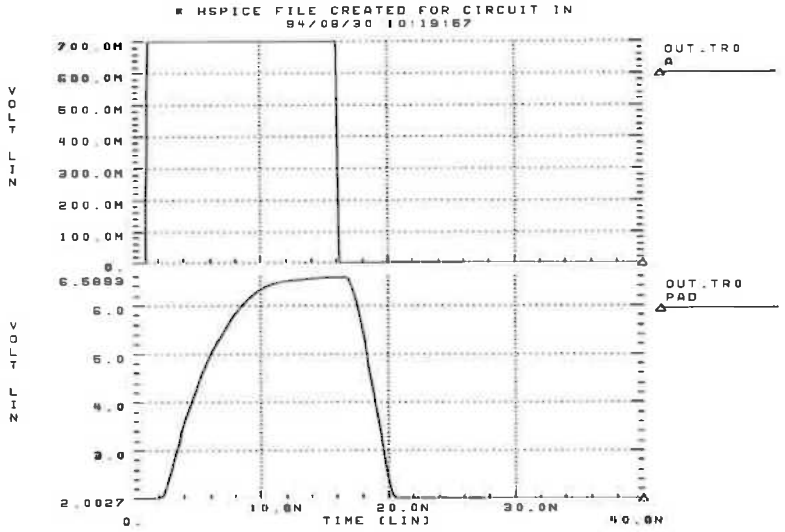
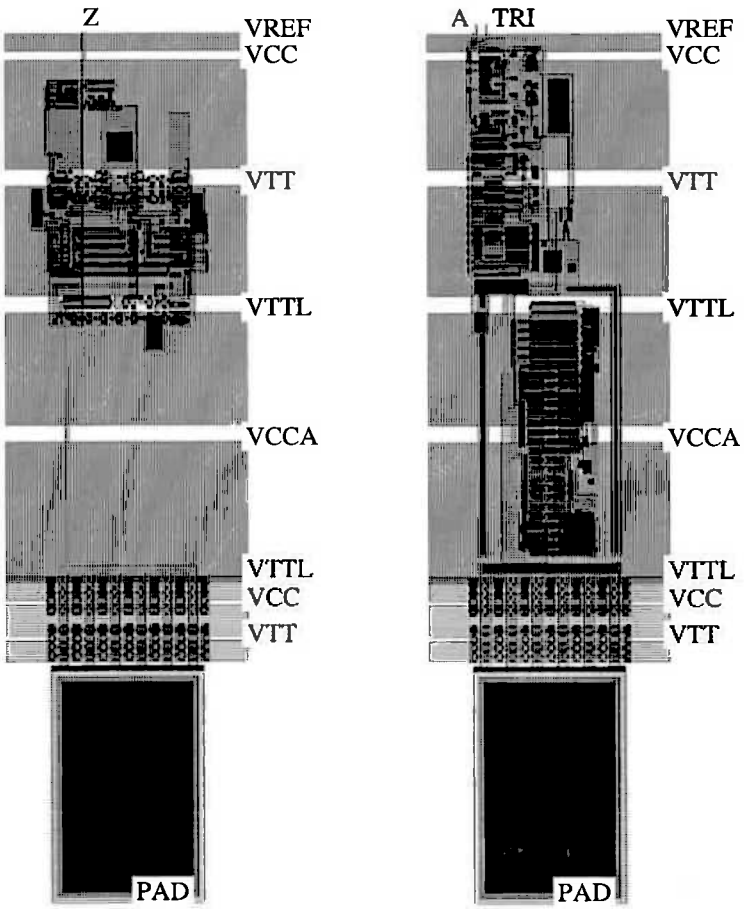


Figure 6.12 The HSPICE simulated results of the GaAs-to-CMOS driver.

Finally, the layouts of the receiver and driver are shown in Figure 6.13. The size of the passivation opening over the bonding area is 96 × 148 μm.



(a) CMOS-to-GaAs receiver (b) GaAs-to-CMOS driver

Figure 6.13 The layouts of (a) receiver, (b) driver.

6.3.2 The Chip Layout

The floorplan and the layout of the whole chip with the VITESSE 132 padring are shown in Figure 6.14 and 6.14, respectively. The pads are numbered clockwise from the arrowed corner of the padring.

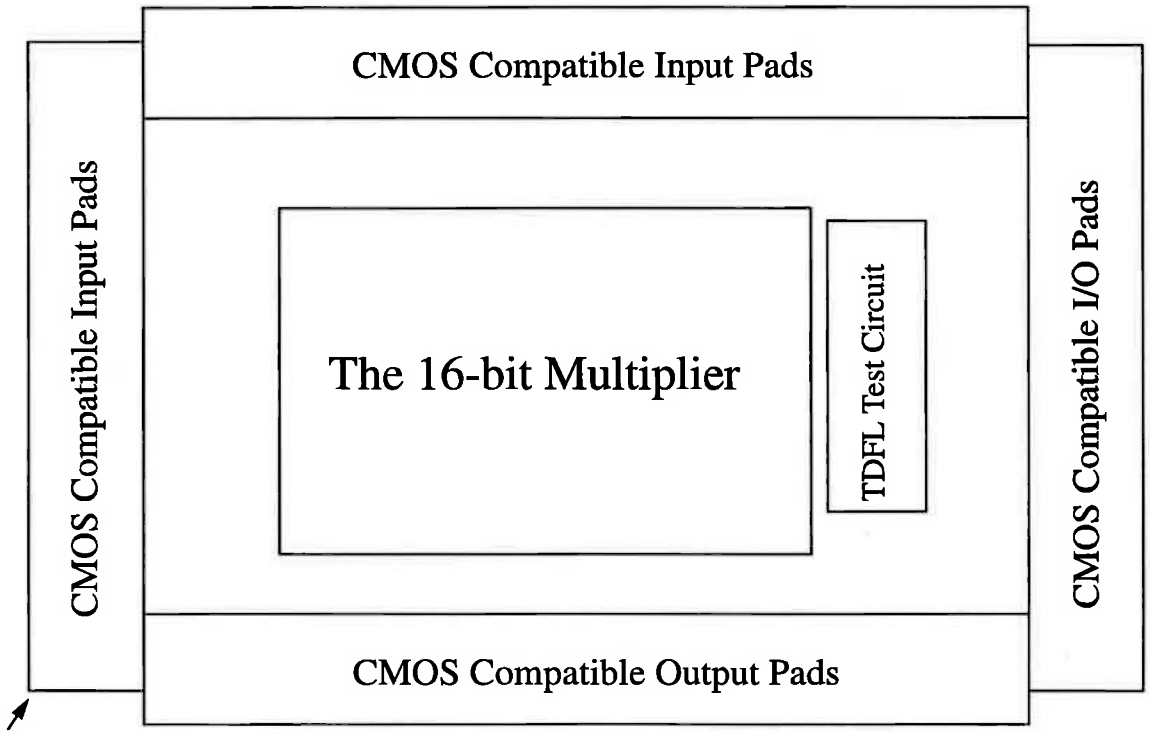


Figure 6.14 The floorplan of the 16×16 bit multiplier chip.

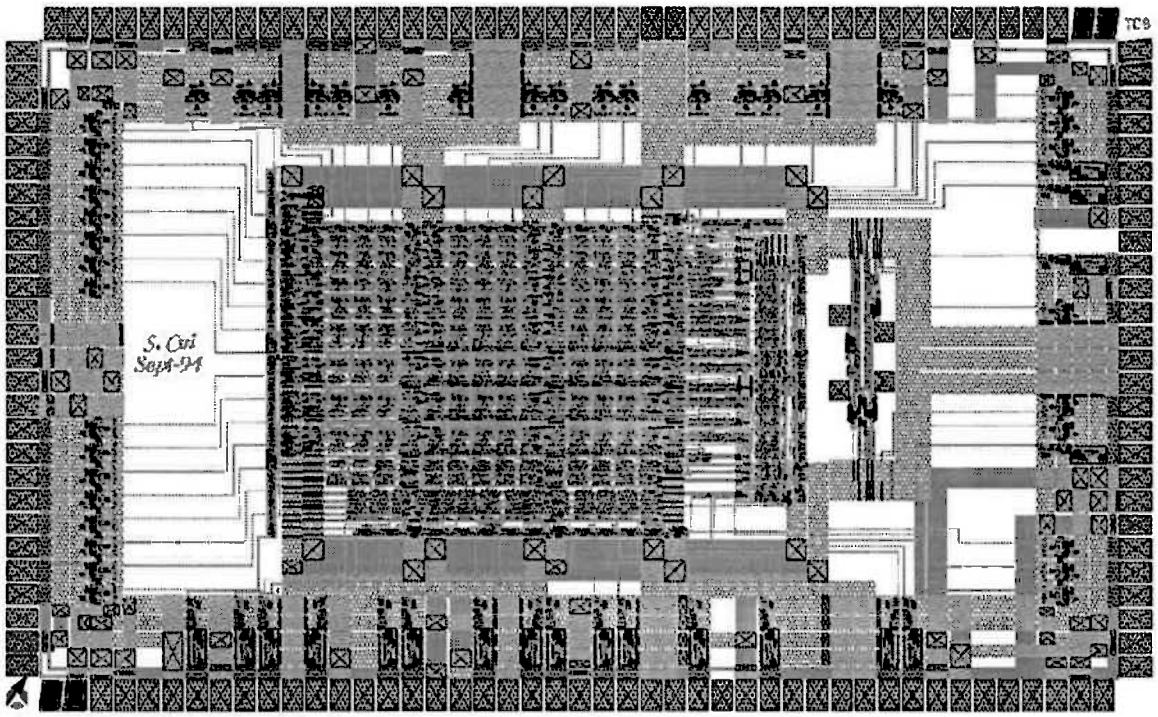


Figure 6.15 The final layout of the 16×16 bit multiplier.

6.3.3 Packaging The Chip

The chip is packaged using the VITESSE 132 pin ceramic LDCC package which offers 132 leads and 92 signal lines. The package is a high speed multilayer ceramic package specifically developed for the requirements of very high performance IC's with Cu-W heat spreaders for efficient cooling.

Signals are carried on a 50Ω controlled impedance transmission line between the package leads and the cavity bond pads resulting in good cross talk control, low impedance power supply leads and low thermal resistance from junction to case. Multiple ground pins provide for excellent signal isolation. Internal ground, power planes and decoupling capacitors minimize switching noise on the power supplies.

