VLSI IMPLEMENTATION OF A ROUTER FOR THE BACKTRACK-TO-THE-ORIGIN-AND-RETRY-ROUTING SCHEME OF THE HYPERCYCLE BASED INTERCONNECTION NETWORKS

by

SIVAKUMAR RADHAKRISHNAN
B.Eng., University of Madras, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in the Department of
Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

[redacted]

Dr. N. J. Dimopoulos, Supervisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. K. F. Li, Co-Supervisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. D.M. Miller, Outside Member, Dept. of Computer Science

[redacted]

Dr. R. Vahldieck, Graduate Advisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. M. Serra, External Examiner, Dept. of Computer Science

Supervisors: Dr. N.J. Dimopoulos and Dr. K.F. Li

ABSTRACT

Hypercycles are a new class of multidimensional graphs that are ideal vehicles for designing interconnection networks for distributed systems. Hypercycle graphs are generalizations of n-cube and they offer simple, elegant and efficient routing among the different nodes of the network. Hypercycles embody a cyclic interconnection topology and have the ability, given a fixed degree, to form a number of alternate size graphs. This property can be tailored to meet the required application resulting in an efficient utilization of the network.

The regular structure of a Hypercycle allows the implementation of simple routing algorithms which require no detailed knowledge of the network interconnection. The Backtrack-to-the-origin-and-retry-routing scheme (BTOR) considered here is one of the several types of routing strategies that can be implemented in Hypercycle networks. The basic tenet of the BTOR scheme is to perform the routing by choosing at random, one of the available paths from each intermediate node that are at shortest distances from the source to the destination. If a path is blocked at some stage in the network, the routing is collapsed back to the origin for retry. The BTOR scheme is controlled by a routing engine present in each node of the network.

In this thesis, the theory, design and hardware realization of the proposed router are presented. The designs which have evolved as part of this research have been fabricated in the 1.2 μ and 3 μ CMOS technologies and the prototypes have been tested and verified.

Examiners:

[redacted]

Dr. N.J. Dimopoulos, Supervisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. K.F. Li, Co-Supervisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. D.M. Miller, External Member, Dept. of Computer Science

[redacted]

Dr. R. Vahldieck, Graduate Advisor, Dept. of Elect. & Comp. Eng.

[redacted]

Dr. M. Serra, External Member, Dept. of Computer Science

Table of Contents

List of Tables

List of Figures

Acknowledgment

I wish to record my gratitude to my supervisor, Dr. N. J. Dimopoulos of the Department of Electrical and Computer Engineering, for his perpetual encouragement, guidance, and advice during the course of this research and for his patience and help in the preparation of this manuscript. His undivided motivation, endeavour, and ability to create a congenial and informal atmosphere for discussion have been the major driving factors in the success of this project.

I thank my co-supervisor Dr. K.F. Li for his valuable comments and for aiding me with text books and technical papers during the progress of this research. I am grateful to Dr. M. Serra and the VLSI group of the Dept. of Computer Science for their help in the testability aspects of VLSI.

The assistance rendered by the Canadian Microelectronics Corporation in the fabrication of VLSI chips for this project is fully acknowledged. I am grateful to Dr. D. M. Miller of the Dept. of Computer Science in this regard.

I sincerely express my gratitude to Mr Alfred Keddy, Mr. Roger Kelly and Mr. Kevin Jones and the system administrators for their untiring efforts in helping me in many critical periods of this project, particularly in setting up the Cadence Design environment and IC testing facilities in the VLSI/microelectronics laboratories.

I appreciate the secretaries of the Dept. of Electrical and Computer Engineering for their timely help. Financial assistance received from Dr. N. J. Dimopoulos and Dr. K.F. Li (through NSERC) is gratefully acknowledged.

And finally, a word of gratitude to my parents, relatives, friends and well-wishers for their moral support, motivation, and encouragement without which my studies at UVic would not have crystallized.

The research work concerning the Hypercycle networks presented here is fully dedicated to *His Holiness Sri Chandarasekharendra Saraswathi Sankaracharya Swamigal of the Kanchi Kamakoti Peetam, at Kancheepuram, Tamilnadu, India* and to *His Holiness Sri Mukkur Lakshmi Nrisimhacharyar, Madras, India.*

Notation

List of Principal Symbols

δ	Average distance between two nodes in the Hypercycle graph
ρ_i	connectivity of the Hypercycle graph
η	Wire density of the Hypercycle network
μ_n	Mobility of electrons $(cm^2/volt - sec)$
μ_p	Mobility of holes $(cm^2/\text{volt-sec})$
ρ_i	Connectivity of the Hypercycle graph in dimension i
$\mathcal{G}_m^\rho = \{\mathcal{N}_m^\rho, \mathcal{E}_m^\rho\}$	r-dimensional Hypercycle Graph
$\mathcal{N}_m^\rho$	Set of nodes in the Hypercycle graph
$\mathcal{E}_m^\rho$	Set of edges in the Hypercycle graph
E^n	Euclidean space of dimension n
d	Degree of the Hypercycle graph
k	Diameter of the Hypercycle graph

List of Abbreviations

ACS	Adder-Comparator-Subtractor block
ALFSR	Autonomous Linear Feedback Shift Register
ALU	Arithmetic & Logic Unit
ASIC	Application Specific Integrated Circuit
AT	Area-Time
ATE	Automatic Test Equipment
ATG	Automatic Test Generation
BTOR	Back-Track-to-the-Origin-and-Retry-Routing
BIST	Built-In-Self-Test
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CATPAG	Computerized Automatic Test Pattern Generator
CLA	Carry Look-Ahead adder
CMC	Canadian Micro-electronics Corporation
CMOS	Complementary Metal Oxide Semiconductor
CMOS3DLM	Northern Telecom Electronics 3-micron single-polysilicon, double level metal, P-well CMOS process
CMOS4S	Northern Telecom Electronics 1.2-micron double-polysilicon, double level metal, twin-well CMOS process
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
CSA	Carry Select Adder
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DFT	Design For Testability
DIP	Dual-in-line Package
DSM	Design Scale Micron
DRC	Design Rule Check
DUT	Device Under Test
ECL	Emitter Coupled Logic
ERC	Electric Rule Checker
GF	Galois Field
GRP	Global Router Optimization
HCR	Hypercycle Router
HRE	Hypercycle Router Engine

LFSR	Linear Feedback Shift Register
LSSD	Level Sensitive Scan Design
MCRNG	Mixed Congruential Random Number Generator
MISD	Multiple Instruction Single Data
MISD	Multiple Instruction Multiple Data
MOD	Modulo-Extractor
MOS	Metal-Oxide Semiconductor
MRN	Mixed Radix Number
nMOS	n-channel Metal Oxide Semiconductor
NPG	Next Port Generator
OS	Operating System
pMOS	p-channel Metal Oxide Semiconductor
PE	Processing Element
PGA	Pin Grid Array
PLA	Programmable Logic Array
PLCC	Plastic Leadless Chip Carrier
PS	Port Selector
PV	Port Validator
PODEM	Path Oriented DEcision Making
PRNG	Pseudo Random Number Generator
RA	Residue Adder
RAS	Random Access Scan
RMF	Residue Mapping Function
RNG	Random Number Generator
RNS	Residue Number System
ROM	Read-Only-Memory
SCOAP	Sandia Controllability and Observability Analysis Program
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
TAT	Turn-Around-Time
TTL	Transistor-Transistor Logic
VLSI	Very Large Scale Integration

Glossary

Array Processor An *Array Processor* is a synchronous parallel computer with multiple arithmetic logic units called processing elements that operate in parallel.

ATG *Automatic Test Generator* is a computer-aided-engineering tool that produces test sequences for detecting faults in a circuit.

Automatic Placement and Routing A circuit design tool that automatically positions standard/ I/O cells and computes wire paths (signal interconnections) during the physical layout of a circuit.

CIF The *Caltech Intermediate Form* is a low-level symbolic graphics language for specifying the geometry of integrated circuits. The CIF code is used to describe the chip geometry in a machine readable form for mask making.

Connectivity The *connectivity*[1] ρ_i of a Hypercycle graph in dimension i is defined as the fraction of its degree in dimension i given by $\rho_i = \lceil \frac{d_i}{2} \rceil$.

Controllability *Controllability* of a digital circuit is a numerical measure which represents the ease at which a node in the circuit may be controlled from the primary inputs. Primary inputs are assumed to have a controllability factor of 1.

Degree The *degree* of a node in a graph is defined as the number of edges incident on the node. A graph is said to be *regular* if all nodes have the same degree. The degree of a graph is the maximum of the node degrees.

[1]The function $\lceil x \rceil$ denotes the smallest integer greater than or equal to x

For an r-dimensional Hypercycle, the degree is defined as the sum of the degrees taken over r dimensions.

$$d = \sum_{i=1}^{r} d_i$$

where d_i is the degree in dimension i.

Design Validation A sequence of input stimuli applied to a proposed design for the primary purpose of determining whether the circuit functions as intended.

Diameter The *diameter* of a graph is defined as the longest distance separating two nodes of the graph.

DFT *Design for Testability* is a design process wherein deliberate effort is expended to ensure that the circuit can be thoroughly tested.

DRC The *Design Rule Checker* is a program that checks for geometrical constraints of a device to ensure that they are within the resolution of the fabrication process and do not cause violation in the device physics required for proper operation of the transistors. The constraints include minimum allowable values for widths, separations, extensions, and overlaps of geometrical layers in the device.

DSM *Design Scale Micron* refers to the basic unit of defining the dimensions of devices during layout generation. In the CMOS3DLM technology, a design scale microns is scaled to 60 % of its size before pattern generation. In CMOS4S, it is scaled to 80 %. These parameters pertain to the Northern Telecom Electronics CMOS foundry.

ERC *Electric Rule Checker* is a program that checks for electrical connectivity in a design including circuit connectivity, pull-up, pull-down and fanout conditions.

Fault Coverage *Fault Coverage* is the ratio of the number of detectable faults for a given set of test patterns to the total number of faults for a particular fault model.

Gate Array A geometric pattern of basic gates arranged in rows or columns. The desired circuit is realized by interconnecting these predefined elements during manufacturing.

Noise Margin The *noise margin* is a parameter that indicates the ability of a circuit to assure correct operation regardless of variations in process parameters and external disturbances such as voltage spikes. There are two noise margins, namely noise margin (high) and noise margin (low).

Observability *Observability* of a digital circuit is defined as the ease at which a node in the circuit may be observed from the primary outputs. Primary outputs have an observability factor of 1.

Semi-custom IC An integrated circuit designed using precharacterized functional blocks or standard cells. Standard cells are placed in rows or columns so that they share power rails and have open routing channels.

Sequential circuit A *sequential* circuit is a digital circuit whose output depends on the past and present inputs.

Testability A circuit is said to be *testable* if a set of test patterns can be generated, evaluated and applied in such a way as to satisfy pre-defined

levels of performance in terms of fault-detection, fault-location and test application within a pre-defined cost budget and time scale.

Test program A *test program* is essentially a computer program that generates test vectors for testing chips on an automatic test equipment.

Timing verification A tool or procedure that determines if a circuit design functions at the required speed. Timing parameters such as set-up and & hold times, clock skews, propagation delays are measured.

Walk A *walk* in a Hypercycle graph is defined as a finite sequence of vertices starting from a source node and ending with a destination node of the graph. No intermediate edge or vertex is traversed more than once during each walk.

Unix is a trade mark of AT & T laboratories. **Cadence** and **Edge** are trademarks of Cadence Design Inc. **SILOS** and **SILOS II** are trademark of Simucad Inc. **Logic Master** is a trademark of Integrated Measurement Systems Inc. **ASIX** is a trademark of Credence Systems Corporation. **Extend** is a trademark of Imagine That Inc. **HSPICE** is a trademark of Metasoftware Inc.

Chapter 1

Introduction

The development of high speed digital computers since World War II has been one of the most significant contributions to technology and science during this century. Digital computers spanning right from the *ENIAC* of the 1940's to the present day *Cray X-MP* supercomputers have witnessed a major revolution in terms of processing speed and cost. Yet, the demand for higher speed, higher performance and more reliable computers at low costs continues to confront humanity and is a topic of active research for scientists and engineers.

The improvement in computing power from the first to the third generation computer systems was solely confined within the traditional von Neumann Architecture or SISD machines with hardware costs being a dominant and limiting factor. Despite the architectural advances within the CPU and memory of the SISD machines, the inherent bottleneck of sequential processing was a major limitation in many real time applications such as in image processing, signal processing, fluid mechanics, pattern recognition, weather forecasting, simulation and modeling and so on [1]. To circumvent the Von Neumann bottleneck, innovations have been and are brought about in the form of *parallelism* and this has given way to the new field of parallel computing. The important approaches

to parallelism include pipelining, provision of parallel functional units, array processing and multiprocessing.

Another line of parallel development is the spectacular growth and advances made in the microelectronics industry. The technological progress in VLSI has heralded the arrival of chips which have over a million transistors and which operate at very high speeds. With decreasing hardware costs, it is now possible to connect several thousands of processors. The Connection machine consisting of 65536 processors is a good example of a massively parallel machine [2].

1.1 Distributed Computing

Parallel or distributed computing will be a key issue in the design of supercomputers [3]. The term *distributed computing* can be interpreted in two different ways. The first implies that a distributed computer system is a collection of processor-memory pairs interconnected by a communication network and logically integrated in a distributed system. The communication topology may be a local area network or a geographically distributed wide area network. The potential advantages of a distributed system are many-fold. Increased performance-to-cost ratio, high throughput, good reliability, efficient resource sharing and extensibility. Examples of geographically distributed computer networks include ARPANET, INTERNET, and BITNET [4].

The second meaning is that they denote a collection of processors that are located within a short distance of each other. Their main purpose is to execute jointly a computational task assigned to them. Parallelism inherent in the job makes it possible to identify the concurrent parts and distribute them to the various processors interconnected in some way, namely by a shared bus, memory or some kind of a special interconnection network. We use the general term

parallel system to denote a computer system incorporating several processing elements capable of cooperating in exploiting the inherent parallelism of a job. Parallel systems are classified further as

1. SIMD or MIMD depending on whether they can process single or several instruction streams with multiple data streams in parallel.

2. loosely or tightly coupled depending on the interprocessor communication cost.

3. fine or coarse grained depending on the size of the cooperating process entities.

Parallel systems may be pipelined or organized as systolic arrays [5, 6] or dataflow systems [3].

In this thesis, we use the term *concurrent computer systems* to denote a parallel system comprising a multitude of processor-memory nodes interconnected via a specialized high speed point-to-point interconnection network and cooperating in the processing of a particular task.

1.2 Design Issues for Parallel Computers

The major issues confronting the design of concurrent computer systems include

- Resource management
- Information exchange mechanisms
- System software and parallel languages
- Interconnection methods.

In order to make effective use of multiprocessor systems, it is necessary to ensure that all Processor Elements (PE) are utilized fully. Resource management addresses the following tasks:

- Partitioning
- Scheduling & load balancing
- Synchronization.

These directly affect the degree of parallelism, resource utilization rate and the system throughput of the parallel computer. The general methods of interprocessor communication include shared variables, message passing, name passing and value passing. There is also a concomitant need for developing parallel programming languages and algorithms and compilers for efficient utilization of the multicomputer system.

Communication efficiency and hardware connectivity are major concerns in the design of a cost-effective parallel system. The total execution time of a parallel algorithm on a parallel system is dependent on the interprocessor communication time besides the execution time. In this context it would be

appropriate to state *Amdahl's law* [7] for the execution time of a task in a parallel computer system.

Amdahl's Law 1.2.1 Given f_n the portion of code that can run in parallel over n nodes, then the speed-up s_n achievable is given by

$$s_n = \frac{1}{(1 - f_n) + \frac{f_n}{n} + \sigma} \tag{1.1}$$

where σ is a monotonically increasing function of n that corresponds to the communication overhead.

The communication time is dependent on several factors such as the interconnection network, storage scheme for data and so on. Efforts have been made in the past to design efficient interconnection networks. In the next section, we address the importance of interconnection networks in a parallel computer system.

1.3 Interconnection Networks

A typical concurrent computing system is shown in Fig. 1.1. The interconnection network facilitates communication between the various processor-memory nodes of the system. An interconnection network is modeled by an undirected graph $\mathcal{G}(V, E)$ whose vertices represent processors and whose edges represent bidirectional communication links. Switches present in the network perform an intermediate communication function called *routing*. Routing is the decision process which determines the path that a message should follow from a source to a destination node in the system.

One can cite several examples of interconnection networks from the open literature [8]. Some of the important interconnection networks include the ring,

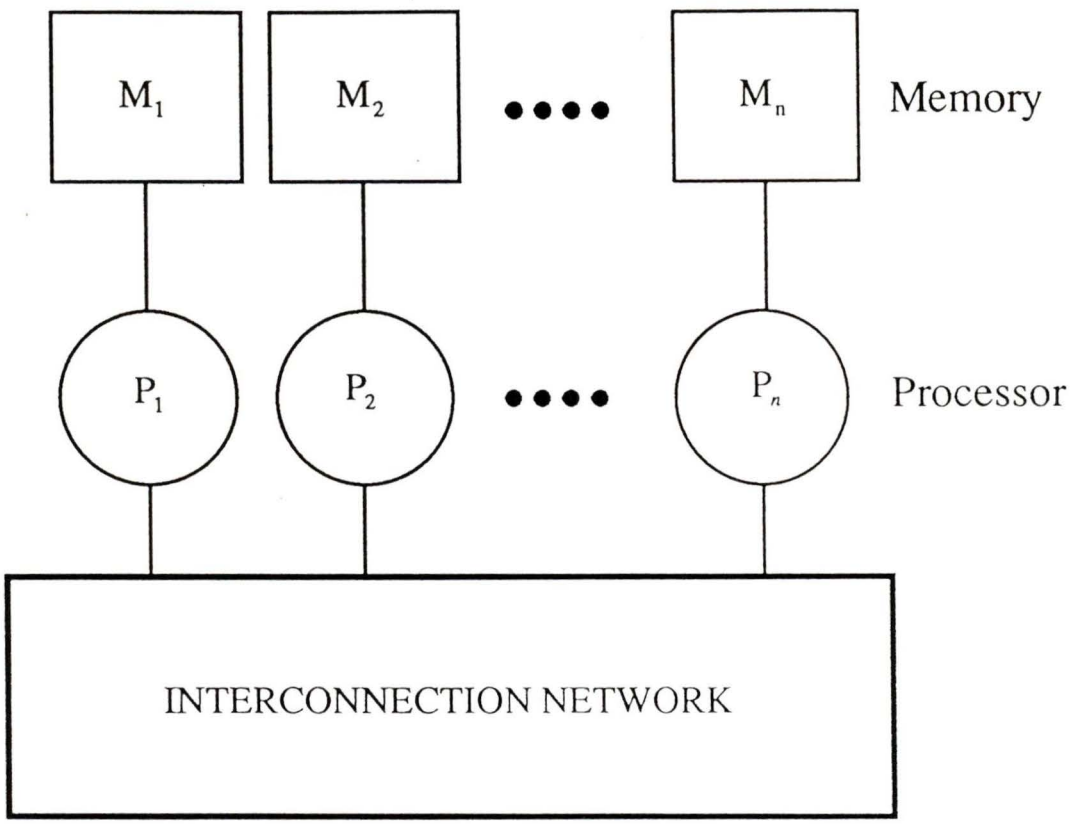


Figure 1.1: General structure of a concurrent computing processing system

the star, the tree, the near-neighbour mesh, the fully connected network, the cross-bar, the shuffle-exchange [11], the omega [12], the delta [13], the banyan [14] and the Hypercube [15].