Figure 6.16 illustrates the 132 LDCC package. The pins are numbered anti-clockwise from the angled corner of the package body (when viewed from above). The chip pad assignments and their corresponding package pins are shown in Table 6.1 with the function of the pin and the name of the package line. All V_{dd}s, V_{cc}s and V_{cca}s are 2 volts. All V_{tt}s are 7 volts for the multiplier and 5 volts for the TDFL test circuit. All V_{tt}s are 0 volts or grounded. Clk1 and Clk2 are external clock inputs for the multiplier. All grounds are connected.

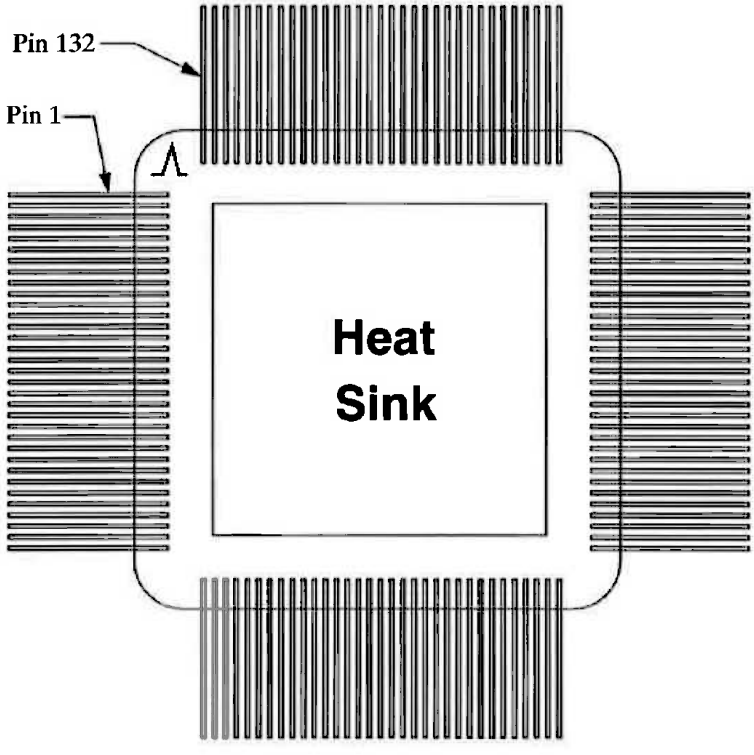


Figure 6.16 The 132 pin ceramic LDCC package.

Table 6.1 Assignment of pins and their corresponding pads.

Package Pin	Signal Name	Chip Pad	Function	Package Pin	Signal Name	Chip Pad	Function
1	-	140	-	67	-	68	-
2	s1	141	Vcc	68	-	69	-
3	GND	142	Vtt	69	GND	70	Vtt
4*	GND	143,144	Vtt	70*	GND	71,72	-
5**	s2	1,2	Vcc	71**	s49	73,74	Vcc
6	-	3	-	72	s50	75	x1
7	s3	4	y15	73	s51	76	x0
8	s4	5	y14	74	s52	77	set_i
9	s5	6	y13	75	s53	78	set_o
10	s6	7	y12	76	s54	79	p0
11	s7	8	y11	77	s55	80	Vcca
12	s8	9	y10	78	-	81	-
13	s9	10	y9	79	s56	82	Vo
14	s10	11	y8	80	s57	83	Vtl5
15	s11	12	Vttl	81	s58	84	Vin
16*	GND	14	Vtt	82*	GND	86	Ground
17**	s12	13	Vcc	83**	s59	85	Vdd
18	s13	15	Vref	84	s60	87	Vss
19	-	16	-	85	s61	88	stop
20	s14	17	y7	86	s62	89	in
21	s15	18	y6	87	s63	90	out
22	s16	19	y5	88	GND	91	Vcca0
23	s17	20	y4	89	s64	92	Vt2
24	s18	21	y3	90	GND	93	Vcc0
25	s19	22	y2	91	s65	94	c
26	s20	23	y1	92	s66	95	clk2
27	s21	24	y0	93	s67	96	clr2
28*	GND	28,29	Vtt	94*	GND	100,101	Vtt
29**	s22	25,26	Vcc	95**	s68	98	Vcc
30	GND	30	Vtt	96	s69	102	Vt2
31	s23	31	Vcc	97	-	103	-
32	-	32	-	98	-	104	-
33	s24	34	clk1	99	s70	106	Vcca
34	s25	33,35	Vttl	100	s71	105,107	Vttl
35	s26	36	clr1	101	s72	108	p16
36	s27	37	x15	102	s73	109	p17
37	s28	33,35	Vttl	103	s74	105,107	Vttl
38	s29	39	x14	104	s75	111	Vdd
39	s30	40	x13	105	-	112	-

Table 6.1 Assignment of pins and their corresponding pads.

Package Pin	Signal Name	Chip Pad	Function	Package Pin	Signal Name	Chip Pad	Function
40	s31	42	x12	106	s76	114	p18
41	GND	41,44	Vtt	107	-	113,116	-
42*	GND	47	Ground	108*	GND	119	Ground
43**	s32	38	Vdd	109**	s77	110	Vdd
44	s33	43	Vttl	110	s78	115	Vttl
45	s34	45	x11	111	s79	117	p19
46	GND	46	Ground	112	GND	118	Ground
47	GND	41,44	Vtt	113	-	113,116	-
48	s35	48	x10	114	s80	120	p20
49	s36	49	x9	115	s81	121	p21
50	s37	51	x8	116	s82	123	p22
51	s38	52	x7	117	s83	124	p23
52	GND	56,59	Vtt	118	GND	128,131	Vtt
53	GND	54	Ground	119	s84	126	p24
54	s39	55	x6	120	s85	127	Vcca
55	s40	57	x5	121	s86	129	p25
56	GND	56,59	Vtt	122	GND	128,131	Vtt
57*	GND	53	Ground	123*	GND	125	Ground
58**	s41	62	Vdd	124**	s87	134	Vdd
59	s42	58	x4	125	s88	130	p26
60	s43	60	x3	126	s89	132	Vcca
61	s44	61	Vdd	127	s90	133	p27
62	s45	65,67	Vttl	128	s91	137,139	Vttl
63	s46	63	x2	129	s92	135	p28
64	s47	64	Vcc	130	s93	136	p29
65	s48	65,67	Vttl	131	s94	137,139	Vttl
66	-	66	-	132	s95	138	g29

* Vtt pins: 4, 16, 28, 42, 57, 70, 82, 94, 108 and 123 are internally connected.
** Vcc pins: 5, 29, 43, 58, 71, 83, 95, 109 and 124 are internally connected.

6.3.4 PCB Board

A custom designed test fixture was fabricated for testing of the packaged chip. The package is surface mounted to the PCB with an elastomer ring (pressure contact) to allow easy mounting and unmounting of the test chip. The board is designed to interface to the Tektronix DAS-9200 digital analyser. The connections are standard gold PCB pins with a 0.1″ pitch. The PCB is made from double-sided fibre glass which provides controlled impedance lines to the test interface pins [118]. Solder pads are provided to connect chip resistors and capacitors to either end of the board trace. The

board is 4.6″ square. The top and bottom artworks of the board are shown in Figure 6.17 and 6.18 respectively. The design parameters are:

- The signal traces are 0.015″ wide, 0.010″ spacing (0.025″ pitch).
- The board material is fibre glass (dielectric = 4), double sided 1 ounce copper. Thickness is 1/32″.
- Pins to the DAS must be a single-ground pair which can be arranged in a group of 8 or individually. The pin spacing in both directions is 0.100″. They must be gold to make reliable contact and to prevent metal migration of tin into the DAS probe tips.
- Due to the compact nature of the test board, trace lengths could not be equalized so there is a slight difference in delay between some traces.

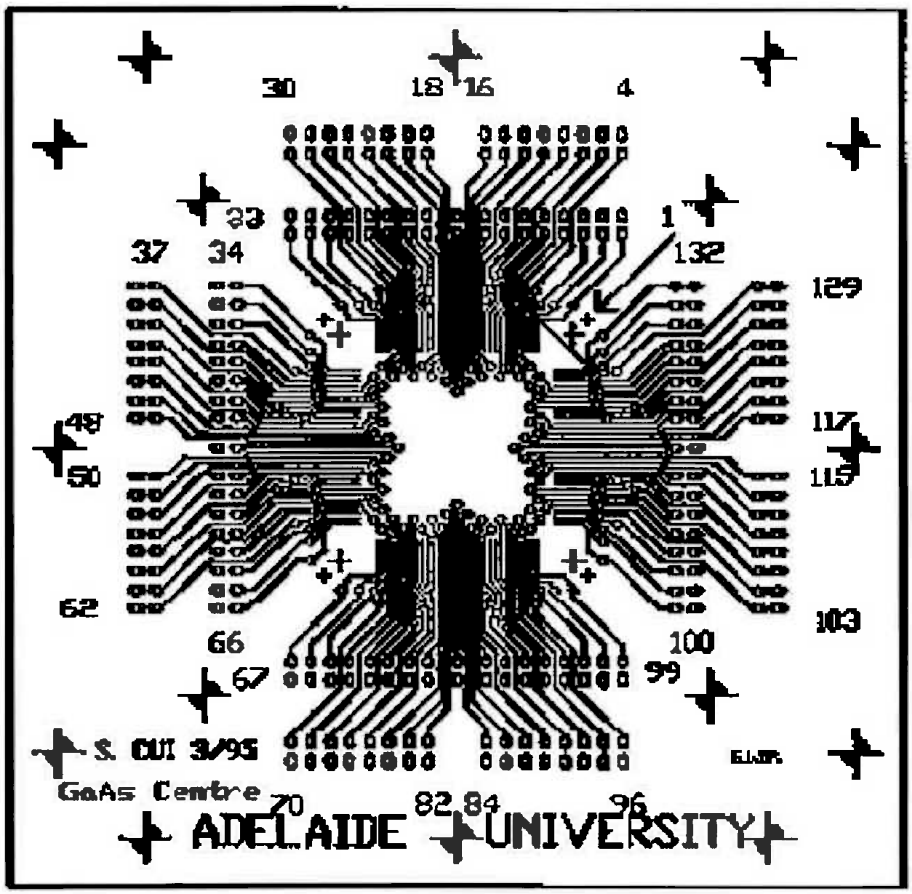


Figure 6.17 Top layer of the test fixture PCB.