An interconnection network may be structured as a bus or a multistage network or employ a multitude of dedicated links. In the case of bus oriented networks such as in *Ethernet* [16], a variety of bus arbitration techniques such as CSMA/CD, token passing or time-sliced methods are employed to avoid contention or deadlocks in the networks. However bus architectures affect system performance due to high latency and single point failures. From the fault tolerant point of view they are unsuitable for constructing high performance systems. Further, a fully connected topology is not viable for system expansion beyond a certain degree in view of the large number of interconnections.

1.4 Survey of Parallel computers

The earliest approach in using multiple CPUs for reliability dates back to PILOT computer system in the late 1950s [18]. Later developments include the ILLIAC IV [19] array processor which consists of 64 processing elements (PE) each containing an ALU and local memory and interconnected via a two dimensional mesh structure. The C.mmp machine which originated at the Carnegie Mellon University consists of 16 processors with 16 memory modules connected over a cross-bar switch [20].

The Univac 1100/94, Cray X-MP are examples of shared memory, tightly coupled MIMD machines with dedicated/bus oriented interconnection [3] but supporting a limited number of processing elements. The popular interconnection schemes for loosely coupled MIMD computers are the Hypercube, ring,

butterfly [26], Hypertrees [27] and Hypernets [28].

Examples of machines that incorporate a Hypercube topology include Intel iPSC [29], AMETEK System 14, NCUBE/10 [15], Cosmic Cube [30] and the Connection machine [2]. The CDC Cyber plus machine [31] has a ring topology while the BBN Butterfly uses the butterfly network.

1.5 Hypercubes and Hypercycles

The Hypercube [1, 3] topology is a popular interconnection topology for concurrent computer systems. The Hypercube or binary n-cube consists of a pair of nodes in each dimension connected to each other such that there are a total of $N = 2^n$ nodes in the system. Extensive research has been carried out in studying the performance of Hypercube based multicomputer systems. Bhuyan and Agarwal [32, 33] have introduced the concept of generalized Hypercubes and Hyperbus based structures.

The advantages of using regular and structured interconnection networks have been exemplified in many practical implementations such as MARK II Hypercube of Caltech/JPL [40]-[42], NCUBE and Intel iPSC. Simple routing is accomplished in these machines by decentralizing the routing process.

In a concurrent computer system, the interconnection topology and the allocation of processing elements to meet certain tasks is dictated by the application requirements. In some cases, an increase in performance may be the objective. In airborne computers, for instance, fault tolerance and space-weight limitations are major issues. Another important parameter of multicomputer systems is system expansibility and specifically the smallest increment by which the system can be expanded in a useful way. In all these instances the underlying interconnection network should be modular, flexible and lend itself to changes.

Hypercycles proposed in [44]-[46], are product graphs of "basic" graphs that allow a richer set of component sub-graphs ranging in complexity from simple rings to a fully connected network. The Hypercycles being regular graphs retain the advantages of simple routing and structural generality similar to the Hypercubes. In addition, Hypercycles can grow in small increments as opposed to Hypercubes where the number of nodes is always given as a power of two. Since Hypercycles represent a class of graphs rather than isolated graphs, they offer the flexibility of tailoring them to suit applications that demand computation, communication, fault-tolerance and reliability.

1.6 Previous Work

In [45], an extensive work on the performance analysis of a particular routing scheme called the *Backtrack-to-the-Origin-and-Retry routing* (BTOR) was conducted and a comparison was made with other routing schemes. Furthermore, a partial design of the switching hardware of the Hypercycle networks was presented in [45].

1.7 Outline of Thesis

The focus of this thesis is the VLSI implementation of the router for the BTOR scheme for the Hypercycle based interconnection networks. The thesis is organized as follows:

Chapter 2 introduces the basic terminologies associated with Hypercycle networks. In particular, we discuss the graph structure of the Hypercycles along with analytical expressions for single and multidimensional routing.

Chapter 3 begins with a survey of various routing schemes for computer networks. The *Backtrack-to-the-origin-and-retry-routing scheme* is considered and the framework for constructing the router engine is formulated. The functions of the various modules of the router are also discussed.

Chapter 4 is devoted to an analysis of design methodologies pertaining to VLSI systems. Various issues inherent in the IC design cycle spanning from the specification phase to fabrication and testing are discussed. A design for the Next Port Generator which is one of the modules in the Hypercycle router is proposed and simulation results are presented. The circuit has been implemented in the 3 μ CMOS3DLM technology.

Chapter 5 is concerned with the principles of residue arithmetic and number theory which are used for building a *modulo extractor* for the Hypercycle router. The asymptotic area-time complexity of the proposed structure is derived and the circuit has been simulated and implemented in the 3 μ CMOS3DLM technology.

Chapter 6 is devoted to another functional block of the proposed router, namely, the design of a random number generator. A survey of various types of random number generators is conducted and a mixed congruential generator is chosen. Various statistical tests are conducted on the chosen model and its per-

formance is compared with other types of generators including the Unix RNG. The structure has been simulated and designed in the CMOS3DLM technology.

Chapter 7 is concerned with an analysis of a *binary search* algorithm for implementing the line selector circuit for the router. An area-time analysis of the structure is conducted and the circuit is realized in hardware.

Chapter 8 deals with actual hardware realization of the Hypercycle router which is the ultimate goal of this project. The specifications of the router are defined and the logical schema is synthesized. The proposed structure has been implemented in the 1.2 μ CMOS4S technology.

Chapter 9 is concerned with testing and verification of the implemented chips. Functional, fault and parametric testing are conducted on the fabricated chips using patterns generated by deterministic and random methods and test results including fault coverage, chip yield, AC/DC characteristics are enlisted.

Finally conclusions, critical evaluation and suggestions for further research are given in **Chapter 10.**

Appendices A-D present the schematics of the various modules of the Hypercycle router along with pin descriptions, pin layout and bonding diagrams.

Chapter 2

Hypercycles - A Perspective Outlook

2.1 Introduction

A multi-processing system is characterized by the ability of each processor in the system to share a set of memory modules, I/O devices and communicate with other processors. A parallel computer is built by interconnecting many processors and memory modules through an appropriate interconnection network. The execution of many real-time programs on a parallel computer is *communication limited rather than processing power limited.* Therefore, the underlying processor interconnection network plays an important role in determining the performance of the system.

A major factor in realizing highly reliable and efficient networks with a limited number of links is finding a graph with minimal diameter (distance) and maximal connectivity (degree) for a given number of nodes [35]. The diameter is related to efficiency factors such as network latency and transmission capacity while the connectivity is related to fault tolerant aspects of the system.

Message passing parallel machines such as the NCUBE [15], Cosmic Cube [30], MAX [42, 43] consist of several processing nodes that interact via messages exchanged over the communication channels linking these nodes into a single entity. There are several ways to interconnect these processing nodes. A regular and well structured interconnection network is needed to achieve high throughput and efficiency besides fault-tolerance and reliability. Especially in power, mass and space limited environments such as in spacecraft, airborne computers and satellites, it is imperative to use an efficient interconnection network for a a parallel machine in order to achieve high performance within strict constraints.

In this chapter, the Hypercycle graphs are introduced. The basic definitions and terminologies associated with Hypercycles are discussed along with expressions for connectivity, degree, diameter and average distance for the Hypercycles. Single and multi-dimensional routing in Hypercycles are also considered.

2.2 Hypercycles

Hypercycles [46] are a class of multidimensional graphs that are generalizations of the n-cube. Hypercycles offer simple routing and the ability, given a fixed degree, to choose among a number of alternate size graphs. These graphs can be used in the design of interconnection networks for concurrent systems tailored specifically to the topology of the application. Hypercycles can be construed as products of basic graphs that allow a richer set of component subgraphs ranging in connectivity from a simple ring to a fully connected graph. The power of the Hypercycles stems from the fact that the graph topology can be configured to match the requirements of a large variety of applications. In the following sections, we define the basic terminology and briefly discuss the theory associated with Hypercycles graphs.

2.3 Mixed Radix Number System

The Mixed Radix Number (MRN) system [33, 48] is a positional number representation which is a generalization of the standard b-base representation. It allows for each position of the representation to follow its own base independent of the others.

Given a decimal number M which can be expressed as a product of r factors such that $M = m_1 m_2 \dots m_r$ then any number $0 \leq X \leq M-1$ can be expressed as

$$(X)_{m_1 m_2 \dots m_r} = x_1 x_2 \dots x_r|_{m_1 m_2 \dots m_r}, \quad 0 \leq x_i \leq m_i - 1, i = 1, \dots r \tag{2.1}$$

The x_i's are chosen such that

$$X = \sum_{i=1}^{r} x_i w_i \qquad \text{where } w_i = \frac{M}{m_1 m_2 \dots m_i} \tag{2.2}$$

The MRN system is used in defining the topology of the Hypercycle graph as shown later.

2.4 Hypercycle Graphs

A comprehensive description of Hypercycle graph structures can be found in [44, 45, 46]. A r-dimensional Hypercycle is a regular undirected graph defined by

$$\mathcal{G}_m^{\rho} = \{\mathcal{N}_m^{\rho}, \mathcal{E}_m^{\rho}\} \tag{2.3}$$

where

1. $\mathcal{N}_m^{\rho}$ is the set of nodes

2. $\mathcal{E}_m^\rho$, the edges in the graph

3. $m = m_1, m_2, \ldots m_r$, nodes defined in the mixed radix number system

4. $\rho = \rho_1\rho_2 \ldots \rho_r, \rho_i \leq \frac{m_i}{2}$, the connectivity vector which determines the connectivity of the graph from a cycle($\rho_i = 1$) to the fully connected network[1] (for which $\rho_i = \lfloor \frac{m_i}{2} \rfloor$) and

5. $\mathcal{N}_m^\rho = \{0, 1, 2, \ldots M-1\}$ where M is the total number of nodes in the graph.

Given $\alpha, \beta \epsilon N_m^\rho$, if α, β can be represented in the MRN as

$$(\alpha)_{m_1 m_2 \ldots m_r} = \alpha_1\alpha_2 \ldots \alpha_r \tag{2.4}$$

$$(\beta)_{m_1 m_2 \ldots m_r} = \beta_1\beta_2 \ldots \beta_r \tag{2.5}$$

then $(\alpha, \beta) \epsilon \mathcal{E}_m^\rho$ iff there exists some j for $0 \leq j \leq r$ such that

$$\beta_j = (\alpha_j \pm \xi_j) \bmod m_j, \text{for } 1 \leq \xi_j \leq \rho_j \tag{2.6}$$

$$\alpha_i = \beta_j \quad \text{for } i \neq j \tag{2.7}$$

The degree d of the Hypercycle graphs is given by

$$d = \sum_{i=1}^{r} f(m_i, \rho_i) \quad \text{where} \tag{2.8}$$

$$f(m_i, \rho_i) = \begin{cases} 2\rho_i & \text{if } 2\rho_i < m_i \\ m_i - 1 & \text{if } 2\rho_i = m_i \end{cases} \tag{2.9}$$

The diameter k of the graph is given by

$$k = \sum_{i=1}^{r} \left\lceil \frac{\lfloor \frac{m_i}{2} \rfloor}{\rho_i} \right\rceil \tag{2.10}$$

[1]The function $\lfloor y \rfloor$ denotes the largest integer that is less than or equal to y.

The general expression for the number of edges $\mathcal{E}_m^\rho$ is given by

$$\mathcal{E}_m^\rho = \frac{Md}{2} \tag{2.11}$$

where M is the number of nodes in a Hypercycle of degree d.

Examples of Hypercycle graphs are illustrated in Fig. 2.1. G_8^1 is a Hypercycle consisting of 8 nodes connected as a ring ($\rho = 1$). The degree d of this graph is 2 while the diameter is 4. The number of edges $\mathcal{E}_8^1$ is 8. G_8^2 represents a ring with chords. G_6^3 is a fully connected Hypercycle with 6 nodes. G_{44}^{11} depicts a nearest neighbour mesh with 4 nodes in each dimension. The binary n-dimensional cube can be expressed as a Hypercycle with $M = \underbrace{2 \times 2 \times \ldots \times 2}_{n} = 2^n$ and $\rho = \underbrace{11\ldots1}_{n}$. For example, in Fig. 2.1 the binary 3-cube is represented as a Hypercycle given by G_{222}^{111}. Similarly, the binary 4-cube is expressed as G_{2222}^{1111}.

2.4.1 Average distance calculation

Given an r-dimensional Hypercycle G_m^ρ , the number of nodes along any dimension i, that are at distance l from a source node $(\alpha)_{m_1 m_2 \ldots m_r} = \alpha_1 \alpha_2 \ldots \alpha_r$ is given by [45]

$$n_l^i = \begin{cases} 2\rho_i & \text{if } l\rho_i < \lceil \frac{m_i}{2} \rceil \\ (m_i - 1) - 2(l-1)\rho_i & \text{if } \lceil \frac{m_i}{2} \rceil \leq l\rho_i \leq \frac{m_i - 1 + 2\rho_i}{2} \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

Given an arbitrary node $(\alpha)_{m_1 m_2 \ldots m_r} = \alpha_1 \alpha_2 \ldots \alpha_r$, we can find other nodes

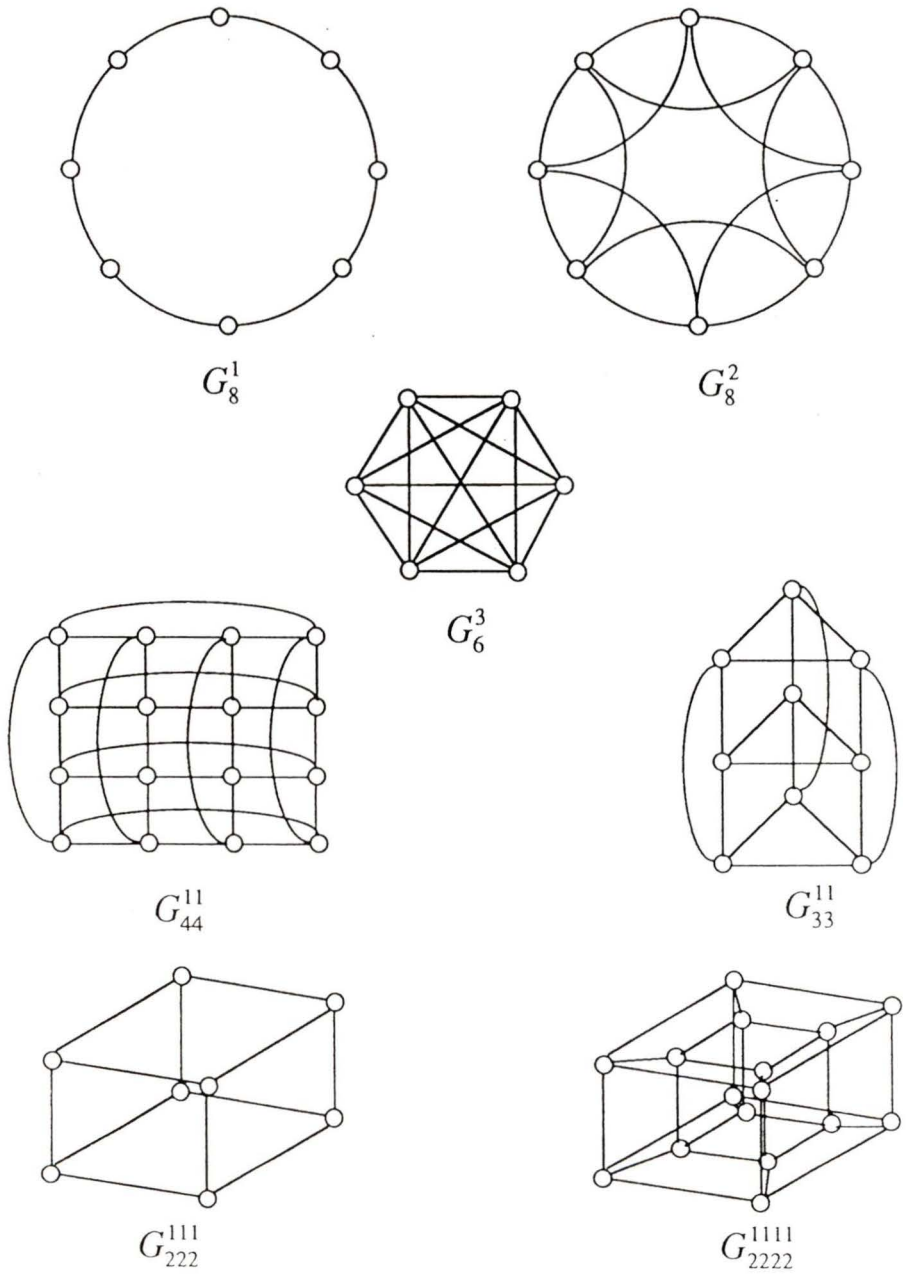


Figure 2.1: Hypercycle graphs

in the network which are at distance 1 in all r-dimensions and this is given by

$$n_1 = \sum_{i=1}^{r} n_l^i \qquad \text{for nodes at distance 1}$$

In general

$$n_l = \sum_{l_1+l_2+\ldots+l_r=l} \prod_{i=1, l_i \neq 0}^{r} n_{l_i}^i \qquad \text{where } l_1, l_2, \ldots l_r \geq 0 \tag{2.13}$$

For the case of a fully connected Hypercycle ($\rho_i = \lfloor \frac{m_i}{2} \rfloor$)

$$n_l = \sum_{i_1=1}^{r-l+1} \sum_{i_2=i_1+1}^{r-l+2} \cdots \sum_{i_l=i_{l-1}+1}^{r} (m_{i_1} - 1)(m_{i_2} - 1) \ldots (m_{i_l} - 1) \tag{2.14}$$

The average distance between any two nodes of a Hypercycle graph is given by

$$\delta = \frac{\sum_{l=1}^{r} l n_l}{M - 1} \qquad \text{where } M = m_1 m_2 \ldots m_r \tag{2.15}$$

2.5 Routing in Hypercycles

The routing properties of Hypercycles [45, 46] are similar to those of the n-cube. Given two nodes

$$(\alpha)_{m_1 m_2 \ldots m_r} = \alpha_1 \alpha_2 \ldots \alpha_i \ldots \alpha_r$$

and

$$(\alpha^*)_{m_1 m_2 \ldots m_r} = \alpha_1 \alpha_2 \ldots \xi \ldots \alpha_r \tag{2.16}$$

whose MRN addresses differ in the i^{th} position, it can be easily seen that in dimension i, the two nodes are separated by a distance of at most $\left\lceil \frac{\lfloor \frac{m_i}{2} \rfloor}{\rho_i} \right\rceil$.

A walk from node α to α^* can be constructed as follows:

$$\begin{array}{l} \alpha_1\alpha_2 \ldots \alpha_i \ldots \alpha_r; \\ \alpha_1\alpha_2 \ldots \xi_1 \ldots \alpha_r; \\ \alpha_1\alpha_2 \ldots \xi_2 \ldots \alpha_r; \\ \qquad \vdots \\ \alpha_1\alpha_2 \ldots \xi \ldots \alpha_r; \end{array} \tag{2.17}$$

such that[2]

$$\xi_{j_i+1} = \begin{cases} (\xi_{j_i} + \rho_i) \bmod m_i & \text{if } \left[(\xi - \xi_{j_i}) \bmod m_i = |\xi_{j_i}, \xi|\right] > \rho_i \quad \text{(a)} \\ (\xi_{j_i} + |\xi_{j_i}, \xi| \bmod \rho_i) \bmod m_i & \text{if } \left[(\xi - \xi_{j_i}) \bmod m_i = |\xi_{j_i}, \xi|\right] > \rho_i \quad \text{(b)} \\ & \text{and } |\xi_{j_i}, \xi| \bmod \rho_i \neq 0 \\ (\xi_{j_i} - \rho_i) \bmod m_i & \text{if } \left[(\xi j_i - \xi) \bmod m_i = |\xi_{j_i}, \xi|\right] > \rho_i \quad \text{(c)} \\ (\xi_{j_i} - |\xi_{j_i}, \xi| \bmod \rho_i) \bmod m_i & \text{if } \left[(\xi j_i - \xi) \bmod m_i = |\xi_{j_i}, \xi|\right] > \rho_i \quad \text{(d)} \\ & \text{and } |\xi_{j_i}, \xi| \bmod \rho_i \neq 0 \\ \xi & \text{if } |\xi_{j_i}, \xi| \leq \rho_i \quad \text{(e)} \end{cases} \tag{2.18}$$

with $\xi_0 = \alpha_i$, $\xi_{max} = \xi$.

Equations (2.18a-e) provide the analytical expressions for computing the minimal length paths to be taken from the source to the destination nodes in any dimension. Parts (a) and (c) of eqn.(2.18) constitute the *greedy strategy* since a maximum hop of ρ_i is taken towards the destination. On the other hand, parts (b) and (d) of the above equation constitutes what is called a *liberal strategy* since it provides alternate paths to be taken earlier similar to that given by part (e) of eqn.(2.18). Note that the liberal strategy can be employed when there is one and only one length of hop smaller than the maximal jump ρ_i. The remaining jumps to the destination are guaranteed to be maximal [130] because

$$|(\xi_{j_i} \pm |\xi_{j_i}, \xi| \bmod \rho_i) \bmod m_i, \xi| \bmod \rho_i \;=\; 0 \tag{2.19}$$

[2] $|\mu, \nu| = \min\{(\mu - \nu) \bmod m, (\nu - \mu) \bmod m)\}$, the distance between two integers modulo m

2.5.1 Multi-dimensional Routing

Multidimensional routing in Hypercycles is akin to single dimensional routing given by eqn.(2.17) except that when the source address is modified each time, a decision has to be made in selecting the eligible dimension for effecting the change.

Given a source node $(\alpha)_{m_1 m_2 \ldots m_r} = \alpha_1 \alpha_2 \ldots \alpha_r$ and a destination node $(\beta)_{m_1 m_2 \ldots m_r} = \beta_1 \beta_2 \ldots \beta_r$, if q_i denotes the distance along dimension i, then the total distance $\mathrm{dis}(\alpha, \beta)$ is given by

$$\mathrm{dis}(\alpha, \beta) = q = \sum_{i=1}^{r} q_i \tag{2.20}$$

As explained in [46], there are a total of

$$\begin{aligned} I &= \begin{pmatrix} q \\ q_1, q_2, \ldots, q_r \end{pmatrix} \\ &= \frac{q!}{q_1! q_2! \ldots q_r!} \end{aligned} \tag{2.21}$$

possible combinations of distinct walks of length q between α and β. These paths can be constructed by modifying the source address sequentially, each time substituting a source digit by an arbitrary walk digit chosen from the eligible dimension until the final destination is reached. The whole sequence is determined by the routing equations given by eqn.(2.18). As an illustration, the following walk connects the source (α) to the destination node (β) by successive modification of the address digits.

$$
\begin{aligned}
\textbf{source} = \; & \alpha_1\alpha_2 \ldots \alpha_i \ldots \alpha_j \ldots \alpha_r; \\
& \alpha_1\alpha_2 \ldots \xi_1 \ldots \alpha_j \ldots \alpha_r; \\
& \alpha_1\alpha_2 \ldots \xi_1 \ldots \psi_1 \ldots \alpha_r; \\
& \alpha_1\alpha_2 \ldots \xi_2 \ldots \psi_1 \ldots \alpha_r; \\
& \alpha_1\alpha_2 \ldots \xi_2 \ldots \psi_2 \ldots \alpha_r; \\
& \qquad \vdots \\
& \alpha_1\alpha_2 \ldots \beta_i \ldots \psi_2 \ldots \alpha_r; \\
& \qquad \vdots \\
& \beta_1\beta_2 \ldots \beta_i \ldots \beta_j \ldots \beta_r \quad = \textbf{destination}
\end{aligned}
\tag{2.22}
$$

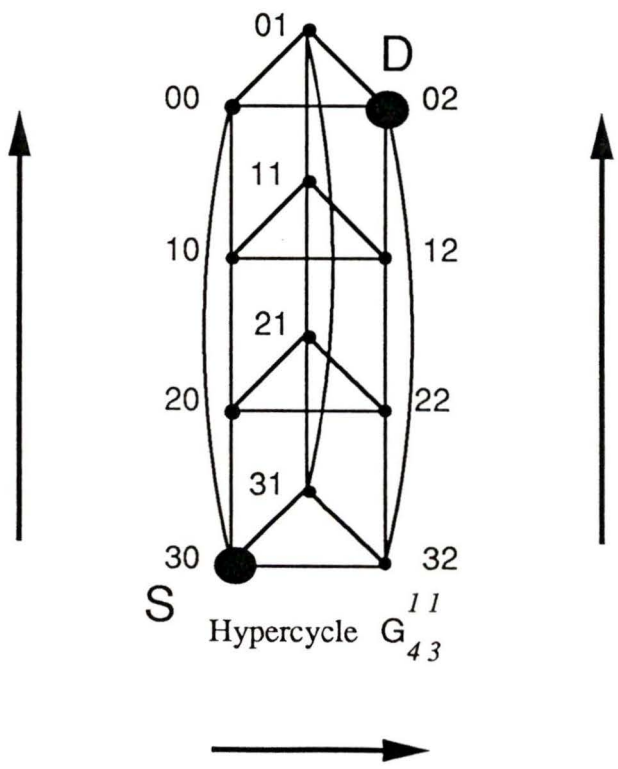


Figure 2.2: Routing in Hypercycle G_{33}^{11}

Consider the Hypercycle $\mathcal{G}_{43}^{11}$ shown in Fig. 2.2 where each node is expressed as a MRN. If the source and destination addresses are 30 and 02 respectively, then the possible routes determined by eqns.(2.18) are:

1. 30-32-02
2. 30-00-02.

2.6 Conclusion

In this chapter, Hypercycle graphs were introduced and expressions for average distance, density, diameter, degree were given. Routing constitutes an essential and integral part of an interconnection network. Single and multidimensional routing were considered for the Hypercycles and they guarantee that routing from one node to the other is of minimal distance. The throughput of the network is directly related to the average distance between two nodes and the number of alternate paths available for routing. Hypercycle networks have been developed primarily for message passing concurrent computers.

The technological strides made in integration coupled with the rapidly decreasing costs have made it possible to integrate the functionality necessary for routing in Hypercycle based concurrent systems using a limited number of VLSI components. In the following chapters, the routing algorithms for the Hypercycles are discussed and the architecture for their hardware implementation is developed.

Chapter 3

Router Architecture for BTOR Scheme

3.1 Introduction

In the previous chapter, the routing properties of Hypercycles were discussed and we showed that it establishes at least one minimal distance path from a source to a destination. In this chapter we give a brief overview of the various types of routing strategies. The Backtrack-to-the-origin-and- retry routing (BTOR) scheme [45] for Hypercycles is discussed and its performance is compared with other types of routing schemes. Based on these results, the architecture of the router engine which controls the above routing scheme is formulated. Finally, the function and organization of the various modules of the router are discussed.

3.2 Routing Schemes

Many developments in the design of routing algorithms for computer networks have been reported in recent years. An extensive survey can be found in [49]-[50]. The routing algorithm determines the path that a message takes through the

network in order to reach the destination. Since the path taken is not known in advance, each node determines individually an address onto which its outgoing links are connected. This process is repeated at each intermediate node till the destination is reached and is dependent on the network connectivity, network traffic, link failures and the routing algorithm.

There are two major switching methodologies for message transmission, namely

1. *circuit switching*

2. *packet switching.*

In circuit switching, a physical path is actually established between the source and destination addresses of the network before the data is transmitted. On the other hand, in packet switching, the information to be transported is split into small messages called packets which move through the network in a discrete fashion. Each packet must contain a destination address so that the packets can be routed. One obvious limitation of packet switched networks is the large transmission time caused by queueing at each intermediate node. Furthermore, there is the possibility of information being lost due to overflow of the buffers owing to a sudden surge in network traffic.

There exist several routing paradigms for computer communication which utilize circuit and/or packet switching techniques. Some of the important routing policies include *store-and-forward [4], worm-hole [56], staged [57], circuit [58], adaptive routing [59]* and so on. In the store-and-forward routing scheme, the data transmitted from one node to another are temporarily stored in the memory of the intermediate nodes and later forwarded to the destination node. If a message travels N hops, it is temporarily stored in $N-1$ nodes. In worm-hole routing, instead of storing a message completely in each node and then

transmitting it to the next node, the header of the message is transmitted to the outgoing channels. Each word is represented by means of flow control digits called *flits* which can be buffered at each stage. The construction of pipelines is therefore possible. The circuit, staged and adaptive routing paradigms use the circuit switching methodology by physically establishing a path between the source and the destination. When the path is blocked at some point in the network, they either return to the originating node as in the BTOR scheme [46] or backtrack to the previous node as in the Hyperswitch routing [40].

Interconnection networks are evaluated in terms of *latency* which is the average time taken for a message to reach its destination and in terms of the *throughput* of the network, which is the number of messages successfully routed per unit time in the network [52] and the ability to detect and or avoid deadlocks. In the following sections, we discuss the merits and demerits of the various routing algorithms for interconnection networks.

3.2.1 Deadlock Avoidance

Deadlock avoidance is an important metric to judge the performance of any routing strategy since deadlocks can cripple the entire network affecting throughput/performance drastically. Deadlocks in an interconnection network is defined as the situation wherein no process can proceed without acquiring a resource already held by some other process. Deadlocks can occur when resources (links) are assigned so that the completion of a partial path requires a segment which has been already allocated to some other path which in turn needs a link from the first partial path.

A concurrent computer system consisting of multiple processors linked by a cyclic interconnection can be deadlocked. For example, consider the 4-cycle

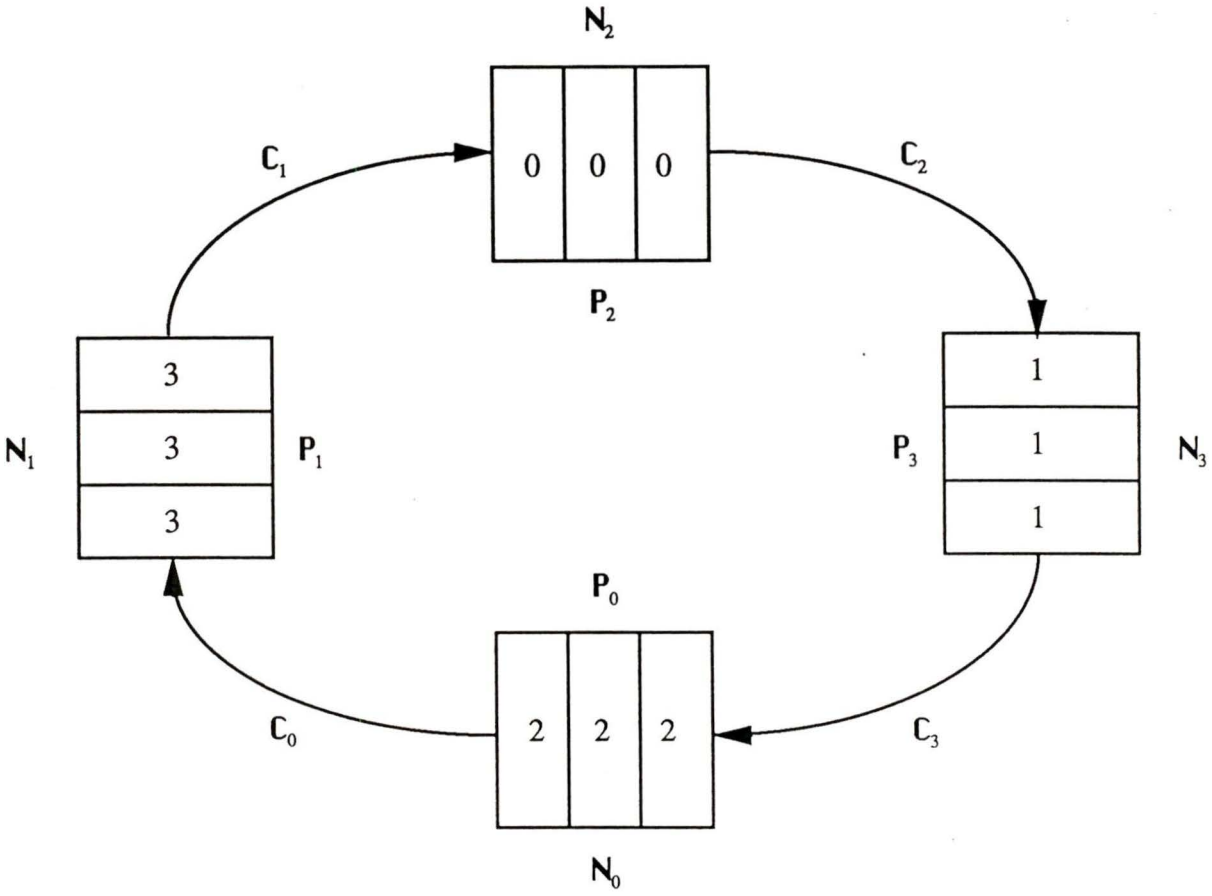


Figure 3.1: Deadlock in a 4-cycle network

network as shown in Fig. 3.1. The numbers $N_0, \ldots N_3$ represent the computer nodes while $C_0, \ldots C_3$ denote the links and $P_0, \ldots P_3$ represent concurrent processes. The numbers within each node indicate the destination addresses of the messages stored in the buffers. Nodes $N_0, \ldots N_3$ form a loop in the clockwise direction. Assume that messages are bound for opposite nodes in the network through circuit switching. A deadlock situation arises when the resources held by processes P_0, P_1, P_2, P_3 are required by P_3, P_0, P_1, P_2 respectively resulting

in a deadlock. An analogous situation arises in packet switching when the queues of the individual nodes become full with messages destined for opposite nodes and the system gets deadlocked. Thus, the presence of cycles in an interconnection network is a potential threat to deadlock free operation. To avoid deadlocks, either the routing algorithm should be designed to detect them or special provisions should be incorporated to prevent their occurrence.

The e-cube [47] routing guarantees deadlock free routing in binary n-cubes (Hypercubes). Each node n_k of the Hypercube has n edges, one for each dimension, labeled $c_{0k}, c_{1k}, \ldots c_{(n-1)k}$ where k is a n-digit number. A message bound for n_l from node n_k is routed along channel c_{ik} where i is the position of the most significant bit in which k and l differ. Thus messages are routed in order of decreasing dimension and the absence of cycles makes it deadlock free. The e-cube routing makes use of only one path from source to destination in contrast to $\mathcal{O}(n!)$ paths that are available and hence the bandwidth utilization is poor.

In [47], a modified version of the e-cube routing called *virtual channel* routing that avoids deadlocks is presented. It has been shown that the algorithm can be used for cyclic graphs like rings, toroidal meshes and n-cubes.