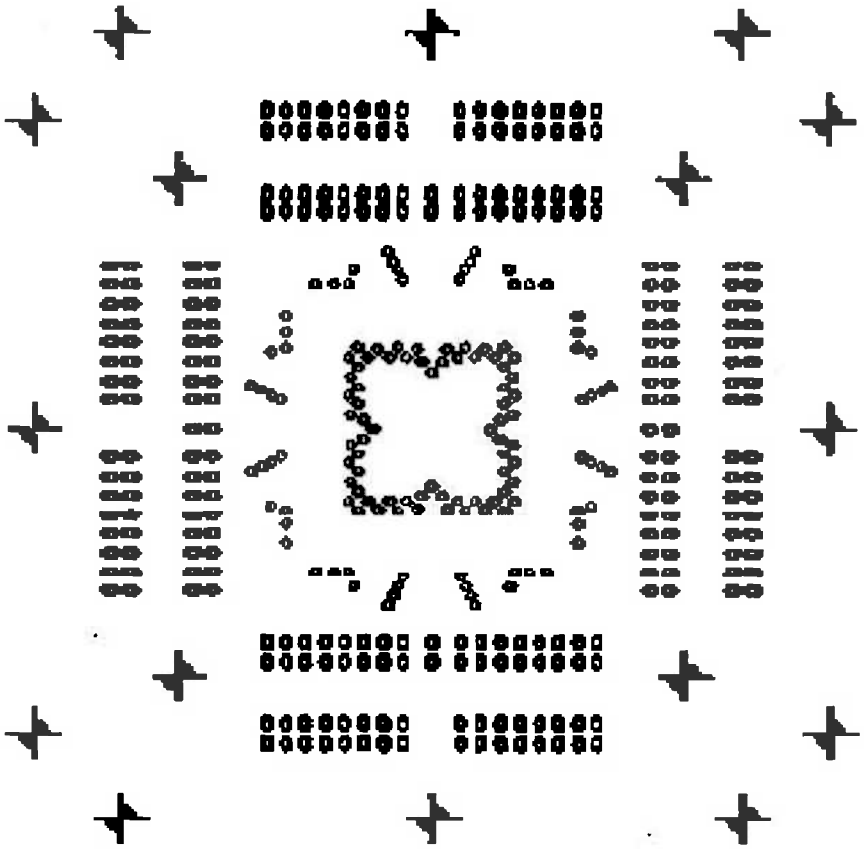


Figure 6.18 Bottom layer of the test fixture PCB.

Since GaAs components have very fast transitions, but they have CMOS output, the trace lengths must be considered as transmission lines and must be terminated correctly to retain signal integrity. The traces on the PCB are treated as lossy transmission lines. The structure can be modelled by a microstrip line as described in **2.4.1** where the backside ground plane reflects the signal to produce its dual. The impedance can be approximated by Eq.2.20 and 2.21:

$$Z_0 = \frac{60}{\sqrt{\varepsilon_{eff}}} \ln\left(\frac{8h}{w} + \frac{w}{4h}\right) \text{ and}$$

$$\varepsilon_{eff} = \frac{\varepsilon_r + 1}{2} + \frac{\varepsilon_r - 1}{2}\left(1 + \frac{10h}{w}\right)^{-1/2}.$$

where $\varepsilon_r = 4$, $w = 0.015''$ and $h = \frac{1}{32}''$. Hence $Z_0 = 100\Omega$.

Therefore, the output pads of the chip drive into a 100Ω resistor to ground. HSPICE was used to model a signal trace driven by one of the output pads through a bond wire, a source resistor R_s, a transmission line, a lead, a terminating resistor R_l in a probe. The equivalent circuit is shown in Figure 6.19.

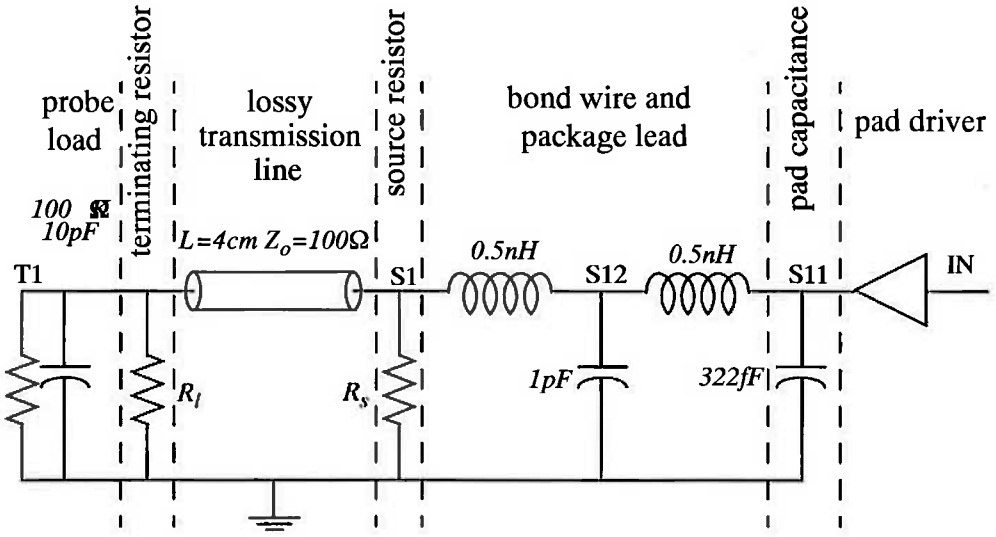


Figure 6.19 Equivalent circuit of a signal driven off chip.

The following simulations were carried out to determine the effects of various board parameters on the signals:

- a 4cm long line with no resistance and terminating resistors of 50, 75, 100, 125 and 150Ω. Figure 6.20 shows that the 100Ω load has the best damping, a satisfactory voltage swing and a reasonably good delay at the start (S1) and end of the PCB line (T1). It also can be seen from panel 3 and 4 of Figure 6.20 that the effects on the two neighbours of the test trace (from S2 to T2 and S3 to T3 respectively) is ±200mV which is acceptable comparing to a 1.6V to 3.6V voltage swing.
- a 4cm long line with a 100Ω terminating resistance for source resistance values of 50, 100, 150 and 200Ω. Figure 6.20 shows that finite source resistance gives negligible improvement in rise/fall time and cross-talk relative to large source resistance.

Therefore, from the above simulations the terminating resistance is set to 100Ω for fast rise time and the source resistance is set to infinity (i.e. open circuit) for best voltage swing. The delay from the input to the PCB pin is measured to be about 4.15*ns*. The output voltage swing is about 3.1V. The terminating resistance can be increased to obtain a larger swing if required at the expense of slower rise and fall times.

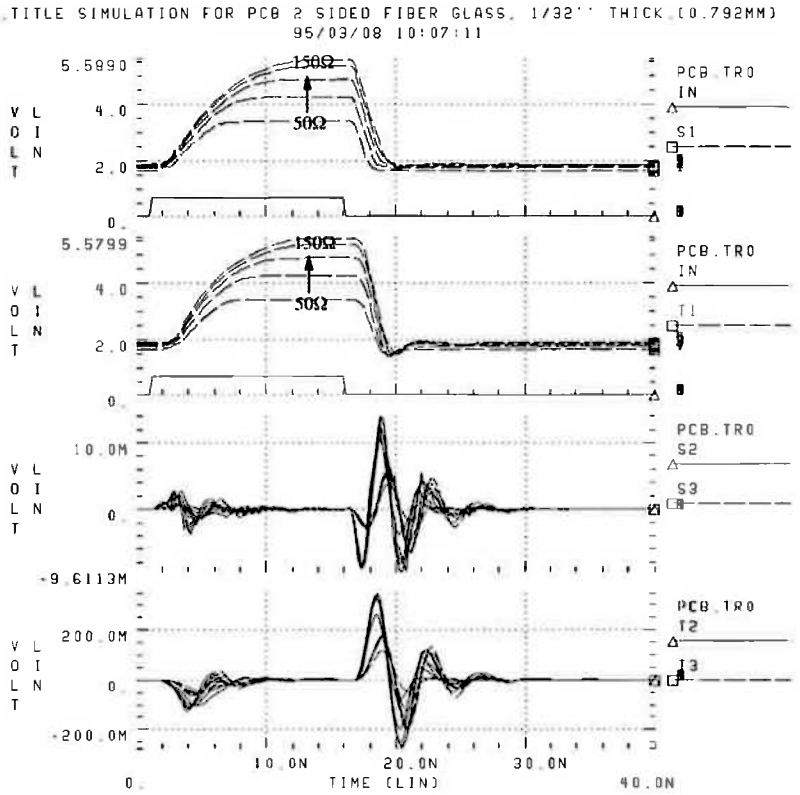


Figure 6.20 Simulation of a 4cm long line with R_l=50, 75, 100, 125, 150Ω and no R_s.

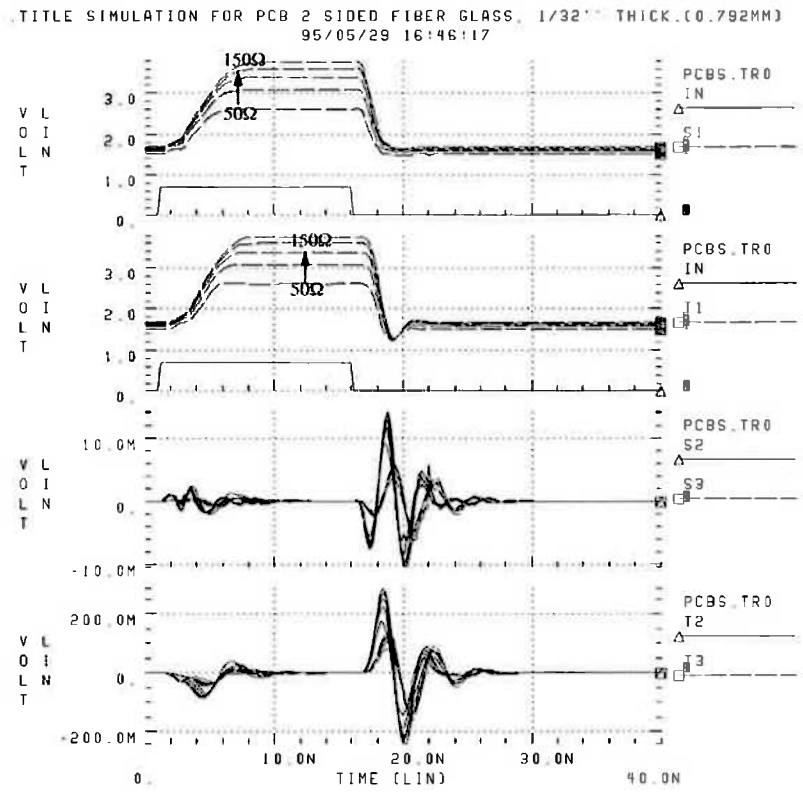


Figure 6.21 Simulation of a 4cm long line with R_l=100Ω, and R_s=50, 75, 100, 125, 150Ω.

6.3.5 Test of the chip

In order to test the CMOS compatible pad driver and receiver, and the speed of the 16×16-bit multiplier, a test structure as shown in Figure 6.22 is used. The delay of the I/O pads is the time from a change in high input of *set_in* signal to the change in output of *set_out* signal. The delay of the multiplier is the delay from change in input of *set_in* signal to the change in output of g_{29}, minus the delay of the I/O pads.

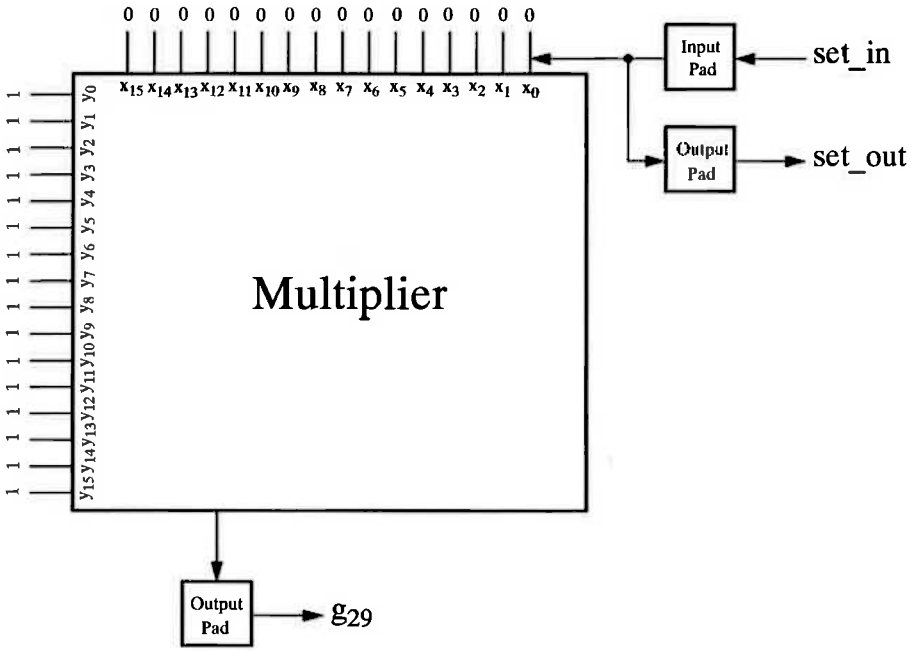


Figure 6.22 Speed test structure.

The micrograph of the chip is illustrated in Figure 6.23. A serious problem with the fabricated chip was found by examining the chip under the microscope. An output register array at the bottom of the multiplier had been displaced from its correct position by a significant distance. The register array now runs from the lower left side of the multiplier across some signal lines and over the top of a pad driver (see Figure 6.23).

A closer examination of the chip has revealed that some arrays of cells have been reflected in either the x or y axis. It would seem that this problem affects the arrays of cells that have themselves been reflected.

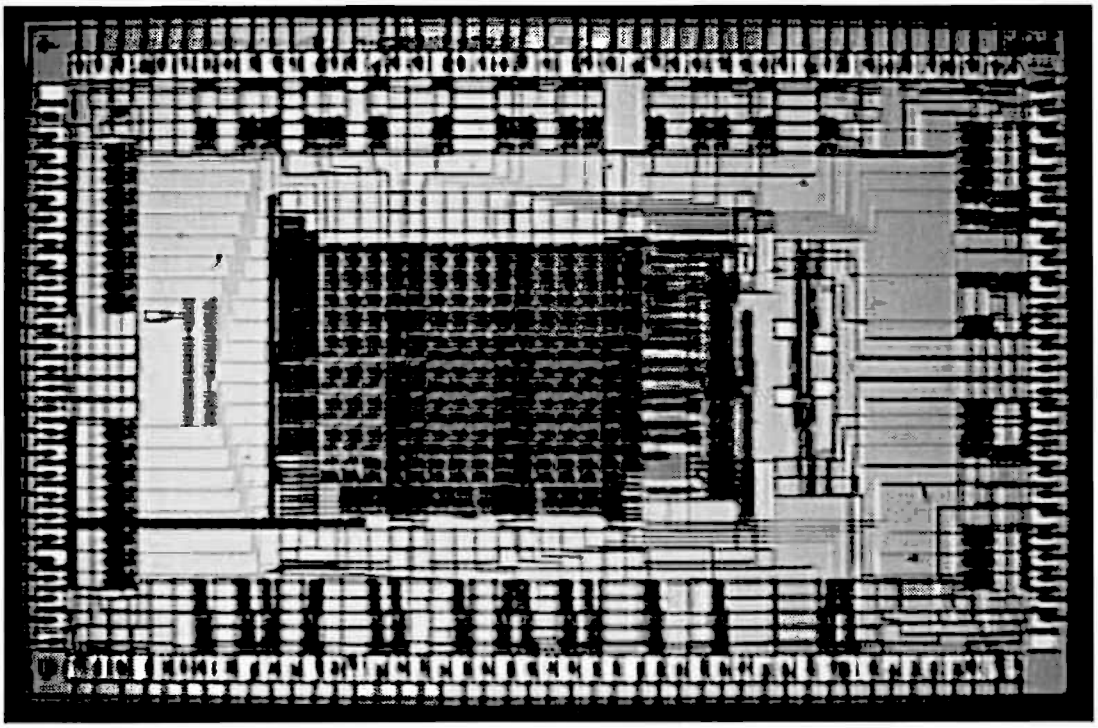


Figure 6.23 The micrograph of the multiplier chip.

The result of this fault made the fabricated chip completely useless. The cause of the fault has been traced to the software used by CMP to design the multi-project wafer. The foundry has offered a free refabrication of the chip. However this will take several more months, therefore the measured results will not be included in this thesis. They will be reported later when available.

6.4 Summary

A 16×16-bit fixed point multiplier chip was fabricated using VITESSE H-GaAs-II 0.8μm Technology. The details of the design, implementation and testing were described in this chapter. To enhance the speed, a modified carry save array and a multiple carry select adder are used in the multiplier architecture. The chip used the Modified Ring Notation approach to improved layout density. A net-like connection for global and local power supply and ground were used to reduce the *ir* drop, electromigration and inductance.

CHAPTER

7

Conclusions and Recommendations

7.1 Conclusions

The importance of a solid modelling system in CAD/CAM has been widely recognized. The trend toward real time solid modelling system, especially for complex object applications, demands the development a Solid Modelling Accelerator.

An examination of a solid modelling program call GWB was shown that half of execution time is spent on floating point intensive calculations (another half is spent on accessing to the data structure and hence on memory subsystem). An architecture of the Solid Modelling Accelerator was proposed using the GaAs/CMOS/BiCMOS unified technology. The essence of the unified technology concept is that the mix of the technologies can be adjusted to suit high performance architectures through appropriate partitioning. Because of the superior performance of digital GaAs circuits in terms of speed and power dissipation, GaAs technology is used in the core sections (VPU and Vector Cache) of the Solid Modelling Accelerator to implement floating point intensive calculations, while CMOS technology is used where high speed outputs are not required such as for frequent accesses to heavily interlinked high density data structures. BiCMOS is used as buffer to drive large loads in CMOS circuits.

The main objective of the research work described in this thesis was to design and implement the hardware mapping of critical paths of a GaAs Core Processor for the proposed Solid Modelling Accelerator. This was led to characterization of suitable logic families and development of suitable algorithms and a design approach to produce high speed, high density and low power dissipation GaAs VLSI IC's.

The evaluation and characterization of four GaAs logic families—Direct Couple Logic (DCFL), Source Follower DCFL (SDCFL), Super Buffer (SBFL) and Two-phase Dynamic Logic (TDFL) showed that DCFL gate was fast and occupies little area, used extensively throughout this research for local logic blocks. SDCFL showed that it had strong fan-out and capacitive driving ability, however it had higher power consumption. SDCFL was used in critical paths, while SBFL was used for buffering global signals (such as clocks) due to high power dissipation and noise injection into the power buses during high to low transition. TDFL was used in the mixed dynamic/static approach to reduce the power and area while maintain high speed.

A variety of division algorithms was examined, an 8-bit *nonrestoring* divider was implemented based on this mixed dynamic/static approach. TDFL was used as shift register and DCFL was implemented in static logic functions with the ability to drive moderate loads. The divider showed to operate at 1GHz clock frequency at *typical typical* processing parameters. The associated power dissipation from V_{DD} at this frequency was 0.35mW on average including the static adder/subtracter part. This work showed that the mixed dynamic/static approach was very promising for GaAs VLSI circuits because of low power dissipation and small area occupation. However, because the sensitivity of TDFL to variations in process spread and clock skew, it would better to be used locally.

A floating point multiplier is the most important elements in the GaAs Core Process. Therefore, a 32-bit IEEE floating point multiplier was designed and used as vehicle to select a suitable architecture for the GaAs Core Processor. After investigating various fixed and floating point multiplier algorithms, a modified carry save array, a multiple carry select adder and a Trailing-1's Predictor were used and developed to improve the speed of the floating point multiplication.

The Modified Ring Notation (MRN) approach was developed for the implementation of the GaAs Core Processor and many GaAs VLSI's. The analysis of this new layout approach included interconnections, power supply and ground considerations. The original Ring Notation has the advantages of easy design structuring and improved reliability. The MRN maintains the advantages of the original Ring Notation while reducing layout area and delay.