3.3 Backtrack-to-the-Origin-and-Retry Routing Scheme

Hypercycles are regular, symmetric, cyclic undirected graphs and because of the existence of cycles in each dimension, the use of an e-cube type of routing that avoids deadlocks is impossible.

A *Backtrack-to-the-Origin-and-Retry routing* strategy for Hypercycles has been proposed in [45] to avoid any impending and imminent deadlocks. Deadlock avoidance is dynamic and depends on the network traffic and load of the system. In the BTOR scheme:

1. Paths are formed by randomly selecting from the set of available intermediate links.

2. Blocked partial paths are dissolved and requests are requeued at the origin.

The basic tenet of the BTOR scheme is to identify at each node the set of potential nodes for routing as defined by eqn.(2.18). For all these potential nodes, the actual free ports (links) which are available are identified. One of these free ports is selected at random and the message is routed through the corresponding port to the next node. This process continues till the message reaches its destination or no free ports are available at some intermediate point. This implies that the partial path established thus far cannot be continued any further. Hence to avoid possible deadlocks, the already established partial path till the stage when the block was encountered is *dissolved.* A new attempt is made to find another route for the message. This routing strategy avoids deadlocks through backtracking and also guarantees that the established path will be of minimal length [45] as determined by eqn.(2.18).

3.4 Performance Analysis

In [45], extensive simulation studies of the performance of the BTOR scheme were reported. The throughput and delay characteristics of the BTOR were established for several topologies of Hypercycles including binary n-cubes. In addition, delay characteristics of the e-cube routing for binary n-cubes were also computed. A comparison of the performance characteristics of these two routing algorithms yielded the following.

The performance of the BTOR scheme was superior to that of the e-cube routing under *low to moderate* loads. This can be attributed to the fact that the BTOR scheme uses alternate paths to the destination in contrast to the single path routing by the e-cube algorithm. Under *heavy* loads, the e-cube routing performed marginally better than the BTOR scheme. The reason for this behaviour is that there is a high probability for a path being blocked under heavy loads and when the messages have to be transmitted over a greater distance. This consequently results in frequent dissolution of the partial paths resulting in lower throughput. The conclusion is that the BTOR scheme provides throughput and delay which are better than those of e-cube routing under average loads and it avoids deadlocks by dissolving blocked partial paths.

3.5 Routing Engine

A concurrent computer system consists of a number of processing nodes complete with local memory and communication pathways to its nearest neighbours as determined by the topology chosen. A routing engine offloads the processor from the task of routing intermediate messages towards their destination. If the Hypercycle network topology is chosen, each processor in the system is equipped

with routing hardware called the *Hypercycle Router Engine* (HRE) illustrated in Fig. 3.2. The engine consists of three modules, namely

1. Universal switch
2. Controller
3. Hypercycle Router.

The Controller provides the interface between the CPU, universal switch and the router. In addition it implements the chosen routing strategy (in our case the BTOR scheme).

The switch is essentially a cross-bar with incoming and outgoing ports connected to the various nodes of the network depending on the configuration. The router, given the current node address, message destination address and link availability, decides the next link to be included in the message path. This information is supplied to the controller which in turn sets the switch for the continuation of the path. Furthermore the router informs the controller in the case of a block. The routing strategy implemented by the controller then decides on a subsequent action.

Requests for network access are initiated by a CPU which acts as the source node sending the destination address to the HRE. This protocol is interpreted by the controller which loads the necessary information to the Hypercycle router. The router then determines the appropriate port to route the messages based on eqns.(2.18) in conjunction with the network parameters and the output is used by the controller to switch the cross-bar. Each HRE is initially configured by the corresponding node with information such as the current address of the node, the population of the network, connectivity and so on. This thesis is

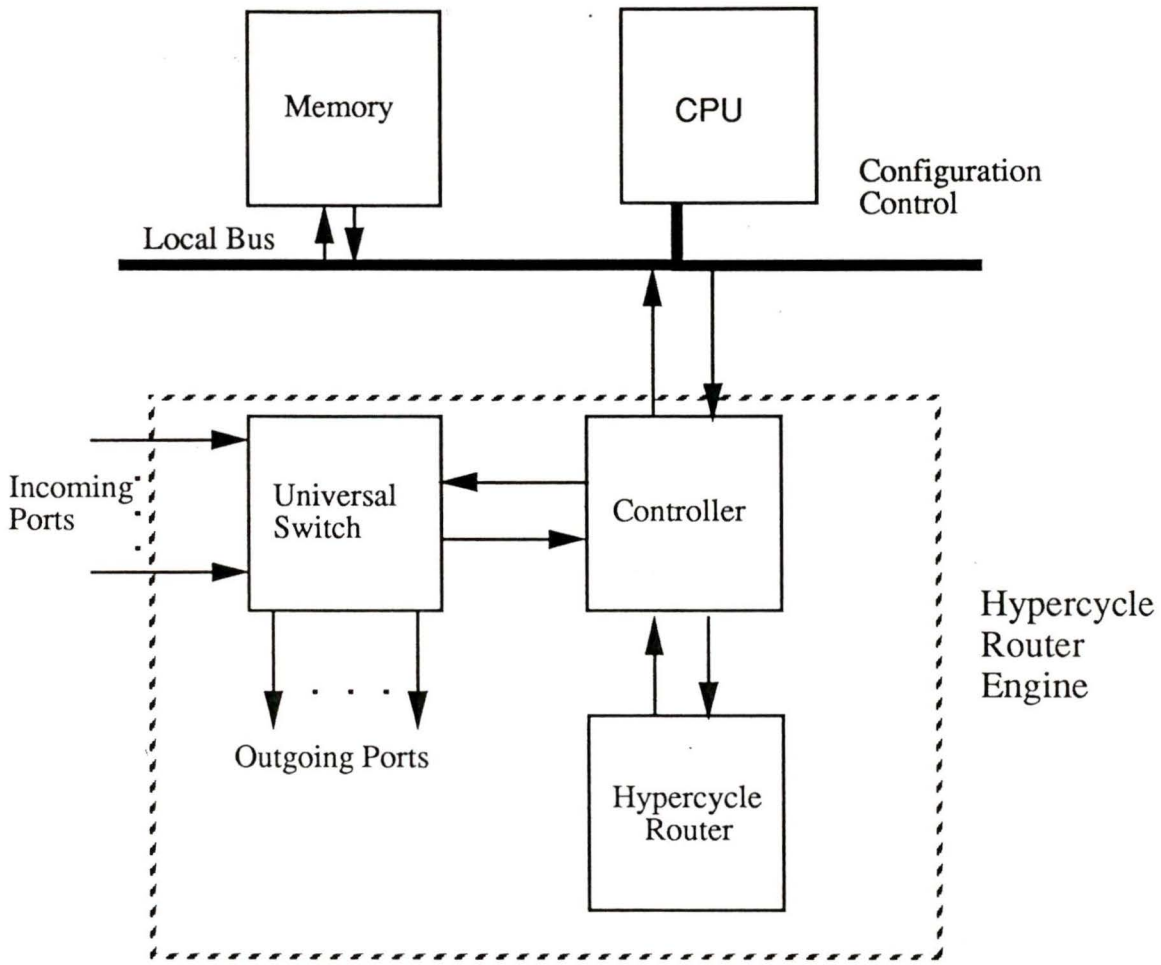


Figure 3.2: Structure of a single processor node in the Hypercycle networks

concerned mainly with the design and implementation of the router which will be the subject of discussion in the following sections.

3.6 Structure of the Hypercycle router

The schematic diagram of the Hypercycle router for a graph of dimension r is shown in Fig. 3.3. The router is comprised of

1. r Next Port generator (NPG) modules corresponding to the maximum r dimensions of the network
2. The Port Validator (PV)
3. The Port Selector (PS).

A destination address generated by the host node or as a request of continuation of a partial path is presented to the router by the controller. The router computes the set of forwarding ports to the nearest nodes from the current node with the help of the Next Port Generator (NPG) based on information such as current address, destination address, connectivity and population. The Port Validator determines which of the set of forwarding ports computed by the NPGs are free. The Port Selector randomly chooses one of the validated ports to which the originating port will be connected through the switching network. We shall delve into the details of each of the modules in the subsequent sections.

3.6.1 Next Port Generator

The Next Port Generator implements the Hypercycle routing equations to determine the next possible port to route the messages. It can be seen from Fig. 3.3 that the r modules of the NPG work in parallel to generate a set of

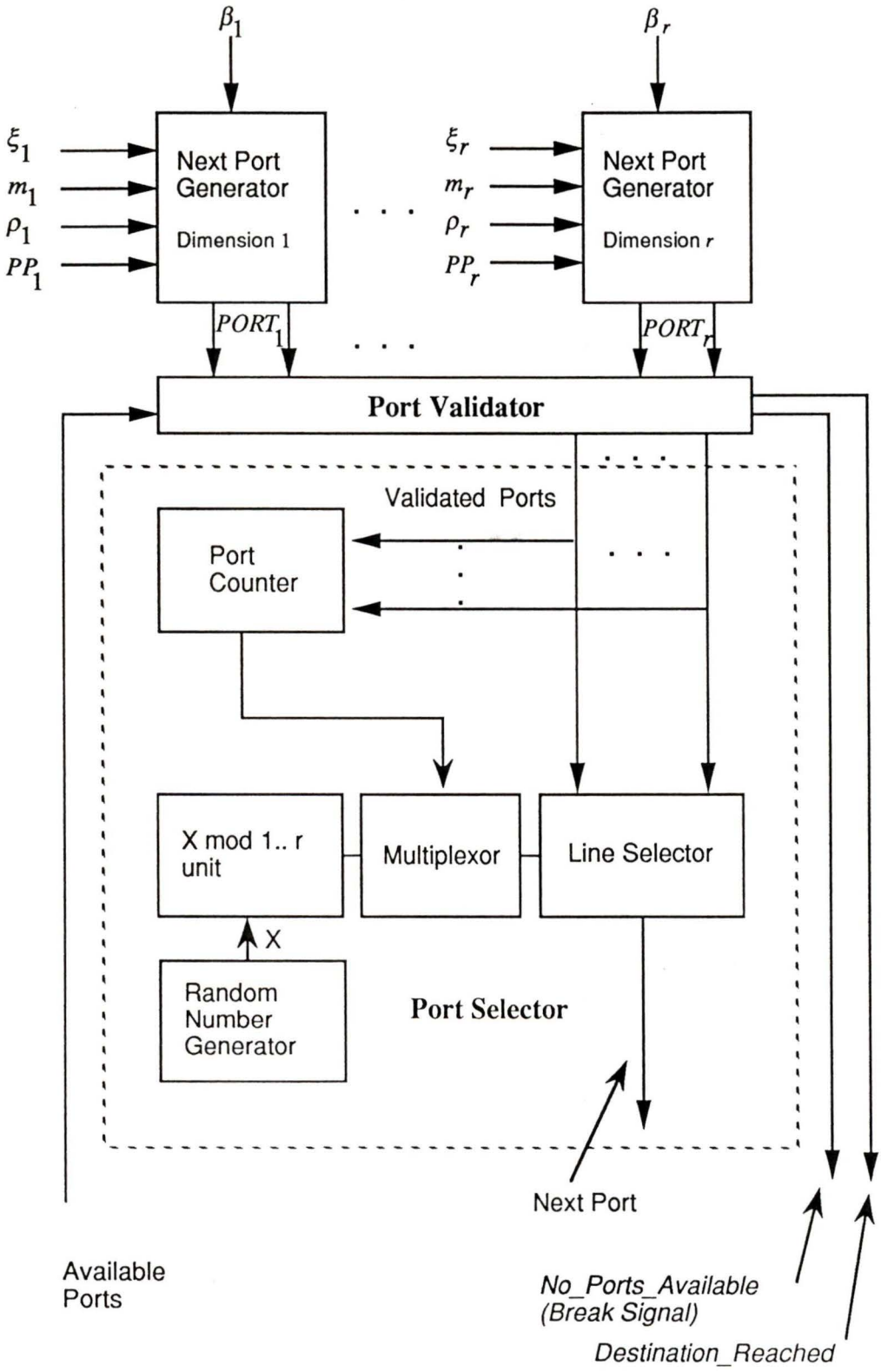


Figure 3.3: Architecture of the Hypercycle Router

at most r possible ports. The algorithm used by the NPG for routing in the i^{th} dimension given a source address ξ_i, destination address β_i, population m_i, connectivity ρ_i is as follows.

1. Compute the minimum of $(\xi_i - \beta_i) \bmod m_i$ and $(\beta_i - \xi_i) \bmod m_i$. This determines the direction of movement either clockwise or counterclockwise towards the destination node.

2. Compute the minimum of ρ_i and the result obtained from step (1). This simply means that the largest possible step, namely ρ_i is taken if the destination is beyond the reach of the source node. Alternately if the destination node is reachable from the source node, then the corresponding port determined from step (1) is selected.

3. Add the offset for dimension i to the logical port determined from step(2) to obtain the actual physical port for dimension i. The offset is computed by adding the connectivities of the previous dimensions. The expression for the offset (PP_i) for dimension i is given by

$$PP_i = \sum_{t=1}^{i-1} e_t \quad \text{for } 1 \leq i \leq r \tag{3.1}$$

where

$$e_t = \begin{cases} 2\rho_t - 1 & \text{if } \rho_t = \frac{m_t}{2} \\ 2\rho_t & \text{otherwise} \end{cases} \tag{3.2}$$

4. If $\xi_i = \beta_i$ then assert the destination reached signal.

As explained in [45], the following convention is used in establishing the correspondence between the communication links and the logical/physical ports.

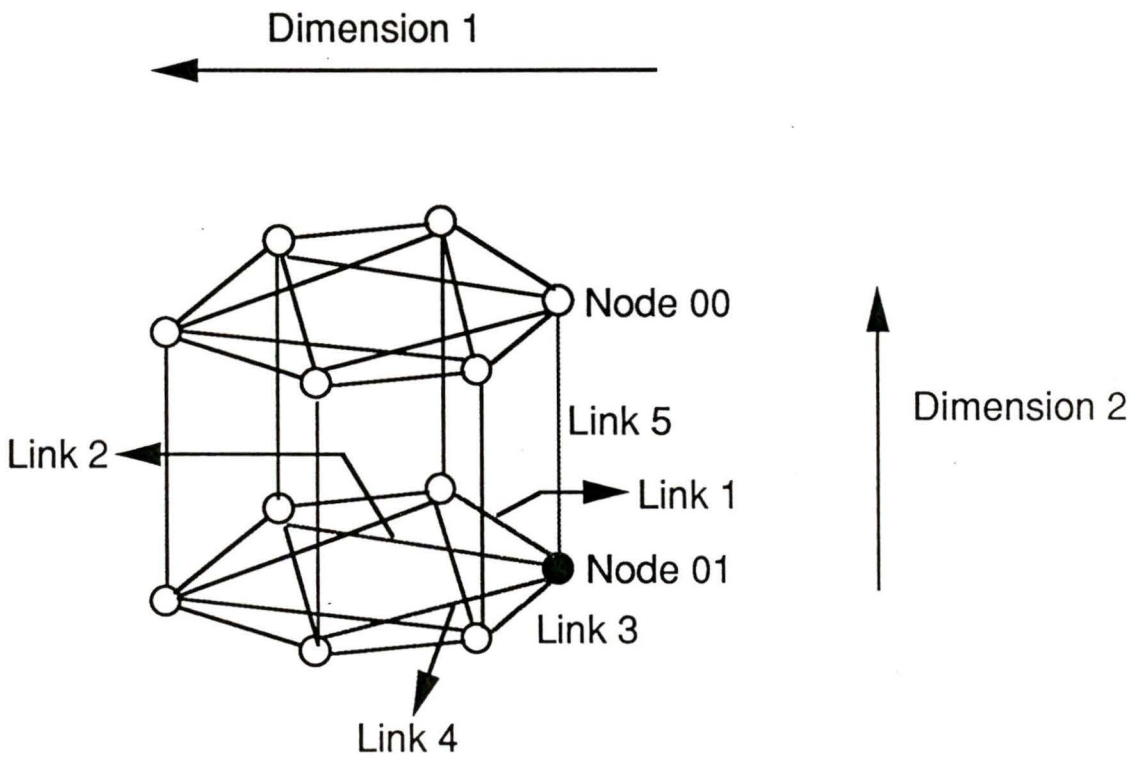


Figure 3.4: Physical Port Numbering in Hypercycle $\mathcal{G}_{62}^{21}$

In each dimension, there are $2\rho_i$ ($2\rho_i - 1$ if $\rho_i = m_i/2$) edges. We number the logical ports from 1 to $2\rho_i$ ($2\rho_i - 1$) such that in the counterclockwise direction the minimum step will correspond to logical port 1 while the maximum step to logical port ρ_i. For the clockwise direction, the minimum step will correspond to logical port $\rho_i + 1$ and the maximum step to logical port $2\rho_i$ ($2\rho_i - 1$). The logical port is converted to a physical port by accounting for all the ports of the previous dimensions and adding this as an offset to the computed logical port to obtain the physical port for dimension i.

As an illustration, consider the Hypercycle $\mathcal{G}_{62}^{21}$ shown in Fig. 3.4. The physical port numbering for node 01 is given by links 1,2,3,4 in dimension 1 and link 5 in dimension 2 as per the above discussion. Consider a message from source node 01 (expressed in MRN) bound for destination node 00, the two nodes being separated by a distance 0 in dimension 1 and distance 1 in dimension 2. The algorithm computes the next port address by adding the offset of dimension 1, namely 4, to logical port 1 in dimension 2 and hence physical port 5 constitutes the next port address.

3.6.2 Port Validator

The Port Validator acts as a preprocessor checking the availability of the computed next ports fed from the NPGs by comparing them with the available ports. The other functions of the Port Validator are

1. to pass the validated ports to the port selector circuit;
2. to inform the controller when the final destination has been reached;
3. to generate a path-blocked (break) signal when there are no available ports for routing.

The fact that the final destination has been reached or not can be determined from the destination reached (DTR) signals of the NPGs. When the DTR signal is asserted in all the r-dimensions it can be construed that the intended destination has indeed been reached.