The floating point multiplier presented in the previous chapter was implemented by the new layout approach. The final layout provided an adequate measure of the efficiency of the proposed layout approach. The combination of the fast arithmetic architecture and compact layout style achieves *4ns* multiplication time with *3.5W* power dissipation at *75°C*. The area of the chip is *2.43mm by 3.77mm* (excluding pads) and uses *28,000* transistors to give a density of *3056 transistors/mm*^{2} for 0.8-μm GaAs technology. It has the best performance amongst the current designs in terms of delay, area and power dissipation.

A 16×16-bit multiplier chip was implemented to test the architecture and the layout methodology. It was fabricated using VITESSE H-GaAs-II 0.8μm Technology. It was found that there is a bug in the software used by CMP to design the multi-project wafer. The bug affects arrays of cells that have themselves been reflected The result of this bug made the chip completely useless. The chip will be refabriated by CMP. However, because of the long time needed for refabrication, the measured results of the chip is not included in this thesis.

7.2 Recommendations for Future Work

High speed and high precision computation are required for many digital signal processing, computer graphics, model simulation and image processing application, and so is the Solid Modelling Accelerator. For more accurate performance, solid modelling systems require double precision floating point operations. By using the modular architecture, all of the high-speed components needed to communicate with each other should be in the same modules. Then modules such as adder/subtracter, multiplier, divider and square root etc. can be packaged as a hybrid semiconductor to communicate with each other. However, with VITESSE H-GaAs-II 0.8μm Technology, it is difficult to implement the whole double precision floating point multiplier into one single chip even using MRN. Also, from Chapter 1 it can be seen that besides the Vector processing Unit the GaAs Core Processor also needs a Vector Cache. However, one of the major limitations of GaAs technology is the difficulty of producing large quantities of data storage. This arises from the relatively high power dissipation occurred through using storage elements on the one hand, and the fact that leakage currents prohibit the use of dynamic storage elements on the other. Furthermore, the interface between modules can use GaAs compatible I/O pads which have less than *1ns* delay [69]. However, the CMOS compatible I/O pads for interface with CMOS/BiCMOS part are quite slow (*5ns*).

Therefore, further development of the GaAs Core Processor based on the work of this thesis is likely to proceed on four fronts.

Firstly, use VITESSE H-GaAs-III 0.6μm technology or Fraunhofer-Institut für angenwandte Festkörperphysik high speed 0.3μm HEMT process, to scale down transistor's sizes, hence the chip's areas. Based on the work of Chapter 2, re-characterize

the four logic families using the new process parameters. Search for better logic families to enable higher operational speed while preserving low power consumption.

Secondly, finish all the arithmetic elements in the GaAs Core Processor using the scaled logic gates, and use the MRN for the layout.

Thirdly, design a dynamic RAM cell for the Vector Cache of the GaAs Core Processor using a three phase clock strategy to refresh charge at the storage node during read operations. The object is to use the cell to make a RAM array which consumes no static power, and is as fast as dynamic RAMs and consumes less area than static RAM.

Finally, modify the CMOS compatible pads to make them compatible with low voltage (3V) CMOS and have significantly faster speed than *5ns*.

Appendix A

Procedure to Implement the Face Equation in GWB

```
\footnotesize
\begin{verbatim}
# include "gwb.h"
int    faceeq(l, eq)
Loop   *l;
vector  eq;
{
      HalfEdge      *he;
      double        norm;
      vector        vi, vj, vc, v1, v2;
      int           i;
      vi[3]=vj[3]=vc[3]=v1[3]=v2[3]=eq[3]=1.0;

      he = l->ledg;
            veccopy(vi, he->vtx->vcoord);
            veccopy(vj, he->nxt->vtx->vcoord);
            vecminus(v1, vi, vj);
```

```
        he = he->nxt;
            veccopy(vj, he->vtx->vcoord);
            veccopy(vc, he->nxt->vtx->vcoord);
L:          vecminus(v2, vj, vc);

        cross(eq, v1, v2);

        norm = sqrt(dot(eq, eq));
        if(norm != 0.0)
        {  for( i=0; i<3; i++)
            eq[i] / =  norm;
            eq[3] = -dot(eq, vi);
        return(SUCCESS);
        }
        else
        {
           /*   printf("faceeq: null face %d\n", l->lface->faceno);
             return(ERROR);  */
           veccopy(vc, he->nxt->nxt->vtx->vcoord);
           goto L;
        }
}
\end{verbatim}
\normalsize
```

Appendix B

Design Tools Used in This Thesis

- VHDL (Very high speed integrated circuit Hardware Description Language) was used to describe modules at high level, then simulated using MENTOR GRAPHICS VHDL simulator before detailed subcircuits were designed.
- OCTTOOLS is package of various tools. It was mainly used for gate and transistor level design and functional simulation of sub-modules and random logic.
- MAGIC is a layout editor. It was used to implement layouts.
- HSPICE was used for detailed simulation from Subcells to the whole chip including processing variations.
- IRSIM is a fast and efficient event driven switch level simulator for the simulation of larger blocks. It provides a reasonable accurate estimation of the system's timing, but here it was mainly used to check functional behaviour.

Appendix C

HSPICE Simulation Files for Crosstalk

```
*** Generated from try.mag by Adelaide Uni modified magic spice generator
*** technology: cmp
*** extraction style: default
*** units per lambda: 0.10
*** timestamp: 774594049
*** version: 1.0
*** Capacitance Threshold = 100000 units
*** Resistance Scale     = 1000 units
.options post dcap=1 brief probe
* cmp uses Vitesse HSpice model parameters
.protect
.inc '/opt/vlsi/lib/tcm/hspice/vsc2.01models'
.lib '/opt/vlsi/lib/tcm/hspice/vsc2.01corners' tt
.unprotect
.global 1 vdd gnd
Vdd 1 0 2v
.global sub
Vsub sub gnd 0.6v
```

```
.probe tran v(out111) v(uline1) v(out112) v(uline2)  v(out11) v(tline1)v(out12)
v(tline2) v(out211) v(uline21) v(out212) v(uline22) v(out311) v(uline31) v(out312)
v(uline32)

.temp 75
.model line2 U level=3 plev=1 elev=1 nl=2
+KD=4.0 xw=0.4u rho=2.1e-8 rhob=3e-8
*+wd=1.5u ht=3.2u th=0.3u sp=2u
+wd=1.5u ht=2u th=0.5u sp=2u
+llev=1 dlev=0 maxl=25

.subckt sdcfl 101 100
J0 05 101 0 sub jfet04 L=1.2U W=21.0U $ x = -2287, y = -2674
J2 100 05 1 sub jfet04 L=1.2U W=25.0U $ x = -1958, y = -2674
J1 05 05 1 sub jfet19 L=1.6U W=6.0U $ x = -2045, y = -2710
J4 100 0 0 sub jfet16 L=1.2U W=12.0U $ x = -1678, y = -2675
C1 101 0 0.24F
C2 100 0 1.67F
C3 05 0 0.99F
C0 1 0 20.32F
.ends

.subckt sdcflb 101 100
J0 05 101 0 sub jfet04 L=1.2U W=42.0U $ x = -2287, y = -2674
J2 100 05 1 sub jfet04 L=1.2U W=50.0U $ x = -1958, y = -2674
J1 05 05 1 sub jfet19 L=1.6U W=12.0U $ x = -2045, y = -2710
J4 100 0 0 sub jfet16 L=1.2U W=24.0U $ x = -1678, y = -2675
C1 101 0 0.24F
C2 100 0 1.67F
C3 05 0 0.99F
```

```
C0 1 0 40.32F
.ends

.subckt inv 101 100
J0 100 101 0 sub jfet04 L=1.2U W=8.0U $ x = 8327, y = -2655
J1 100 100 1 sub jfet19 L=2.0U W=2.0U $ x = 8439, y = -2663
C1 101 0 0.24F
C2 100 0 1.67F
C0 1 0 5.43F
.ends

********** the followin calculations are for a line with L=5mm
****The characteristics of the line used is as follows:
****  L(unit length)=9.2nH/cm
****  C(unit length)=0.9pF/cm
****  V = 10 cm/ns
x111 in out111 sdcflb
x112 vhigh out112 sdcflb
x121 uline1 out1131 inv
x122 uline2 out1132 inv
x131 out1131 out1141 inv
x132 out1132 out1142 inv
u3 out111 out112 0 uline1 uline2 ref1 line2 L=0.005
rref1 ref1 0 1

vin in 0 pwl(0 0.65 0.5ns 0.65 0.6ns 0.15)
.tran (0.01ns 2ns)
.END
```

Appendix D

Testing Equipment

1. The Test Jig

A test jig was made from aluminium with a sealed cavity under the PCB as shown in Figure D.1 [117].

The PCB posts are gold plated so as not to corrode or react with the gold strips of the DAS pattern generation probes. The terminating resistors and capacitors are surface mount 805 type packages which are suitable for high frequency applications. Terminating resistors for chip output lines are mounted next to the PCB pins. All power pins have decoupling capacitors of 0.1μF connected to ground near the chip.

2. Chip Mounting

The following procedure was followed in mounting the chip:

- Ensure normal ESD prevention techniques for handling CMOS circuits are used. Carefully place the package on the PCB and align the pins with the PCB tracks. The elastomer ring and the retainer can be assembled and placed over the package and the four bolts inserted. With the bolts loose, adjust the retainer to correctly position the package and tighten the bolts.

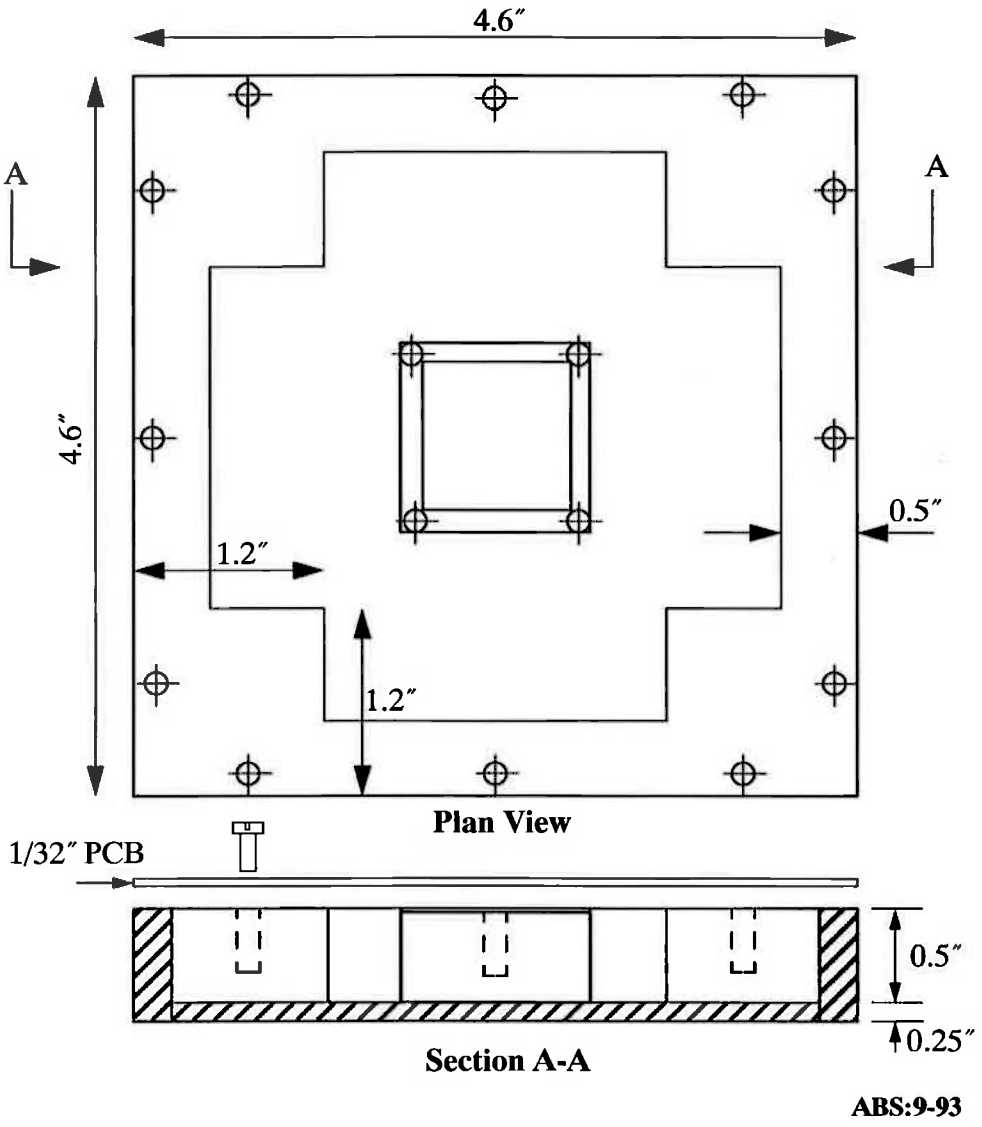


Figure D.1 High speed test jig for VITESSE 132 LDCC package.

- Check alignment of trace and corresponding pad by measuring the resistance between internally connected pins (e.g. pins 4, 16, 28, 42, 57, 70, 82, 94, 108 or 123). If aligned correctly, the resistance should be zero.
- Check the electrical integrity of the test fixture by measuring the resistance between ground and power supply at several pins.
- Place a finned heat sink on top of the package with a smear of Unick UH-102 Silicon heat transfer compound.

3. DAS 9200 High Speed Digital Tester

The Tektronix Digital Analysis System (DAS) 9200 is a tool which provides the operating environment for a *pattern generator module*, *data acquisition module*, and the software to control them. The software is configured to run via a host system using an X-window interface.

The software allows programming of the algorithmic pattern generation module with test vectors, after which the program can be run, applying signals to the chip under test. The resulting output signals can then be read from the data acquisition probes and displayed as a timing diagram from which a hard copy can be generated. Figure D.2 shows the setup of the DAS testing system.

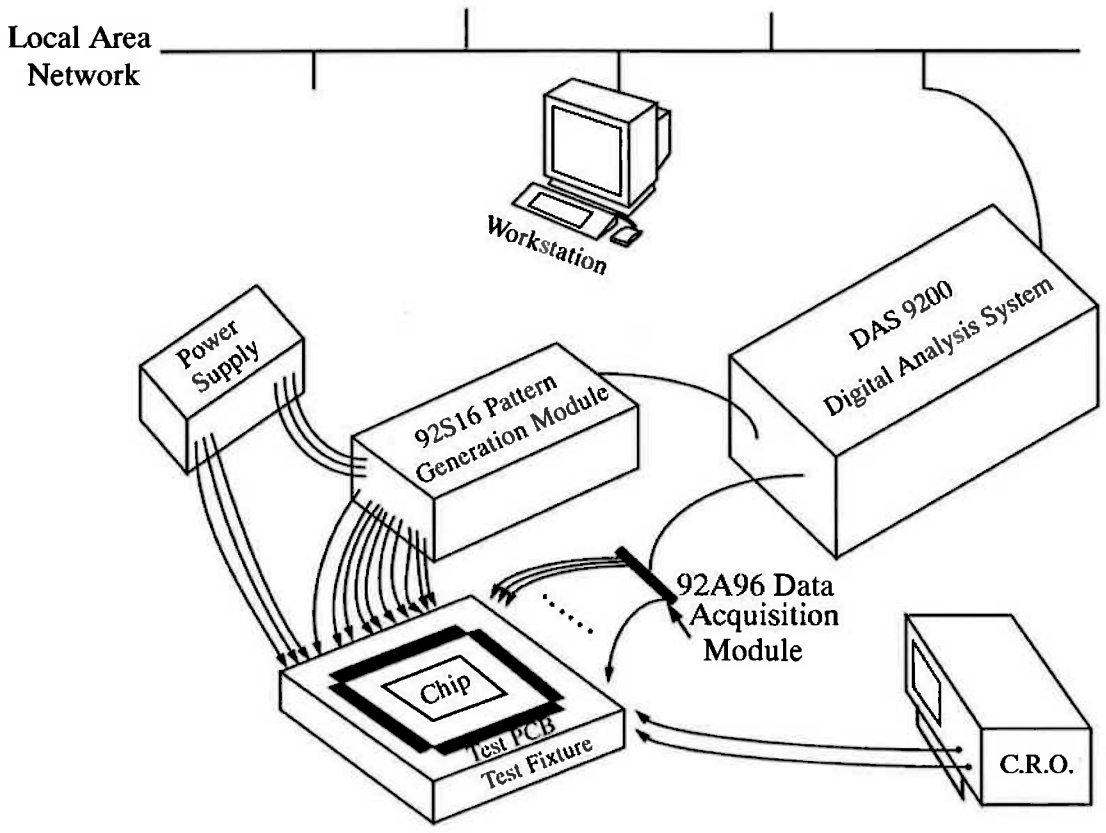


Figure D.2 Setup of the DAS9200 Digital Analysis System.

4. 92S16 Pattern Generation Module

The 92S16 Pattern Generator Module is connected to the device under test through the P6464 Pattern Generator Probe. The P6464 provides a total of 9 signal lines (bits 0-8) and a clock line (clk). The maximum clock frequency available is 50 MHz. The P6464 can supply signals at either TTL or ECL levels.

The P6464 receives power via three sense leads connected to the probe: a red line for V_H (voltage high), a black line for V_H (voltage low) and a green line for ground.

The specifications for the 92S16 Pattern Generator are shown in Table D.1

Table D.1 DAS 92S16 Pattern Generator Specifications

Characteristic	Specification
clock maximum frequency	50 MHz (20 ns)
V_H	-0.5V to +5.5V @ 100mA + I_{LOAD}
V_L	+0.3V to -5.5V @ 100mA + I_{LOAD}
V_H - V_L	4.8V to 5.2V
TTL V_L out	V_L + 0.8V
TTL V_H out	V_H - 1.1V
ECL V_L out	V_H - 1.75V
ECL V_H out	V_H - 1V
driver capability	20 mA (sink or source)

The 92S16 Pattern Generator Module can produce outputs at TTL levels with 5 volts swing or ECL levels with 0.8 volts swing. TTL output high and output low voltages can be controlled by the supply voltages applied to the P6464 probe module. By providing suitable supply voltages, the 0 to 5 volts TTL outputs can be level shifted up. Hence the two P6464 pods' power wires should be connected as follows:

- red: +6 volts
- black: +1 volts
- green: circuit chassis ground and power supply ground (These two should be connected together.)

This gives a logic high of 4.9 (V_H - 1.1V) volts and a logic low of 1.8 (V_L + 0.8V) volts at the probe tip. This meets the voltage swing requirement for the CMOS compat-

ible pads. ECL logic levels can not be used because the voltage swing is too small (0.8 V).

5. 92A96 Data Acquisition Module

The 92A96 Data Acquisition module provides a facility for sampling digital data in a format which can be viewed and analysed by DAS software. The user specifies a reference voltage, variable from -5 volts to +5 volts. Any voltage sampled above this voltage will register as a logic high and any voltage below this as a low.

The 92A96 in 24 channel high speed acquisition mode has a resolution of 2.5 ns or 400 MHz. The signal side of the probe tips is marked with a colour and should point towards the chip.

Bibliography

[6] Chiyokura, H."Solid Modelling with DESIGNBASE", *Addison-Wesley Publishers Company*, Inc. 1988

[7] Luo, Y. and Lukacs, G."A Boundary Representation of Form Features and Non-manifold Solid Objects." *1st ACM/SIGGRAPH Symposium on Solid Modelling Foundations and CAD/CAM Applications*, Austin, Texas, June 5-7, 1991.

[8] Rossignac, J."Beyond Solid Modelling". *Computer Aided Design* Vol. 23, no.1, Jan./Feb. 1991, pp. 2-3.