When the message to be routed cannot proceed further because of the non-availability of free ports, a break signal is generated which is used by the controller in its implementation of the routing strategy. For the BTOR scheme, the break signal causes the already established partial path to collapse. Special detectors at the cross-bar switch detect the break signal and dissolve the corresponding connection.

The basic hardware needed for the Port Validator constitutes an array of AND gates. By ANDing the available ports with the next ports, we can determine which of the next ports are available in reality. It is assumed here that an output of '1' indicates that the port is available and '0' that it is unavailable. Thus if the output of all the AND gates are zeroes, a *break* is encountered. If the ANDing of all the DTR signals from the r NPGs indicate a '1', then the final destination has been reached.

3.6.3 Port Selector

The Port Selector circuit accepts the validated ports as input from the Port Validator. The Port Selector proceeds as follows:

1. counts the number of validated ports;

2. generates a random number between 1 and this count;

3. selects the port corresponding to the random number.

To accomplish these tasks, the major building blocks needed are adders, random number generators, modulo-arithmetic units, line selectors, decoders

and multiplexors. Since the number of ports to be selected varies from 1 to r, a modulo arithmetic unit that is capable of computing X mod k for $k = 1 \ldots r$ where X is a random number needs to be designed. As shown in Fig. 3.3, the output of the RNG is given to the modulo unit and this is multiplexed depending on the count of the validated ports. The resulting output from the multiplexor is transported to a line selector circuit whose output constitutes the next port address of the router.

3.7 Conclusion

In this chapter, the BTOR scheme for the Hypercycle networks was introduced and the system architecture of the router was formulated. The functions of the sub-blocks of the Hypercycle router, namely the Next Port Generator, Port Validator and Port Selector were analyzed. In the following chapters, we discuss in detail the VLSI implementation of these sub-components and the system integration of the router.

Chapter 4

VLSI Design Considerations

4.1 Introduction

While dealing with the implementation issues of the router, we need to analyze the pros and cons of various techniques that are available for the purpose. One possible implementation is to code the algorithm in software using a microcontroller. This is relatively simple, cheap and flexible but the obvious degradation in the throughput of the router caused by slow instruction cycles makes it unsuitable. Another method is to use ROM look-up tables. The very fact that we need to have separate tables for each node and configuration besides the additional control circuitry makes it unrealistic from the point of view of cost, performance and ease of reconfiguration especially for large networks.

The Hypercycle routing equations can be analytically computed in that the basic operations involved are addition, subtraction, modulus evaluation and magnitude comparison which are amenable to implementation using discrete components, ASICS, gate arrays, standard cell VLSI or full custom VLSI. The major drawbacks of implementing the circuit in VLSI are the limits imposed by the available area, pin count, power dissipation, low yield and the excessive turn-around-time. However the merits of VLSI include high reliability, increased

performance/cost ratio, low power consumption (depending on the choice of technology such as CMOS), high component density, high speed, reduced volume & weight, standard packaging and secure design. High performance designs can be produced by exploiting the latest *state-of-the-art* VLSI technologies.

4.2 VLSI system design

The micro-electronics era has ushered in a new frontier in the field of computing and this has been largely made possible by the availability of low cost, high density, high speed VLSI circuits, superior technologies and sophisticated CAD tools. VLSI architecture should benefit from modularity, regularity, expandability, local communication, massive parallelism and minimized I/O connections. The key issues to be addressed when designing VLSI systems are:

1. Algorithm used
2. Simple and regular data, control paths
3. Localized interconnections
4. Regular, modular structures
5. Trade-off between I/O and computation speed
6. Concurrency
7. Synchronous/ asynchronous operation
8. Reconfigurability and fault-tolerance
9. Chip partitioning

10. Role of CAD tools.

Hence, before starting a VLSI development project, it is incumbent to validate the system design aspects and analyze it from a practical point of view. The development process undergoes four phases, namely

1. Behavioural phase
2. Structural phase
3. Physical design phase
4. Test phase.

Each of the above phases has a dramatic effect on performance, portability and turn-around-time of the chip.

The behavioural phase is essentially an abstract description of the function of the device. The initial assumptions and specifications governing the chip are studied to draw up an architecture of the system. More often than not, the specifications are incomplete, sometimes impractical and need modifications during the successive phases of the design cycle.

The structural phase is the decomposition of the specifications into sub-blocks typically through a top-down design methodology. The functions of each of the modules of the system are defined. A gate level layout of the modules are obtained and the performance of the system is validated through simulation.

In the physical design phase, the layout of the design is realized in a given technology. During this phase, extensive use of CAD tools and device physics are made use of in designing the chip.

The final phase is the testing of the fabricated chips which includes AC/DC characterization and design validation.

The advent of VLSI has brought about a great increase in circuit complexity. To cope with this enormous complexity, highly structured and modular design approaches need to be adopted. The top down approach gives a perspective view of the structure and behaviour of the system while a bottom-up approach is used in the physical design and layout stage. The design cycle should be highly interactive with feedback from each stage necessary for the realization of the initial objective.

In short, the entire design cycle is long and complex and involves several trade-offs on the part of the designer including an analysis of technologies, methodologies, approaches and strategies.

4.2.1 Design Methodology

The layout design methodology has a direct bearing on the performance, cost, turn-around-time and yield of the chip. The following are some of the approaches used in creating the layout of the intended chip.

4.2.2 Full custom approach

The full custom VLSI design is the most expensive and time consuming layout methodology but a very powerful method for circuit implementation. The given circuit is broken down into several modules each of which is handcrafted by paying attention to transistor sizing and interconnection. High throughput is achieved but at the expense of design time. The custom approach is usually used when high speed and density are required.

4.2.3 Gate Array

Gate arrays have brought about a major revolution in VLSI in that they substantially reduce the cost and design time incurred in the full custom approach but with a degradation in performance. Gate arrays consist of pre-fabricated gates without the interconnect metallization. The general layout of the gate array can be converted to an ASIC by depositing the top metal layer through appropriate masks. Thus turn-around-time is short but the freedom of custom design is lost due to the wastage of unused gates.

The Logic Cell Arrays (LCA) by Xilinx and the Field Programmable Gate Arrays permit in-house programmability for reduced turn-around-time.

4.2.4 Standard Cell design

The standard cell based approach is also called the semi-custom approach. It strikes a compromise between full custom and gate array approaches. The design time approaches that of gate arrays while the performance reaches that of the full custom approach. A standard cell library offers the designer a number of logic gate standard cells that are sufficient to implement any digital system. The cells are handcrafted, designed, simulated, laid out, tested and characterized. All the cells have equal heights and the cell boundaries can be joined without conflict. The cells can be manually or auto-routed.

Timing verification of a standard-cell design is usually carried out using a gate level timing simulator such as SILOS [81]. The wastage of gates in gate arrays can be avoided in the standard cell approach since only the cells which are needed in the design are utilized. Complete mask generation as in the full custom approach is required and this renders the system more expensive than gate arrays. The performance of the standard cell approach is better than gate

arrays but not as good as the full custom approach.

4.3 Implementation Considerations

The following are some of the important criteria in designing the architecture of the chip.

1. The size of the die, which is determined by the area occupied by the circuitry
2. The number of I/O terminals
3. The power requirements
4. The Turn-Around-Time (TAT)
5. The performance/cost ratio
6. CAD tools
7. Testability and verification
8. Technology
9. System clocks and synchronization.

The chip area is estimated from the logic design specifications of the chip by separately estimating the area occupied by the logic gates and the interconnection. The maximum number of I/O terminals is determined from the number of bonding pads that can be placed on the periphery of the chip. It is important to determine the I/O configuration as it affects the performance of the chip. If the design is I/O bound, then the design should be either partitioned into

separate chips or multiplexing of pins should be used. This factor affects the performance of the chip. The turn-around-time is dependent on the complexity of the design, the available resources (CAD tools) and the layout methodology.

4.3.1 CAD tools

The increased complexity of current ICs has been greatly mitigated by the availability of sophisticated CAD tools which relieve the designer from the drudgery and error-prone tasks of the design process. These tools are indispensable for IC design and they have a direct impact on productivity, increased reliability and decreased turn-around-time. A well defined and systematic CAD tool for design, simulation, layout generation and verification is indispensable. The CAD tool used in this thesis is the Cadence package [80] whose features will be summarized in the later sections.

4.3.2 Testability & Verification

VLSI has brought with it the problem of testing and verification of the implemented chips. The cost of testing chips is such that without a structured approach, it would be impossible to verify and validate the correctness of the chip. Therefore special emphasis should be given to testability. The economies of test strategies balance the costs of test vector development against the benefits of earlier fault detection. Methods of design for test include ad-hoc [61], scan-path [62] and built-in-self test [63]. In each method, there is a tradeoff between increased testability and increased silicon area and pin count. Software for test vector development includes nodal activity checkers, testability analyzers and automatic test pattern generators.

4.3.3 Technology

The choice of technology is one of the most important and crucial factors in designing VLSI systems as it directly dictates the architecture of the chip in terms of speed, performance, cost, design time and a host of other related issues. Despite the fact that semiconductor technologies like bipolar and MOS have been superseded by newly emerging technologies like BiCMOS and GaAs. the latter technologies being still in their infancy. Bipolar technologies like TTL, ECL are much faster than MOS but on the other hand they require more power and area.

MOS circuits are based on pMOS, nMOS and CMOS technologies [64]-[70]. Despite the inherent simplicity in the fabrication of pMOS circuits, nMOS and CMOS are much superior to pMOS in terms of performance and power consumption. pMOS circuits are slower than nMOS circuits since the mobility of electrons is larger than that of holes ($\mu_n/\mu_p \approx 2.5$). CMOS circuits which are comprised of pMOS and nMOS transistors are slower than nMOS since the input capacitance is larger than that of nMOS. Further more, CMOS circuits require more transistors per gate compared to nMOS. Stated alternatively, the degree of integration for a given area of silicon real-estate is greater in nMOS than CMOS.

Yet CMOS technology is perhaps one of the most popular, commercially exploited and widely used contemporary technologies for IC fabrication because of lower power dissipation and good scalability compared to nMOS. State-of-the-art CMOS submicron processes make switching speeds of CMOS comparable to that of nMOS. The following are some of the important characteristics of CMOS in contrast to nMOS and pMOS.

4.3.3.1 Transfer Characteristics

The outputs of nMOS circuits are *ratioed* and the rise and fall times depend on the ratio of the geometry of the driver and load transistors. In contrast, CMOS is a ratioless logic and the output is independent of the size of the driver and load.

4.3.3.2 Power dissipation

In nMOS, static power dissipation is present since the load transistor always provides a conducting path from power to ground regardless of the state of the input. In CMOS however, the power dissipation is dynamic since under static conditions the output is connected to either V_{dd} or V_{ss} neglecting the leakage currents.

The expression for the dynamic power dissipation of a CMOS inverter is given by

$$P_{dyn} = C_l V_{dd}^2 f \tag{4.1}$$

where C_l is the load capacitance and f is the frequency of operation. Thus it can be seen that under high operating frequency, the power dissipation of CMOS is large and it approaches the power dissipation of nMOS.

4.3.3.3 Scaling

CMOS benefits from scaling in feature size while nMOS, for power dissipation reasons, is more difficult to scale down.

4.3.3.4 Noise Immunity

The noise immunity in CMOS is excellent since the '1' and '0' logic levels are passed undegraded by the pMOS and nMOS transistors respectively. During

transition, the output makes a full swing from V_{dd} to V_{ss} or vice versa and this paves the way for large noise margins and hence the circuit is more secure and reliable from voltage spikes. For a symmetric CMOS inverter with $\beta_n = \beta_p$ and $V_{t_n} = |V_{t_p}| = V_T$, it can be shown that

$$V_{OH} = V_{dd} \tag{4.2}$$

$$V_{OL} = V_{ss} \tag{4.3}$$

$$V_{IH} = \frac{1}{4}\left[\frac{5}{2}V_{dd} - V_T\right] \tag{4.4}$$

$$V_{IL} = \frac{1}{4}\left[V_T + \frac{3}{2}V_{dd}\right] \tag{4.5}$$

and $V_{IH} + V_{IL} = V_{OH} + V_{OL} = V_{dd}$. Hence the high and low noise margins are given by

$$NM_H = V_{OH} - V_{IH} \tag{4.6}$$

$$= V_{dd} - V_{IH} \tag{4.7}$$

$$= V_{IL} \tag{4.8}$$

$$NM_L = V_{IL} - V_{OL} \tag{4.9}$$

$$= V_{IL} \tag{4.10}$$

Thus noise margin high is equal to noise margin low.

The present day CMOS technologies with sophisticated submicron processes have already eclipsed the subnanosecond barrier. This advancement in CMOS technology has been overwhelming in that it has become a major competitor to ECL and TTL technologies in terms of speed, power and degree of integration. All these striking factors have culminated in choosing CMOS as the vehicle of implementation.

4.4 Device Issues

In this section, we consider the inherent problems in VLSI such as the optimal design of clock drivers, power bus and I/O pad design.

4.4.1 Clock drivers

The inherent problem of clock distribution is to drive the high fanout of a clock signal without introducing unacceptable skew that will otherwise cause synchronization failures. Clock skew is caused by

1. variations in gate delays
2. variations in loading
3. variations in wire length.

In order to increase the driving capacity of the source, drivers have to be placed along the distribution wires. In custom layout, the degree of clock skew can be reduced since the designer can alter/adjust the gate delays, clock path length in order to equalize the driver load. However this is very expensive and impractical.

A common scheme of clock distribution called *geometric buffering* is shown in Fig. 4.1. This approach is due to Mead and Conway [76] and the basic idea is to use a series of N cascaded inverters (drivers) of increasing widths to drive a load of capacitance C_L. It has been shown in [77] that for minimum delay, the ratio between the sizes of the adjacent drivers should approximately be e. The problem with this method is the large area consumed by drivers of increasing size and it is not feasible to realize using the CMC's standard cell libraries.

An elegant approach suggested by Dejan [78] for constructing the clock drivers is made use of in designing the driver stages. The basic idea behind

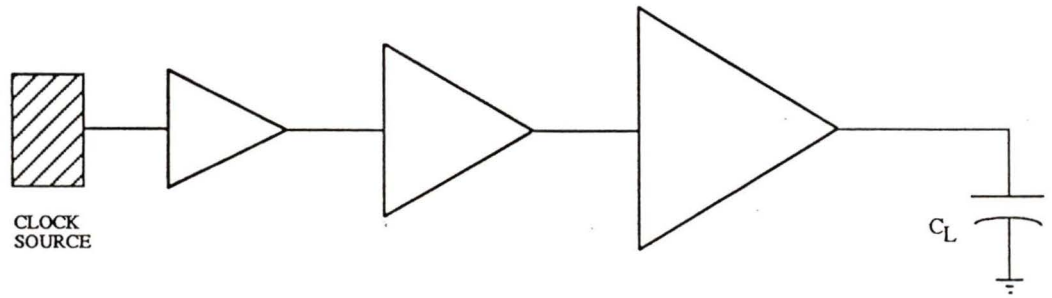


Figure 4.1: Clock distribution of Mead and Conway

Dejan's method is to use identical drivers distributed in a tree-like fashion as shown in Fig. 4.2 and equalize the load thereby reducing the clock skew. This approach is termed as *tree buffering*. It has been proven in [78] that these driver stages can be designed as a multilevel scheme achieving minimum delay of clock distribution and area in addition to utilizing drivers of the same capacity. In our design, we have adopted this method of hierarchical distribution of drivers.

4.4.2 Power Bus

The main objective in designing the power supplies and the power conductor widths is to ensure that power bus voltage noise spikes are small enough so that they do not significantly affect the operation of the chip. In practice, this target is difficult to achieve because of the numerous current spikes and surges that occur both on and off the chip. To circumvent the problem of *metal migration* which is caused by excessive current densities, increased temperature

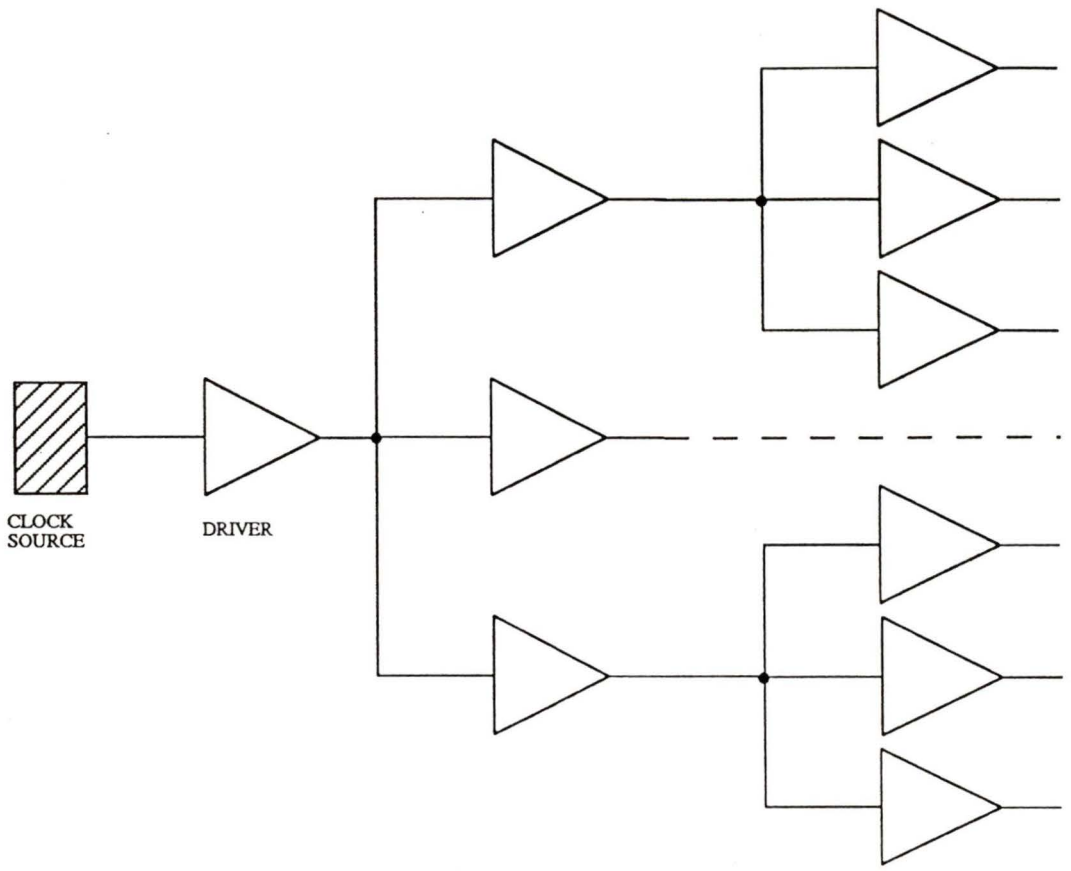


Figure 4.2: Tree buffering of equal sized drivers

and crystal defects, the power and ground lines should be designed to yield acceptable current densities as characterized by their width. The lower bound on the power conductor widths for the CMOS4S technology can be determined as follows based on the procedure given in [74, 75].