[9] Ellis, J. et al. "Breaking Barriers in Solid Modelling", *Mechanical Engineering*, Feb. 1991, pp. 28-34

[10]Requicha, A. A. G. and Voelcker, H. B."Solid Modelling: Current Status and Research Directions", *IEEE Computer Graphics and Applications*, Vol. 3, No. 7, Oct. 1983, pp. 25-37

[11]Requicha, A. A. G. and Rossignac, J. "Solid Modeling and Beyond", *IEEE Computer Graphics and Applications*, Vol. 12, No. 5, Sept. 1992, pp. 31-44

[12]Clark, J. "A VLSI Geometry Processor for Graphics", *Computer*, July, 1980, pp. 59-68

[13]Requicha, A. A. G."Representations for Rigid Solids: Theory, Methods, and Systems", *ACM Computing Survey*, Vol. 12, No., 4, Dec. 1980, pp. 437-464

[14]Requicha, A. A. G."Solid Modeling: A 1988 Update", *CAD Based Programming Sensory Robots*, B. Ravani, ed., Springer Verlag, New York, 1988, pp. 3-22

[15]Weiler, K and Electric, G."Edge-Based Data Structures for Solid Modelling in Curved-Surface Environments", *IEEE CG&A*, Vol. 5, 1985, pp. 21-40

[16]Woo, T. C.:"A Combinatorial Analysis of Boundary Data Structure Schemata", *IEEE CG&A*, 1985, vol.5, pp. 19-27

[17]Requicha, A. A. G. and Voelcker, H. B. "Boolean Operations in Solid Modelling: Boundary Evaluation and Merging Algorithms", *Proc. IEEE*, 73(1), Jan. 1985, pp. 30-44.

[18]Mäntylä, M.: "GWB: A Solid Modeler with Euler Operators", *IEEE CG&A*, 1982, vol.2, pp. 17-31

[19]Mäntylä, M.: "An Introduction to Solid Modeling", *Computer Science Press*, Inc. 1988

[20]Betts, N.:"GWB Data Structure Report", *Technical Report*, CSE-SMA-93-2, Dept. of Electrical and Electronic Engineering, The University of Adelaide

[21]Graham, S.L., Kessler, P.B., McKusick, M.K.: "gprof: A Call Graph Execution Profiler", *Proc.SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices*, Vol.17, No.6, pp. 120-126, June 1982.

[22]Moore, B., Betts, N., Cui, S. , Liebelt, M. and Eshraghian, K., "Memory Architecture for a Solid Modelling Accelerator in Unified GaAs/BiCMOS/CMOS Technology", *Proceedings of Australian Microelectronics conference*, Queensland, 1993, Oct. pp. 209-214.

[23]Elber, G.:"IRIT A Solid Modeling Program",1991

[24]Eshraghian, K. "Fundamentals of Very Hign Speed Systems: Gallium Arsenide VLSI Technology Course Notes. to be published.

[25]Long, Stephen I. & Butner, Steven E. "Gallium Arsenide Digital Integrated Circuit Design". *McGraw-Hill*, Inc. 1990

[26]Wing, Omar. "Gallium Arsenide Digital Circuits". *Kluwer Academic Publishers.* 1990

[27]VITTESS, ASICs, "HGaAsII Foundry Design Manual", Version 5.0

[28]Nary, K. R. and Long, S. I., "GaAs Two-Phase Dynamic FET Logic: A Low--Power Logic Family for VLSI", *IEEE J. Solid-State Circuits*, vol. 27, 1992, pp.1364-1371

[29]Larson, L. E. and Martin, R. W. et al, "GaAs Switched-Capacitor Circuits for High-Speed Signal Processing", *IEEE J. Solid-State Circuits*, vol. 22, 1987, pp.971-980

[30]Hill, C. F., "Noise Margin and Noise Immunity in Logic Circuits", *Microelectronics*, vol. 1, April, 1968, pp. 16-22

[31]Lohstroh, J., Seevinck, E. and Groot, J. De, "Worst-Case Static Noise Margin Criteria and Their Mathematical Equivalence. *IEEE Journal of Solid-State Circuits*, vol, SC-18, no.6, Dec. 1983, pp. 803-807

[32]Weste, Neil H. E. and Eshraghian, K., "Principles of CMOS VLSI Design, A system Perspective", *Addison Wesley*, second edition, 1992

[33]Meta-Software, Inc., HSPICE User's Manual, 1990

[34]Kocot, C. and Stolte, C. A., "Backgating in GaAs MESFETs", *IEEE Trans. Elect. Dev.*, vol. ED-29, July, 1982, pp. 1059-1064

[35]Curtice, W. R., "A MESFET Model for Use in the Design of GaAs Integrated Circuits", *IEEE Transactions on Microwave Theory and Technology*, MTT-28, May 1980, pp. 448-456

[36]Ghione, G. and Naldi, C.U. "Coplanar Waveguides fro MMIC Applications: Effect of Upper Shielding, Conductor Backing, Finite-Extent Ground Planes, and Line-to-Line Coupling", *IEEE Trans. Microwave Theory and Techniques*, vol MTT-35, March 1987, pp. 260-267

[37]Ghione, G. and Naldi, C.U. "Analytical Formulas for Colanar Lines in Hybrid and Monolithic MICs", *Elect. Lett.*, vol.20, Feb. 1984, pp. 179-181

[38]Hilberg, W. "From Approximation to Exact Relations for Characteristic Impedances", *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-17, May 1969, pp259-265

[39]Schneider, M. V. "Microstrip Lines for Microwave Integrated Circuits", *Bell Syst. Tech. J.*, vol. 48, May/June 1969, pp. 1421-1444

[40]Gupta, K. C., Garg, R. and Chadha, R. "Computer-Aided Design of Microwave Circuits, Chap. 3, *Artech House*, 1981

[41]Earl E. Swartzlander, Jr. "Computer Arithmetic", 1990 The Institute of Electrical and Electronics Engineers, Inc.

[42]Thomas, G. B., "Calculus and Analytic Geometry", *Reading, MA. Addison-Wesley*, 1962, pp. 451-452

[43]Waser S. and Flynn M. "Introduction to Arithmetic for Digital Systems Designers", *Holt, Rinehart and Winston*, CBS College Pulblising, 1982

[44]Freiman C. V. "Statistical Analysis of Certain Binary Division Algorithms", *Proc. IRE*, Vol. 49, Jan. 1961, pp. 91-103

[45]Robertson J. E. "A New Class of Digital Division Methods", *IRE Trans. Electronic Computers*, Vol. EC-7, Sept. 1958, pp. 218-222

[46]Tocher T. D. "Techniques of Multiplication and Division for Automatic Binary Computers", Quarter. J. Mech. App. Math., Vol. 2, pt. 3, 1958, pp. 364-384

[47]Robertson J. E. "Methods of Selection of Quotient Digits During Digital Division", Dept. of Computer Scientce, University of Illinois, Urbana, Ill., File 663, 1965

[48]Schwarz E. M. and Flynn M. J. "Parallel High-Radix Nonrestoring Division", *IEEE Transactions on computer*, Vol. 42, No. 10, 1993, pp. 1234-1246

[49]Anderson, F. S., et al., "The IBM System 360/91 Floating Point Execution Unit", *IBM J. Res. Devel.*, Vol. 11, No. 1, January 1967, pp. 34-53

[50]McQuillan S. E. and McCanny J. V. "VLSI Module for High-Performance Multiply, Square Root and Divide", *IEE Proceedings-e*, Vol. 139, No. 6, 1992, pp. 505-510

[51]Lu M. and Chiang J. S. "A Novel Division Algorithm for the Residue Number System", *IEEE Trans. on computers*, Vol. 41, No. 8, 1992, pp. 1026-1032

[52]Wong D. and Flynn M. "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations", *IEEE Trans. on computers*, Vol. 41, No. 8, pp. 981-995

[53]Jain V. K., Perez G. E. and Wills J. M. "Novel Reciprocal and Square-Root VLSI Cell: Architecture and Application to Signal Processing", *1991 IEEE*, pp. 1201-1204

[54]Cui S., Eshraghian K. and Liebelt M. J. "A Serial GaAs Dynamic/Static Divider for a Solid Modelling Accelerator", *Proceedings of Australian Microelectronics conference, Queensland,* 1993, Oct. pp. 249-254

[55]K. Eshraghian, R. Sarmiento et al, "Speed-Area-Power Optimization for DCFL and SDCFL Class of Logic Using Ring Notion", *Microprocessing and Microprogramming 32,* 1991, NORTH-HOLLAND pp. 75-82

[56]Sarmiento R., Carballo P. P. and Nunez. A. "High Speed Primitives of Hardware Accelaerators for DSP in GaAs Technology", *IEE Proceedings-G,* Vol. 139, No. 2, 1992, pp. 205-216

[57]Bushehri E. "Critical Design Issues for Gallium Arsenide VLSI Circuits", Ph.D thesis.

[58]Srinivas H. R. and Parhi K. K. "A Fast VLSI Adder Architecture", *IEEE Journal of Solid-State Circuits,* Vol. 27, No. 5, 1992, pp. 761-767

[59]Brent R. P. and Kung H. T. "A Regular Layout for Parallel Adders", *IEEE Transactions on Computers,* Vol. C-31, No. 3, 1982, pp. 260-264

[60]Wei B. W. Y. and Thompson C. D. "Area-Time Optimal Adder Design", *IEEE Trans. on computers,* Vol. 39, No. 5, 1990, pp. 666-675

[61]Bedrij O. J. "Carry-Select Adder", *IRE Trans. Electrons Comput.,* Vol. EC-11, 1962, pp. 340-346

[62]Tyagi A. "A Reduced-Area Scheme for Carry-Select Adders", *IEEE Trans. on computers,* Vol. 42, No. 10, 1993, pp. 1163-1170

[63]Guyot A., Hochet B. and Muller J. M. "A Way to Build Efficient Carry-Skip Adders", *IEEE Transactions on Computers,* Vol. C-36, No. 10, 1987, pp. 1144-1152

[64]Kantabutra V. "Designing Optimum One-Level Carry-Skip Adders", *IEEE Trans. on computers,* Vol. 42, No. 6, 1993, pp. 759-764

[65]Kantabutra V. "Accelerated Two-Level Carry-Skip Adders—A Type of Very Fast Adders", *IEEE Trans. on computers,* Vol. 42, No. 11, 1993, pp. 1389-1393

[66]"New Multiplevel Scheme for Fast Carry-Skip Addition", *IBM Technical Disclosure Bulletin,* Vol. 27, No. 11, April 1985, pp. 6785-6787

[67]Kantabutra V. "A Recursive Carry-Lookahead/Carry-Select Hybrid Adder", *IEEE Trans. on computers,* Vol. 42, No. 6, 1993, pp. 1495-1499

[68]K. R. Nary and S. I. Long:"GaAs Two-Phase Dynamic FET Logic:A Low-Power Logic Family for VLSI", *IEEE J. Solid--State Circuits,* vol. 27, 1992, pp1364-1371

[69]J. Jakobsen:"Design of SP/PS Test Chips", *Jydsk Telefon Internal Technical Report,* Mar. 1993

[70]Sterbenz P. H. "Floaing Point Computation", *Englewood Cliffs, NJ. Prentice-Hall,* 1974

[71]Garner H. L. "A Survey of Some Recent Contributions to Computer Arithmetic", *IEEE Trans. Computers*, Vol. C-25, No. 12, 1976, pp. 1277-1282

[72]IEEE/ANSI Standard 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic", *IEEE*, 1985

[73]Uya M., Kaneko K. and Yasui J. "A CMOS Floating Point Multiplier", *IEEE J. Solid-State Circuits*, Vol. SC-19, No. 5, 1984

[74]Keikes C. A., Colon-Bonet G., Miller B. and Barlett D. E. "The Design of 200 MFLOP Floating-Point Megacells for PA-RISC 7100", *Hewlett-Packard, WAM-1.5-1*, pp. 25-32

[75]Asprey T. et al. "Performance Features of the PA7100 Microprocessor", *IEEE Micro*, June 1993, pp. 22-35

[76]Huntsman C. and Cawthron D. "The MC68881 Floating-point Coprocessor", *IEEE Micro*, Dec. 1983, pp. 44-54

[77]Wolrich G. et al. "A High Performance Floating Point Coprocessor", *IEEE J. of Solid-State Circuits*, Vol. SC-19, No. 5, 1984, pp. 252-258

[78]Fandrianto J. and Woo B. Y. "VLSI Floatiing-Point Porcessors", *Proceedings of Seventh Symposium on Computer Arithmetic*, 1985, pp. 93-100

[79]Bose B. K., Taylor G. S. and Patterson D. A. "Fast Multiply and Divide for a VLSI Floating-Point Unit", *Proceedings of Eighth Symposium on Computer Arithmetic*, 1987, pp. 87-94

[80]Takeda K. et al. "A Single-Chip 80-bit Floating Point Processor", *IEEE J. Solid-State Circuits*, Vol. SC-20, No. 5, 1985, pp. 275-281

[81]Ware F. A. et al. "64 Bit Monolithic Floating Point Processors", *IEEE J. Solid-State Circuits*, Vol. SC-17, No. 5, 1982, pp. 898-907.

[82]Ware F. et al. "Fast 64-bit chip set gangs up for double-precision floating-point work", *Electronics*, July 12, 1984, pp. 99-103

[83]Sarmiento R. Carballo P. P. and Núñez A. "High Speed Primitives of Hardware Accelerators for DSP in GaAs Technology", *IEE Proceedings-G*, Vol. 139, No. 2, 1992, pp. 205-210

[84]Booth A. D. "A Signed Binary Multiplication Technique", *Quart. J. Mech. App. Math.*, Vol. 4, Part 2, 1951, pp. 236-240

[85]Delhaye E. et al. "A 2.5ns, 40mW, 4 × 4 GaAs Multiplier in Two's Complement Mode", *IEEE J. of Solid-State Circuits*, Vol. SC-22, No. 3, 1987, pp. 409-414

[86]Delhaye E. et al. "A 3.0ns, 40mW, 8 × 8 GaAs Booth's Multiplier", *IEEE GaAs IC Symposium*, 1987, pp. 249-252

[87]Mac Sorley O. L. "High Speed Arithmetic in Binary Computers", *Proc. IRE*, Vol. 49, 1961, pp. 67-91

[88]Rubinfield L. P. "A Proof of the Modified Booths Algorithm for Multiplication", *IEEE Trans. on Computers*, 1975, pp. 1014-1015

[89]Wallace C. S. "A Suggestion for a Fast Multiplier", *IEEE Trans. Electron. Comput.* EC-13, 1964, pp. 14-17

[90]Song Paul. J. and Micheli G. De "Circuit and Architecture Trade-offs for High-Speed Multiplication", *IEEE J. Solid-State Circuits*, Vol. 26, No. 9, 1991, pp. 1184-1198

[91]Oklobdija V. G. and Villeger D. "Multiplier Design Utilizing Improved Column Compression Tree and Optimized Final Adder in CMOS Technology", 1993 *VLSITSA*, pp209-212

[92]Harata Y., Nakamura Y., Nagase H., Takigawa M. and Takagi N. "A High-Speed Multiplier Using a Redundant Binary Adder Tree", *IEEE J. of Solid-State Circuits*, Vol. SC-22, No. 1, 1987, pp. 237-189

[93]Avizienis A. "Signed-Digit Number Representations for Fast Parallel Arithmetic", *IRE Transactions on Electronic Computers*, Vol. EC-10, 1961, pp. 389-400

[94]Santoro M. R. , Bewick G. and Horowitz M. A. "Rounding Algorithms for IEEE Multipliers", *Proceedings of 9th Symposium on Computer Arithmetic*, Sept. 1989, pp. 176-183

[95]Burgess N. Lecture Note

[96]Cui S, Burgess N., Liebelt M. J. and Eshraghian K. "A 32-bit GaAs IEEE Floating Point Multiplier Using Trailing-1's Rounding Algorithm", *Proceedings of the IEEE Electronics Technology Directions To The Year 2000 International Conference*, Adelaide, May, 1995, pp. 246-252

[97]Oowaki Y. et al. "A Sub-10-ns 16×16 Multiplier Using 0.6-μm CMOS Technology", *IEEE J. of Solid-State Circuits*, Vol. SC-22, No. 5, 1987, pp. 762-767

[98]Tate L. R. et al. "32-Bit GaAs IEEE Floating Point Multiplier", *GaAs IC Symposium*, 1992, pp. 85-88

[99]Mori J. et al. "A 10-ns 54×54-b Parallel Structured Full Array Multiplier with 0.5-μm CMOS Technology", *IEEE J. of Solid-State Circuits*, Vol. 26, No. 4, 1991, pp. 600-606

[100]Singh H. P., Burrier R. A. and Sadler R. A. "A 6.5-ns GaAs 20×20-b Parallel Multiplier with 67-ps Gate Delay", *IEEE J. of Solid-State Circuits*, Vol. 25, No. 5, 1990, pp. 1226-1231

[101]Singh H. P., Sadler R. A., Irvine J. A. and Gorder G. E. "GaAs Low-Power Integated Circuits for a High-Speed Digital Signal Processor", *IEEE J. of Solid-State Circuits*, Vol. 36, No. 2, 1989, pp. 240-249

[102]Sekiguchi T., Hirose T., Nishiguchi M., Shiga N. and Hayashi H. "A Multi-chip Packed GaAs 16×16 Bit Parallel Multiplier", *GaAs IC Symposium*, 1990, pp.199-202

[103]Sharma R. et al. "A 6.75-ns 16x16-bit Multiplier in Single-Level-Metal CMOS Technology", *IEEE J. of Solid-State Circuits*, Vol. 24, No. 4, 1989, pp. 922-927

[104]Yano K. et al. "A 3.8-ns CMOS 16x16-b Multiplier Using Complementary Pass-Transistor Logic", *IEEE J. of Solid-State Circuits*, Vol. 25, No. 2, 1990, pp. 388-395

[105]Cirillo N. C. et al. "8x8-Bit Piplined Parallel Multiplier Utilizing Self-Aligned Gate n^+-(Al, Ga)As/GaAs MODFET IC Technology",*GaAs IC Symposium*, 1987, pp. 257-260

[106]Henlin D. A., Fertsch M. T., Mazin M. and Lewis E. "A 16 Bit x 16 Bit Piplelined Multiplier Macrocell",*IEEE J. of Solid-State Circuits*, Vol. SC-20, No. 2, 1985, pp. 542-547

[107]Akinwande A. I. et al."A 500-MHz 16x16 Complex Multiplier Using Self-Aligned Gate GaAs Heterostructure FET Technology", *IEEE J. of Solid-State Circuits*, Vol. 24, No. 5, 1989, pp. 1295-1300

[108]Lu Fang and Samueli Henry, "A 200-MHz CMOS Pipelined Multiplier-Accumulator Using a Quasi-Domino Dynamic Full-Adder Cell Design", *IEEE J. of Solid-State Circuits*, Vol. 28, No. 2, 1993, pp. 123-132

[109]Kanopoulos Nick and Carabetta Joseph H. "Design and Implementation of a Totally Self-checking 16x16 bit Array Multiplier", *Integration, the VLSI Journal*, Vol.14, 1992, pp. 215-228

[110]Lim Y. C. "Single-Precision Multiplier with Reduced Circuit Complexity for Signal Processing Applications", *IEEE Transactions on Computers*, Vol. 41, No. 10, 1992, pp. 1333-1336

[111]Moreno J. M., Castillo F. and Cabestany J. "Shared Row Multiplier", *Electronics Letters*, Vol. 28, No. 15, 1992, pp. 1463-1465

[112]Cosentino R. J. and Vaccaro J. "Adaptation of the Mactaggart and Jack Complex Multiplication Algorithm for Floating-Point Operations",*IEEE Transactions on Computers*, Vol. 41, No. 10, 1992, pp. 1324-1326

[113]Kessler Edward A. "Digital Circuit Multiplication Equipment and Systems—An Overview", *British Telecommunications Engineering*, Vol. 11, 1992, pp. 106-111

[114]Scherson I. D., Kramer D. A. and Alleyne B. D. "Bit-Parallel Arithmetic in a Massively-Parallel Associative Processor", *IEEE Transactions on Computers*, Vol. 41, No. 10, 1992, pp. 1201-1210

[115]LaRue George S. and Grider David E. "Complementary HFET 32-Bit Serial Multiplier", *GaAs IC Symposium*, 1992, pp. 89-92

[116]VITTESS, ASICs, "TCB50K Standard Cell Design Manual", Version 1.0

[117]Beaumon-smith A, McGeever M. K. et al., "Testing the Broadand ISDN Test Chip", *GaAs Internal Report*, GAAS 94-1

[118]Wong A. "Testing the Gallium Arsenide Multiplier Test Chip", Project A Report, May, 1995, Dept. of EEE, Univ. of Adelaide