The limiting value of current density for 1 μm wide aluminium wire is approximately 1 mA /μm. In this analysis, we have assumed that the core consists of about 6000 cells. The power lines feed the different regions of the core in a honey-comb fashion. Assuming that each branch of the power line supplies current to one-eight of core (about 750 cells) we can derive the power conductor

width based on the following assumptions.

- The cells that switch states on any clock cycle are uniformly distributed.
- 20 % of the cells switch per cycle as per the assumption made in [75].
- Cells drive a load of approximately 2.5 nand2 inputs giving a load capacitance C_L of 200fF.
- Rise and fall times are such that $t = t_r = t_f = 0.7ns$
- Duty cycle is less than 10 %.

The average current is given by

$$I_{avg} = 0.1 \times 0.2 \times C_L \frac{V_{dd}}{T} \times 750 \tag{4.11}$$

$$= 0.1 \times 0.2 \times 200fF \times \frac{5V}{0.7ns} \times 750 \tag{4.12}$$

$$= 21.43mA \tag{4.13}$$

Hence the power conductor width is set to 22 DSM. In practice, it may often happen that the power supply conductor widths cannot be increased beyond a certain point for reasons such as increased channel area and routing problems. This can be offset by adding extra V_{dd} and V_{ss} pads to redistribute the current. Furthermore, the general guideline is to use an additional power and ground pad for every 3000 logic gates. Under these assumptions, the dynamic power consumption of the chip is estimated to be

$$P_{dyn} = 21.23mA \times 5V \times 8 \tag{4.14}$$

$$= 850mW \tag{4.15}$$

For our CMOS4S designs which incorporated 6000 gates, 3 pairs of power pads were used.

4.4.3 Output Pads

I/O switching is the principal source of noise in the chip frame. A switching output pad causes large current surges on the supply buses which cause noise and metal migration problems. In general, these disturbances increase with the size of the load and drive capability of the output pads and therefore proper care has to be taken in deciding whether to use high or low current pads. High current pads have shorter transition times and are capable of driving a large fan out but they also create voltage spikes and introduce electro-migration problems. Low current pads on the other hand cause less noise but they are slower than the high current pads. The rule of thumb is to use high drive pads only when the situation such as high speed and large fan out warrant their use. Low current pads have been used while designing the chips in this thesis.

4.4.4 Cell Libraries

The CMOS4S library, *sc_v1.1*, contains 4 types of output pads, namely, *outpad4ma, outpad8ma, bipad4ma and bipad8ma.* The use of separate power nets for the core and the I/O ring provides isolation from noise and voltage spikes. To further reduce the noise problems, the use of multiple power and ground pads for the core and the I/O ring has been recommended [75]. Furthermore, the widths of the power and ground buses of the I/O pad circuitry have to be wider compared to the core to minimize any ringing effects. For the CMOS3DLM and CMOS4S technologies, the power conductor widths of the I/O pads were set to 150 DSM and 75 DSM respectively. The power and ground rails are routed as interdigitated structures [69] and this is shown in Fig. 4.3. The V_{dd} and ground rails of the I/O frame constitute a double ring and are separated from the power rails of the core to minimize voltage spikes.

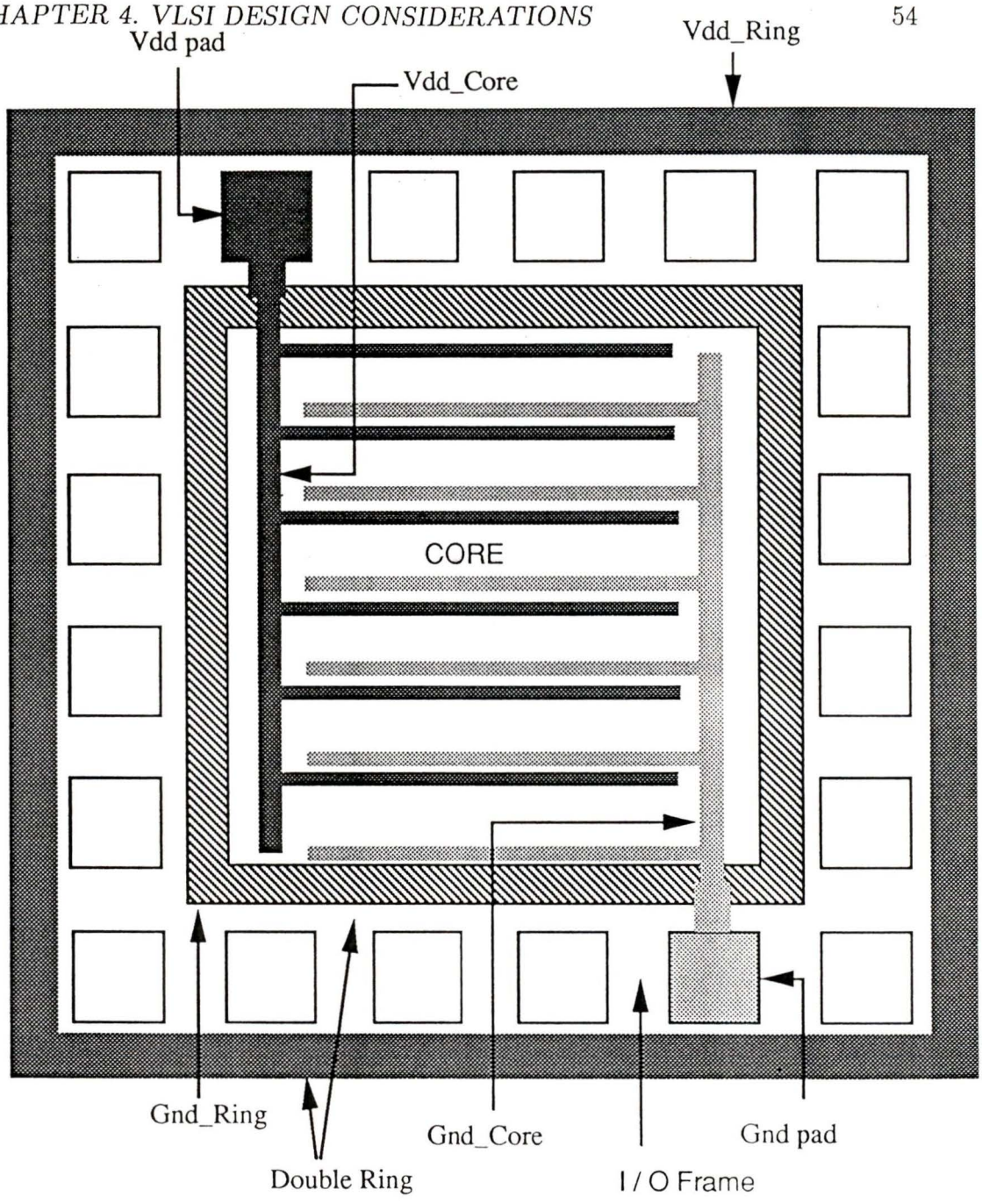


Figure 4.3: Power Bus Routing in VLSI chip

4.5 IC Fabrication at CMC

The Canadian Microelectronics Corporation (CMC) makes available to its member Universities two CMOS processes namely,

1. The 3 μ CMOS3DLM process
2. The 1.2 μ CMOS4S process.

in collaboration with Northern Telecom Electronics (NTE). CMC has several multiproject chip (MPC) runs (typically 4 per year).

The design rules for the physical layouts of the chips conform to the CMC DRC Dracula [79] routine and a design which passes these rules is deemed to have passed the NTE CMOS3DLM and CMOS4S processes. The chip designs which have evolved as part of this research have been fabricated by CMC. The standard cell approach has been used in designing the chips for the benefits stated earlier. The CMC compatible CMOS3DLM and CMOS4S libraries [72, 73] have been extensively used in this project. The CMOS3DLM and CMOS4S library consists of 23 standard cells with 5 I/O pads. A standard cell inverter in the CMOS3DLM and CMOS4S technology has a propagation delay of 1.5ns and 140ps respectively while the area is 20 $\times$ 20 DSM and 10 $\times$ 10 DSM respectively. All cells have been simulated using SPICE [71], tested and checked with the help of CMC's Dracula design rule checker for the respective technologies. The AC, DC characteristics of these cells are given in detail in [72, 74].

4.6 Implementation of the Router

As the ultimate aim of this project is to implement the router in VLSI, it is considered a worthy proposition to make the various modules of the router

as separate chips and to study and validate their performance before finally integrating them. The important modules of the router that are of sufficient complexity include the Next Port Generator, Random Number Generator and Modulo-extractor which were designed in CMOS3DLM technology while the entire Hypercycle router was implemented in CMOS4S.

Such a methodology was adopted in order to verify the individual prototypes and make any improvements in the routing engine depending on their performance. The design of these chips will be the focus of discussion in the subsequent chapters.

4.6.1 Implementation of the NPG

The NPG has been dealt with in detail in [45] and a prototype of the same was implemented in discrete logic using TTL SN74LS series chips. The circuit was found to be working satisfactorily and it had a maximum propagation delay of 175ns. The NPG discussed in [45] was designed in the 3 μ CMOS3DLM technology using the CMC compatible standard cell library with the aid of the EDGE or Cadence VLSI design tool [80]. The Cadence package is a highly versatile CAD tool and provides a completely integrated set of VLSI design and verification tools in a menu-driven, graphical environment. It permits advanced schematic capture and contains sophisticated simulators for logic and circuit simulation including SILOS, SPICE and HSPICE. It also includes facilities for creating custom layout as well as the use of standard cell libraries from the Cadence database.

The automatic Place & Route program makes the placement and routing of the design relatively simple since it has automated routines for placement, clustering, channel generation, routing and and compaction. It includes provisions

Simulation Data for NPG						
Input					Output	
RHO	XI	BETA	M	PP	DTR	PORT
1	1	2	3	0	0	2
1	0	3	4	0	0	1
2	1	3	4	0	0	2
2	0	3	4	0	0	1
2	4	1	5	0	0	4

Table 4.1: Logic simulation data for the Next Port Generator

for DRC, ERC and circuit extraction. The final layout produced from Cadence is converted to a CIF file which is used for mask generation during fabrication. The circuit considered here for implementation is a 15-node, single dimension NPG. The schematic capture of the above circuit was done using the Cadence design tool.

The VLSI layout of the NPG is shown in Fig. 4.4. The chip consists of a total of 26 pins with 19 input, 5 output and 2 power pins. The implementation details of the NPG are summarized in Tables 4.2 and 4.3. The schematic diagrams and pin descriptions of the Next Port Generator are given in Appendix A. A snapshot of the simulation results of the NPG is presented in Fig. 4.5 for the vectors given in Table 4.1. To illustrate the waveforms given in Fig. 4.5, let us consider the case when RHO=1, XI=0, BETA=3, M=4 and PP=0 (second set of vectors in Table 4.1. These vectors are applied to the NPG at time $T = 200$ ns. From the waveforms, it can be seen that the output PORT settles to 1 at approximately $T = 320$ ns while DTR settles to 0 at $T = 240$ ns.

NPG	
Technology	CMOS3DLM
Area of the chip	3105.90 x 3105.90 μ^2m
Area of the core	1905.90 x 1905.90 μ^2m
No. of pins	26
No. of standard cells	546
No. of transistors	2014
Propagation delay (from simulation)	125ns
Propagation delay (from testing)	127-137ns
Area*Time	4.504e08 ns μ^2m
Packaging	40-pin DIP

Table 4.2: Statistics of NPG

NPG	
Gate	Count
Inverters	189
Buffers	54
Nand2	196
Nand3	20
Nand4	16
Nor2	36
Nor3	18
Nor4	17

Table 4.3: Gate level statistics of the implemented NPG

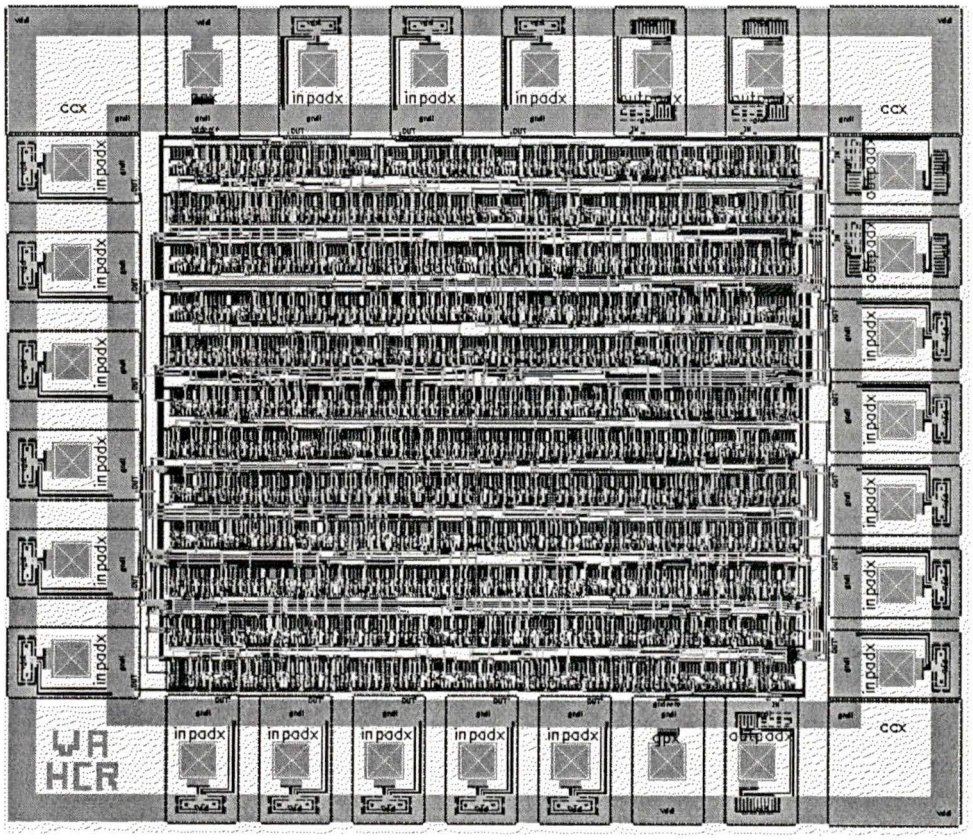
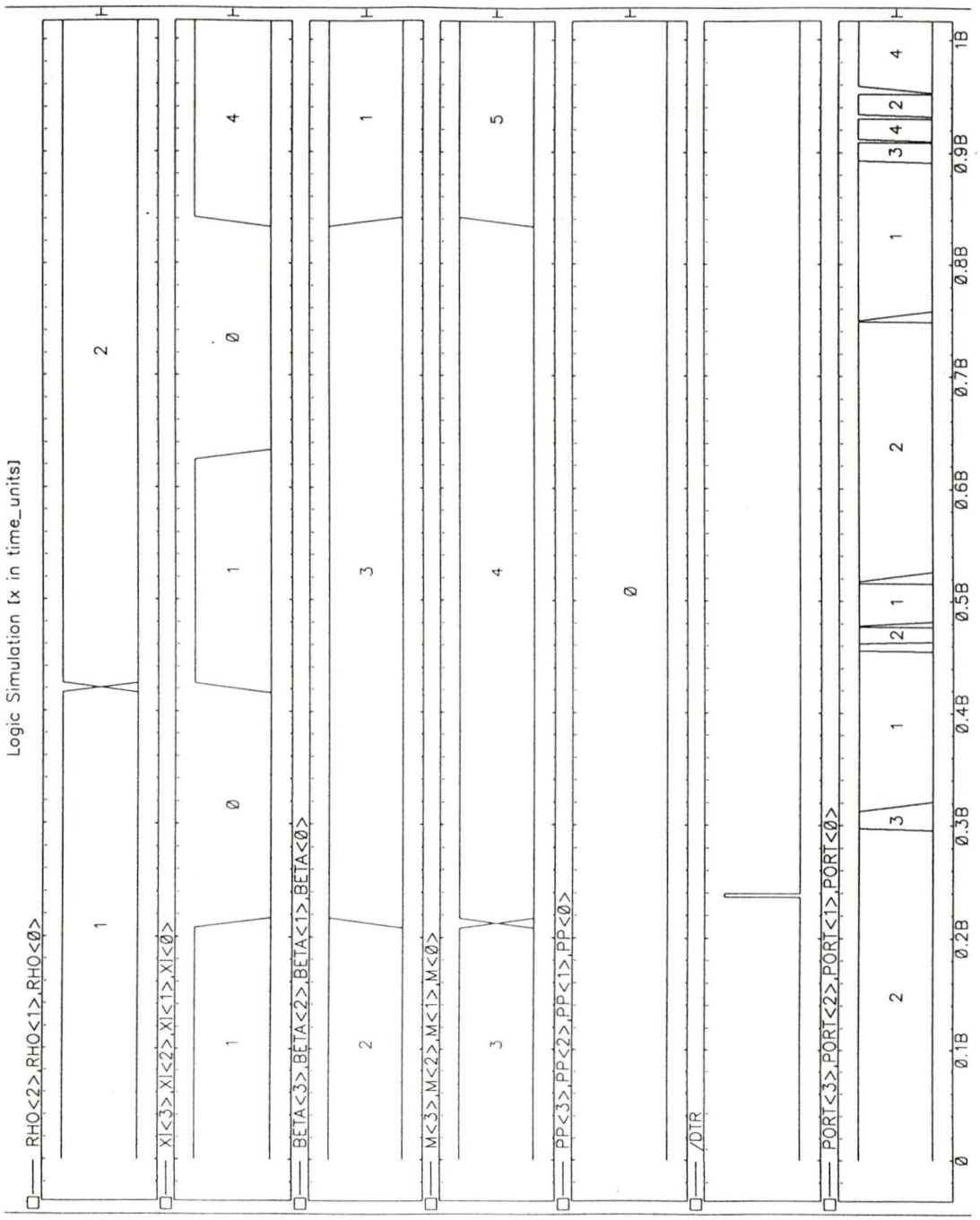


Figure 4.4: VLSI layout of the implemented NPG (3 μ CMOS3DLM)

4.7 Conclusion

The technological impact of VLSI has made it feasible to realize cost effective, high performance VLSI communication chips. In this chapter the fundamental issues concerning the design of integrated circuits were discussed. We considered the implementation of the Next Port Generator in CMOS3DLM technology. In the subsequent chapters, we address the design of other modules of the Hypercycle router from the implementation point of view.



$B = 1\mu s$

Figure 4.5: SILOS simulation of NPG (3 μ CMOS3DLM)

Chapter 5

Design of the Modulo-Extractor

5.1 Introduction

As explained in the previous chapter, a random number generator coupled with a modulo-extractor is needed to select the appropriate port in the Port Selector module of the router. Given the fact that a n-bit RNG produces a number X during each decision cycle, we need to compute $X \bmod i$, $i = 1 \ldots r$. The problem is then reduced to designing circuits that can perform random number generation and modulo extraction. This chapter focuses exclusively on the design of a modulo-extractor for computing the function $X \bmod m$ for specific values of m.

The methodology used in the design of the modulo extractor is founded on the principles of *Residue Arithmetic* [48]. In the following sections, the theory behind the use of residue arithmetic in the modulo-extractor and its formulation are elucidated. Expressions for the area-time complexity of the proposed model are derived. Finally the VLSI implementation of the modulo extractor is discussed in the end. We first introduce the basic definitions and terminology associated with residue arithmetic.

5.2 Residue Arithmetic

Let X and m be any two integers with $m \geq 0$. Then we can write

$$X = mq + r \tag{5.1}$$

where q is the quotient and r is the remainder such that $0 \leq r < m$. The remainder r, called the *residue* of X mod m is denoted as

$$r = \langle X \rangle_m \tag{5.2}$$

The value of $\langle X \rangle_m$ is obtained from the remainder of the integer division X/m. However division is very expensive and does not lend itself to easy implementation.

To compute $\langle X \rangle_m$ without having recourse to division or through software, we use the principles of residue number theory. Residue Number systems are characterized by a high degree of parallelism, regularity and modularity which can be exploited in VLSI to build high speed digital circuits [85]-[88]. Residue number systems lend themselves to several applications in signal processing (digital filters, Discrete/Fast Fourier Transform, convolution), in the ALU of digital computers, in high speed arithmetic units such as fast adders, multipliers, random number generators [89]-[91]. The modulo extractor discussed here is capable of computing the modulus of the forms 2^k, $2^k - 1$, $2^k + 1$. The modulus of other composite numbers which can be expressed as a product of prime factors can be determined through the application of the *Chinese Remainder Theorem* (CRT).

Let N be a number system defined in radix β. Then any number $X \epsilon N$ can be represented as an n-digit tuple $x_{n-1}, x_{n-2}, \ldots, x_1, x_0$ such that

$$\begin{aligned} X &= x_0 + \beta x_1 + \beta^2 x_2 + \ldots + \beta^{n-1} x_{n-1} \\ &= \sum_{i=0}^{n-1} \beta^i x_i \end{aligned} \tag{5.3}$$

5.2.1 Calculation of modulus β^k

For $k < n$, the number X is written as

$$\begin{aligned} X &= x_0 + \beta x_1 + \beta^2 x_2 + \ldots + \beta^k x_k + \ldots + \beta^{n-1} x_{n-1} \\ &= \sum_{i=0}^{k-1} \beta^i x_i + \beta^k [x_k + \beta x_{k+1} + \ldots + \beta^{n-1-k} x_{n-1}] \end{aligned} \tag{5.4}$$

Thus

$$\begin{aligned} \langle X \rangle_{\beta^k} &= \langle \sum_{i=0}^{k-1} \beta^i x_i + \beta^k [x_k + \beta x_{k+1} + \ldots \beta^{n-1-k} x_{n-1}] \rangle_{\beta^k} \\ &= \langle \sum_{i=0}^{k-1} \beta^i x_i \rangle_{\beta^k} \\ &= \sum_{i=0}^{k-1} \beta^i x_i \end{aligned} \tag{5.5}$$

Therefore the number represented by the least significant k digits of the representation of X gives the modulus $\langle X \rangle_{\beta^k}$.

5.2.2 Calculation of modulus $\beta^k - 1$

Before proceeding further, we state the following relations which are important in this analysis.

$$\begin{aligned} \langle \beta^k \rangle_{\beta^k-1} &= \langle \beta^k - 1 + 1 \rangle_{\beta^k-1} \\ &= \langle \langle \beta^k - 1 \rangle_{\beta^k-1} + \langle 1 \rangle_{\beta^k-1} \rangle_{\beta^k-1} \\ &= 1 \end{aligned} \tag{5.6}$$

Similarly, we have

$$\begin{aligned} \langle \beta^{nk} \rangle_{\beta^k-1} &= \langle [\beta^k]^n \rangle_{\beta^k-1} \\ &= \langle [\langle \beta^k \rangle_{\beta^k-1}]^n \rangle_{\beta^k-1} \\ &= \langle 1^n \rangle_{\beta^k-1} \\ &= 1 \end{aligned} \tag{5.7}$$

Let Y_0 denote the number represented by the k least significant digits. That is $Y_0 = \sum_{i=0}^{k-1} x_i \beta^i$. Y_1 is denoted by the next k significant digits given by $Y_1 = \sum_{i=0}^{k-1} x_{k+i} \beta^i$ and so on. In general $Y_l = \sum_{i=0}^{k-1} x_{lk+i} \beta^i$. Then

$$\begin{aligned} \langle X \rangle_{\beta^k-1} &= \langle Y_0 + \beta^k Y_1 + \beta^{2k} Y_2 + \ldots \beta^{(\psi_0-1)k} Y_{\psi_0-1} \rangle_{\beta^k-1} \\ &= \langle \langle Y_0 \rangle_{\beta^k-1} + \langle \beta^k Y_1 \rangle_{\beta^k-1} \ldots + \langle \beta^{(\psi_0-1)k} Y_{\psi_0-1} \rangle_{\beta^k-1} \rangle_{\beta^k-1} \\ &= \langle Y_0 + Y_1 + \ldots + Y_{\psi_0-1} \rangle_{\beta^k-1} \end{aligned} \tag{5.8}$$

where $\psi_0 = \lceil \frac{n}{k} \rceil$. The number ψ_0 represents the number of k-bit numbers to be added in the first step of the algorithm. After the sum $Z = \sum_{i=0}^{\psi_0-1} Y_i$ is determined in the first step, we apply the procedure recursively till the final result is less than β^k.

The number of digits needed to represent the sum Z is given by

$$\hat{Z} = \lceil \log_\beta Z \rceil \tag{5.9}$$

Since $Z < \lceil \frac{n}{k} \rceil \beta^k$, the upper bound of the digits needed to represent Z is given by

$$\hat{Z}_{ub} = \left\lceil \log_\beta \lceil \tfrac{n}{k} \rceil + k \right\rceil \tag{5.10}$$

The problem therefore reduces to evaluating the much simpler $\langle Z \rangle_{\beta^k - 1}$ due to a drastic reduction in its digit representation. The upper bound on the number of k-bit numbers (ψ_1) required for addition in the second step is given by

$$\psi_1 = \left\lceil \frac{\lceil \log_\beta \lceil \frac{n}{k} \rceil + k \rceil}{k} \right\rceil \tag{5.11}$$

The procedure is repeated till ψ_i is equal to $k+1$ which implies that $\langle X \rangle_{2^k - 1}$ is determined at that point. A decoding logic has to be incorporated in the final stage in order to detect the case $\langle \beta^k - 1 \rangle_{\beta^k - 1} = 0$.

A recurrence relation for the time complexity can be defined as follows.

$$T(n) = \begin{cases} T(\lceil \log \lceil \frac{n}{k} \rceil + k \rceil) + t(\lceil \frac{n}{k} \rceil) & \text{if } n > k+1 \\ 2t(2) + c & \text{if } n = k+1 \\ c & \text{if } n < k+1 \end{cases} \tag{5.12}$$

where $t(n)$ is the time taken to add n k-bit numbers. When $n = k+1$, the number of levels of reduction needed is two at most. The term c represents the constant time needed for decoding $\langle \beta^k - 1 \rangle_{\beta^k - 1} = 0$ termed as the *zero condition.*

5.2.3 Recursive Method

We can summarize the discussion from section 5.2.2 to compute $\langle X \rangle_{\beta^k - 1}$ through the following algorithm.

Procedure: 5.2.3.1 *$\langle X \rangle_{\beta^k - 1}$ can be determined through the following procedure illustrated in Fig. 5.1:*

1. *Represent the n-digit number as a weighted sum of powers of β^k as per eqn.(5.8);*
2. *Compute the sum of the $\lceil \frac{n}{k} \rceil$ k-digit factors;*
3. *Evaluate the modulus of the resultant sum represented by $\left\lceil \log_\beta \lceil \frac{n}{k} \rceil + k \right\rceil$ digits with respect to $\beta^k - 1$ recursively till the number of digits is less than $k + 1$.*

5.2.4 Calculation of modulus $\beta^k + 1$

Making use of the properties that

$$\begin{aligned} -1 &\equiv \beta^k \bmod \beta^k + 1 \\ -1^n &\equiv \beta^{nk} \bmod \beta^k + 1 \end{aligned} \tag{5.13}$$

we have

$$\begin{aligned} \langle X \rangle_{\beta^k+1} &= \langle Y_0 + \beta^k Y_1 + \beta^{2k} Y_2 + \ldots \beta^{\psi_0 k} \rangle_{\beta^k+1} \\ &= \langle \langle Y_0 \rangle_{\beta^k+1} + \langle \beta^k Y_1 \rangle_{\beta^k+1} + \ldots + \langle \beta^{\psi_0 k} Y_{\psi_0} \rangle_{\beta^k+1} \rangle_{\beta^k+1} \\ &= \langle Y_0 - Y_1 + Y_2 - Y_3 + \ldots \pm Y_{\psi_0} \rangle_{\beta^k+1} \end{aligned} \tag{5.14}$$

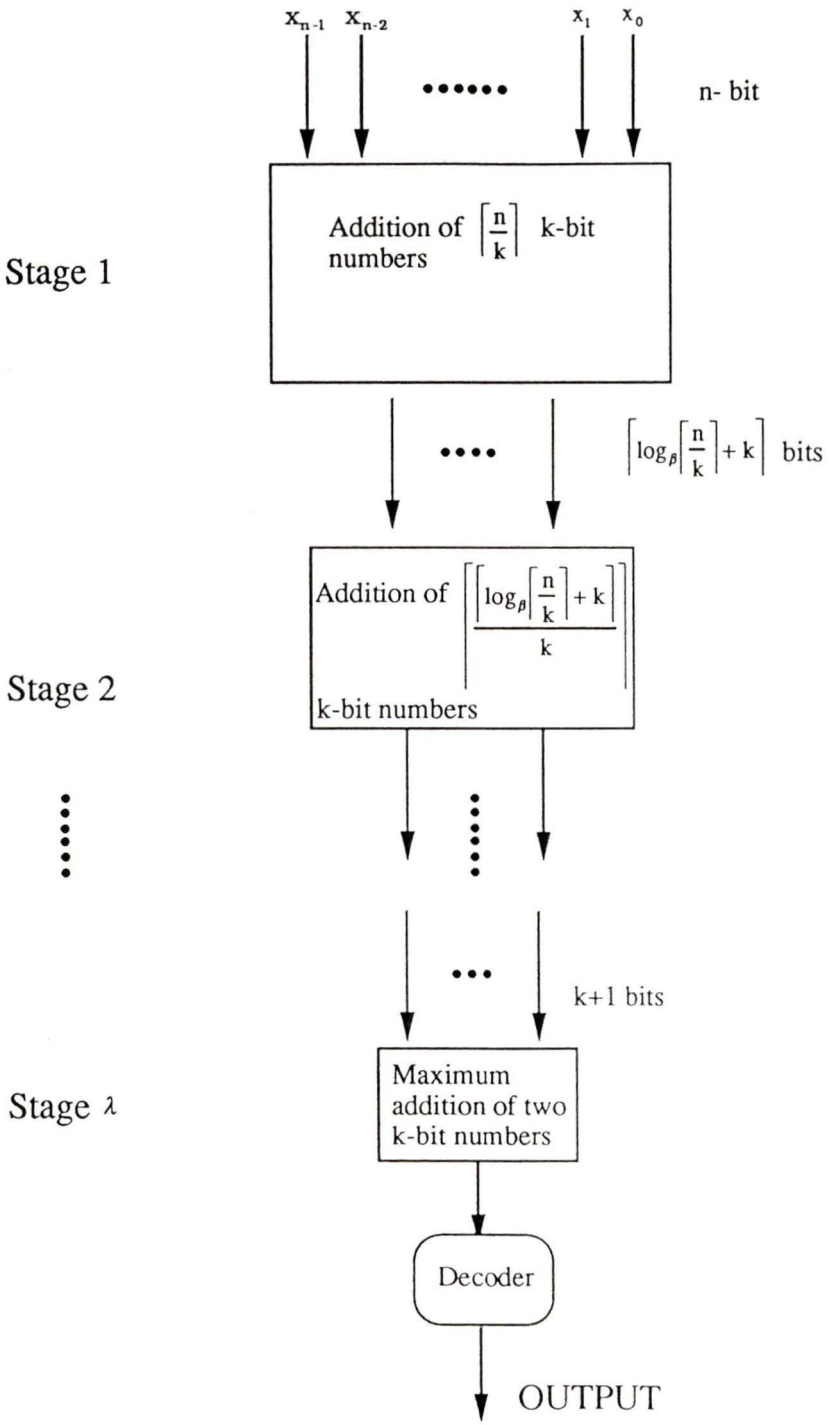


Figure 5.1: Recursive method for computing $X \bmod m, m = \beta^k - 1, \beta^k + 1$

Evaluation of $\langle X \rangle_{\beta^k+1}$ is effectively reduced to addition and subtraction of Y_i factors and the recursive method outline above can be used to determine the result.

5.2.5 Bit-sliced Method

In situations that warrant a simpler, modular and expandable structure from the design point of view, a variant of the recursive approach termed the *bit-sliced* method is proposed. In this method, a bit-slice s is chosen and the given number X is partitioned into $\lceil \frac{n}{s} \rceil$ blocks such that the following relation

$$\begin{aligned} \langle X \rangle_m &= \langle Y_0 + \beta^s Y_1 + \ldots + \beta^{s(\lceil \frac{n}{s} \rceil - 1)} Y_{\lceil \frac{n}{s} \rceil} \rangle_m \\ &= \langle \langle Y_0 \rangle_m \pm \langle Y_1 \rangle_m \pm \ldots \pm \langle Y_{\lceil \frac{n}{s} \rceil} \rangle_m \rangle_m \end{aligned} \tag{5.15}$$

is valid.

The residues of each of the $\lceil \frac{n}{s} \rceil$ sub-blocks are evaluated using the recursive approach outlined earlier and the outputs are passed to a cluster of residue adders. As shown in Fig. 5.2, the residues of the s-bit sub-blocks form the input to the residue adders which are arranged as a tree. It is assumed in this analysis that s divides n such that $\omega = \frac{n}{s}$.

Having explained the principles for evaluating the modulus for a number system defined in radix-β, we consider the special case of the binary system for which $\beta = 2$ which is a tenable proposition for digital implementation. In the following sections, the application of the Chinese Remainder Theorem to the X mod m problem and the theory of Relational Calculus will be explained.

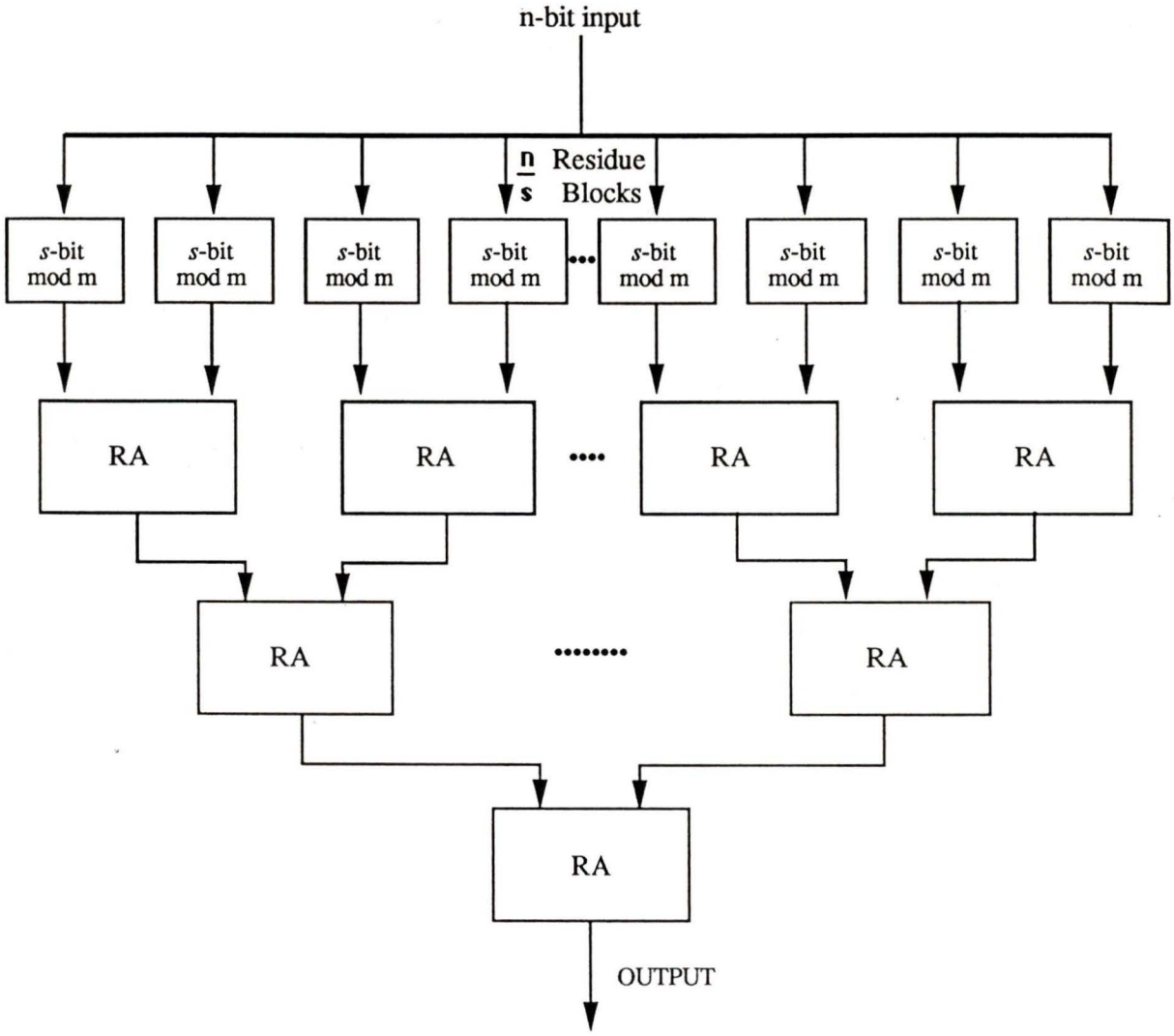


Figure 5.2: Hierarchical diagram of bit-sliced modulo extractor

5.2.6 Calculation of modulus of composite numbers

The CRT provides an elegant method to evaluate $\langle X \rangle_M$ when M is a composite number [48, 83]. If M can be expressed as a product of mutually prime[1] numbers $m_1, m_2, \ldots, m_k$ for which the moduli are known, then X mod M can be generated in terms of the prime numbers. Let

$$M = \prod_{i=1}^{k} m_i$$

$$\langle X \rangle_{m_i} = r_i \tag{5.16}$$

where r_i is the residue of X with respect to m_i. Since the $m_i s$ are mutually prime, we can write
$M_1 = M/m_1$, $M_2 = M/m_2$,..., $M_k = M/m_k$ such that GCD $(M_i, m_i) = 1$. As given in [83], we can then find numbers N_i such that

$$\langle N_i M_i \rangle_{m_i} = 1 \tag{5.17}$$

then

$$\langle X \rangle_M = \langle r_1 N_1 M_1 + r_2 N_2 M_2 + r_3 N_3 M_3 + \ldots + r_k N_k M_k \rangle_M \tag{5.18}$$

If we take the modulus of eqn.(5.18) with respect to m_i then

$$\begin{aligned} \langle \langle X \rangle_M \rangle_{m_i} &= \langle X \rangle_{m_i} \\ &= \langle r_i N_i M_i \rangle_{m_i} \end{aligned} \tag{5.19}$$

The other terms in equation(5.18) will be zero as they contain m_i. Since $\langle N_i M_i \rangle_{m_i} = 1$, we get $\langle X \rangle_{m_i} = \langle r_i \rangle_{m_i}$ for $1 \leq i \leq k$. From the above relation,

[1]Two numbers x,y are said to be mutually prime if x and y have no common divisors. GCD $(x, y) = 1$

it can be concluded that $\langle X \rangle_{m_i} = r_i$. When the $m_i s$ are not mutually prime, the above procedure can be used to determine X mod $\bar{M}$ rather than X mod M and $\bar{M}$ is represented by the least common multiple of $m_1, m_2, \ldots m_k$.

The inherent disadvantage of mapping the CRT directly in hardware is the additional overhead incurred in the form of multipliers, decoders and residue converters. To circumvent this problem, a residue mapping function (RMF) given by $F(x) : P \rightarrow Q$ is defined as follows. Let P be the set of all residue classes generated by $\langle X \rangle_M$ such that

$$P = \{0, 1, \ldots M-1\} \tag{5.20}$$

$$Q = \{n_1, n_2, \ldots, n_k \mid n_i \epsilon \{0, 1, \ldots m_i - 1 \} \} \tag{5.21}$$

$$F = \{(x, n_1, n_2, \ldots, n_k) \mid x \epsilon P, n_i = \langle x \rangle_{m_i}\} \tag{5.22}$$

From the implementation and hardware complexity point of view, the RMF is a better candidate than eqn.(5.18) since the required boolean logic for computing the modulus of M can be derived easily from the mapping tables of the residues of its prime factors. The need for multipliers and other residue decoders can therefore be avoided in the RMF. We prove that the RMF can be directly used to obtain the hardware for $\langle X \rangle_M$ through the following theorem.

Theorem 5.2.1 *$F(x)$ is a bijective function.*

Proof: To prove that $F(x)$ is a bijective function, it suffices to show that $F(x)$ is a *one-one, onto* function.

One-One

Let $n_i = x \bmod m_i$ and $\bar{n_i} = y \bmod m_i$, $\forall\ i = 1 \ldots k$. Assume that $n_i = \bar{n_i}$, $\forall\ i = 1 \ldots k$. Then we have $x \bmod m_i = y \bmod m_i$ and $x - y = A_i m_i$ where A_i is some integer for $i = 1 \ldots k$. It can be concluded from the above equation that $x - y$ is an integral multiple of m_i, $i = 1 \ldots k$. As the only numbers which are multiples of all m_is are $0, \pm M, \pm 2M \ldots$, and since the domain P consists of numbers in the range $0 \ldots M - 1$, it can be argued that the only choice that $x - y$ can assume is 0 and hence $x = y$. Each value of $x \ \epsilon\ P$ has a unique mapping to the set Q and therefore $F(x)$ is *one-one.*

Onto

The range of $F(x)$ is equal to the co-domain Q by definition and it is an *onto* function. Thus $F(x)$ is bijective.

Example:

Consider the problem of evaluating X mod 6 for which $M = 6$. M is decomposed into two prime factors $m_1 = 2$ and $m_2 = 3$. Then we obtain a mapping $F(x)$ between $\langle x \rangle_6$ and $\langle x \rangle_2$, $\langle x \rangle_3$ shown in Table 5.1 for all values of $x \ \epsilon\ P$ where$P = \{0, 1, \ldots, 5\}$. It can be easily confirmed that $F(x)$ is a bijective function. This implies that there exists a one-to-one correspondence between $\langle x \rangle_6$ and the vectors $\langle x \rangle_2 \langle x \rangle_3$ and one can uniquely determine $\langle X \rangle_6$ through its prime residues $\langle X \rangle_2$ and $\langle X \rangle_3$ as guaranteed by the CRT given by eqn.(5.18). This observation can be generalized for any composite M. Since $F(x)$ is bijective, one can uniquely determine $\langle X \rangle_M$ from its prime residues $r_1, r_2, \ldots r_k$.

$\langle x \rangle_6$	$F(x)$
	$\langle x \rangle_2 \langle x \rangle_3$
0	00
1	11
2	02
3	10
4	01
5	12

Table 5.1: Listing of RMF($F(x)$) for various values of $x \in$ P

5.3 Area-time complexity

In this section, the order of complexity of the modulo-extractor is evaluated in terms of silicon area A and the response time T of the circuit. Expressions are derived for the recursive and bit-sliced methods mentioned previously. To analyze the area-time complexity of the circuit, we consider the cases when $m = 2^k - 1, 2^k + 1$.

5.3.1 Recursive Method

5.3.1.1 Area Complexity

The recurrence relation for the recursive method given by eqn.(5.12) suggests that maximum complexity occurs when the addition of $(\lceil \frac{n}{k} \rceil)$ k-bit numbers is performed in the first stage of the algorithm. The subsequent steps are, by and large, less involved since the representative size of the number is reduced logarithmically given by eqn.(5.11). It can be stated that the area and time complexity of the circuit is therefore dominated by the addition in the first step.

Assuming that each k-bit adder has an area $\mathcal{O}(k \log k)$ [93], the number of adders needed in the first step of the algorithm is $\lceil \frac{n}{k} \rceil$. The area complexity of the circuit is given by

$$\begin{aligned} A_R &= k \log k \left\lceil \frac{n}{k} \right\rceil + k \log k \left\lceil \frac{\lceil \log \lceil \frac{n}{k} \rceil + k \rceil}{k} \right\rceil + \\ &\quad + \mathcal{O}(k \log k \log \log \left\lceil \frac{n}{k} \right\rceil) \qquad (5.23) \\ &\approx k \log k \left\lceil \frac{n}{k} \right\rceil + \log k \log \left\lceil \frac{n}{k} \right\rceil \qquad (5.24) \end{aligned}$$

If $n \gg k$, then

$$A_R = \mathcal{O}(\left\lceil \frac{n}{k} \right\rceil) \qquad (5.25)$$

5.3.1.2 Time Complexity

If the adders are arranged in the form of a tree, the number of steps required to add $\lceil \frac{n}{k} \rceil$ k-bit numbers is $\log \lceil \frac{n}{k} \rceil$. If each k-bit adder has a time complexity $\mathcal{O}(\log k)$ [93], then the total time complexity of the circuit is given by

$$\begin{aligned} T_R &= \log k \log \left\lceil \frac{n}{k} \right\rceil + \log k \log \left\lceil \frac{\lceil \log \lceil \frac{n}{k} \rceil + k \rceil}{k} \right\rceil + \\ &\quad + \mathcal{O}\left(\log k \log \log \left\lceil \frac{n}{k} \right\rceil\right) \qquad (5.26) \end{aligned}$$

When $n \gg k$, we have

$$T_R = \mathcal{O}\left(\log \left\lceil \frac{n}{k} \right\rceil\right) \qquad (5.27)$$

Hence the area-time complexity for the structure is obtained by combining eqns. (5.25) and (5.27) to get

$$\begin{aligned} AT_R &= \mathcal{O}(\left\lceil \frac{n}{k} \right\rceil \log \left\lceil \frac{n}{k} \right\rceil) \\ &= \mathcal{O}(\psi_0 \log \psi_0) \qquad \text{where } \psi_0 = \lceil \tfrac{n}{k} \rceil \qquad (5.28) \end{aligned}$$

5.3.2 Bit-sliced Method

5.3.2.1 Area Complexity

As was explained in Section 5.2.5, the crux of the bit-sliced method lies in evaluating mod m for each bit-slice of length of s using the recursive method and the output is processed through the tree of residue adders as given in Fig. 5.2. The bit-sliced method is based on the observation that

$$\begin{aligned}\langle\langle X_1\rangle_m + \langle X_2\rangle_m + \ldots \langle X_k\rangle_m\rangle_m &= \langle\langle X_1\rangle_m + \langle X_2\rangle_m\rangle_m + \\ &+\langle\langle X_3\rangle_m + \langle X_4\rangle_m\rangle_m + \ldots \\ &+\langle\langle X_{k-1}\rangle_m + \langle X_k\rangle_m\rangle_m \qquad (5.29)\end{aligned}$$

The residue adders are modified versions of conventional adders in that they accept the residues from the previous blocks and compute the resultant modulus. In other words, if $\langle X_1\rangle_m$ and $\langle X_2\rangle_m$ are known, then the residue adder can determine $\langle\langle X_1\rangle_m + \langle X_2\rangle_m\rangle_m$. For example, when $m = 2^k - 1$, the residue adder determines the result according to the following conditions.

1. If $\langle X_1\rangle_{2^k-1} + \langle X_2\rangle_{2^k-1} < 2^k - 1$, then the solution is given by the result $\langle X_1\rangle_{2^k-1} + \langle X_2\rangle_{2^k-1}$.

2. If $\langle X_1\rangle_{2^k-1} + \langle X_2\rangle_{2^k-1} = 2^k - 1$, then we have a string of k 1's and the result is decoded to be zero since $\langle 2^k - 1\rangle_{2^k-1} = 0$.

3. If $\langle X_1\rangle_{2^k-1} + \langle X_2\rangle_{2^k-1} > 2^k - 1$, then the addition will result in a $k+1$ bit number, then

$$\begin{aligned}\langle\langle X_1\rangle_{2^k-1} + \langle X_2\rangle\rangle_{2^k-1} &= \langle 2^n + \langle\langle X_1\rangle_{2^k-1} + \langle X_2\rangle_{2^k-1}\rangle_{2^k}\rangle_{2^k-1} \\ &= 1 + \langle\langle X_1\rangle_{2^k-1} + \langle X_2\rangle_{2^k-1}\rangle_{2^k} \qquad (5.30)\end{aligned}$$

We get the required solution by adding the carry bit to the least significant k-bits after the first addition is performed. In effect, two levels of addition is performed. The residue adder (RA) is composed of two adders, the first one for adding the two k bit numbers and the second for absorbing the resultant carry, if any, to the previous sum. The adders can be constructed to have a area and time complexity of $\mathcal{O}(k \log k)$ and $\mathcal{O}(\log k)$ respectively while the decoding circuit for the zero condition can be taken to have constant delay. A similar structure is used for the case $m = 2^k + 1$ but with subtractors instead of adders.

The area of the circuit using the bit-sliced approach is composed of two parts, namely, the area occupied by the bit-sliced stages and the residue adder tree. Since there are $\frac{n}{s}$ mod m blocks, the area for the bit-sliced stage is given by

$$\begin{aligned} A_s &= \left(\frac{n}{s}\right)\left[k \log k \left\lceil \tfrac{s}{k} \right\rceil + k \log k \left\lceil \tfrac{\lceil \log \lceil \frac{s}{k} \rceil + k \rceil}{k} \right\rceil + \right. \\ &\quad \left. + \mathcal{O}(k \log k \log \log \left\lceil \tfrac{s}{k} \right\rceil)\right] \qquad (5.31) \\ \Rightarrow A_s &< \frac{n}{s}\left[\gamma k \log k \left\lceil \tfrac{s}{k} \right\rceil\right] \\ \Rightarrow A_s &< \gamma k \log k \frac{n}{s}\left(\frac{s}{k} + 1\right) \\ \Rightarrow A_s &< \gamma k \log k \left(\frac{n}{k} + \frac{n}{s}\right) \\ \Rightarrow A_s &< \gamma k \log k \left(\left\lceil \tfrac{n}{k} \right\rceil + \frac{n}{s}\right) \\ \Rightarrow A_s &< \mathcal{O}\left(\left\lceil \tfrac{n}{k} \right\rceil + \frac{n}{s}\right) \qquad (5.32) \end{aligned}$$

and γ is a constant.

The area for the residue adder tree is given by

$$A_r = \left(\frac{n}{s} - 1\right) k \log k \tag{5.33}$$

When $n \gg k$, the total area (A_S) of the bit-sliced approach is obtained from eqns.(5.32) and (5.33) and on simplification we get

$$\begin{aligned} A_S &= \mathcal{O}\left(\left\lceil \tfrac{n}{k} \right\rceil + \frac{n}{s}\right) \\ &= \mathcal{O}\left(\mathrm{MAX}\left(\psi_0, \omega\right)\right) \end{aligned} \tag{5.34}$$

where $\omega = \frac{n}{s}$. Thus the area of the bit-sliced approach is dominated by either ψ_0 or ω which ever is greater of the two.

5.3.2.2 Time Complexity

The time complexity for the bit-sliced approach is dictated by the delay of the s-bit modulo blocks and the delay of the residue adder tree. The delay of the s-bit blocks, T_s follows eqn.(5.26) except n is substituted by s.

$$\begin{aligned} T_s = {} & \log k \log \left\lceil \tfrac{s}{k} \right\rceil + \log k \log \left\lceil \tfrac{\lceil \log \lceil \frac{s}{k} \rceil + k \rceil}{k} \right\rceil + \\ & + \mathcal{O}\left(\log k \log \log \left\lceil \tfrac{s}{k} \right\rceil\right) \end{aligned} \tag{5.35}$$

The residue adder tree has a delay given by

$$T_r = \log k \log \frac{n}{s} \tag{5.36}$$

Thus the time complexity, T_S, of the bit-sliced approach is given by combining eqns.(5.35) and (5.36) to get

$$T_S = \log k \log \left\lceil \tfrac{n}{k} \right\rceil + \log k \log \left\lceil \tfrac{\lceil \log \lceil \frac{s}{k} \rceil + k \rceil}{k} \right\rceil +$$

$$+\mathcal{O}\left(\log k \log\log \left\lceil \frac{s}{k} \right\rceil\right) \tag{5.37}$$

When $n \gg k$ and for small s, we have

$$T_S = \mathcal{O}\left(\log \left\lceil \frac{n}{k} \right\rceil\right) \tag{5.38}$$

The above equation suggests that the time complexity of both methods considered in this analysis are comparable. If we take into account the second order terms, the bit-sliced approach will be faster than the normal recursive method depending on the ratio n/k and s/k.

The area-time complexity of the bit-sliced approach is given by

$$\begin{aligned} AT_B &= \mathcal{O}\left(\left\lceil \frac{n}{k} \right\rceil \log \left\lceil \frac{n}{k} \right\rceil, \frac{n}{s} \log \left\lceil \frac{n}{k} \right\rceil\right) \\ &= \mathcal{O}\left(\mathrm{MAX}\left(\psi_0 \log \psi_0, \omega \log \psi_0\right)\right) \end{aligned} \tag{5.39}$$

5.4 Functional description of the modulo extractor

The parallel, modular and bit sliced characteristics of RNS lends itself to concurrent processing in that many arithmetic operations can be performed in tandem in separate modules leading to high data throughput and reduced execution time. The main computing element used in the modulo-extractor is the adder and no multiplication is involved. This paves the way for high speed computation. In this thesis, the design of a modulo-extractor for computing $\langle X \rangle_{2,3,\ldots 10}$ where X is a 16-bit number was considered for a graph of dimension-10. The logical organization of the modulo extractor is given in Fig. 5.3. The various sub-components of the circuit will be discussed in the following sections.

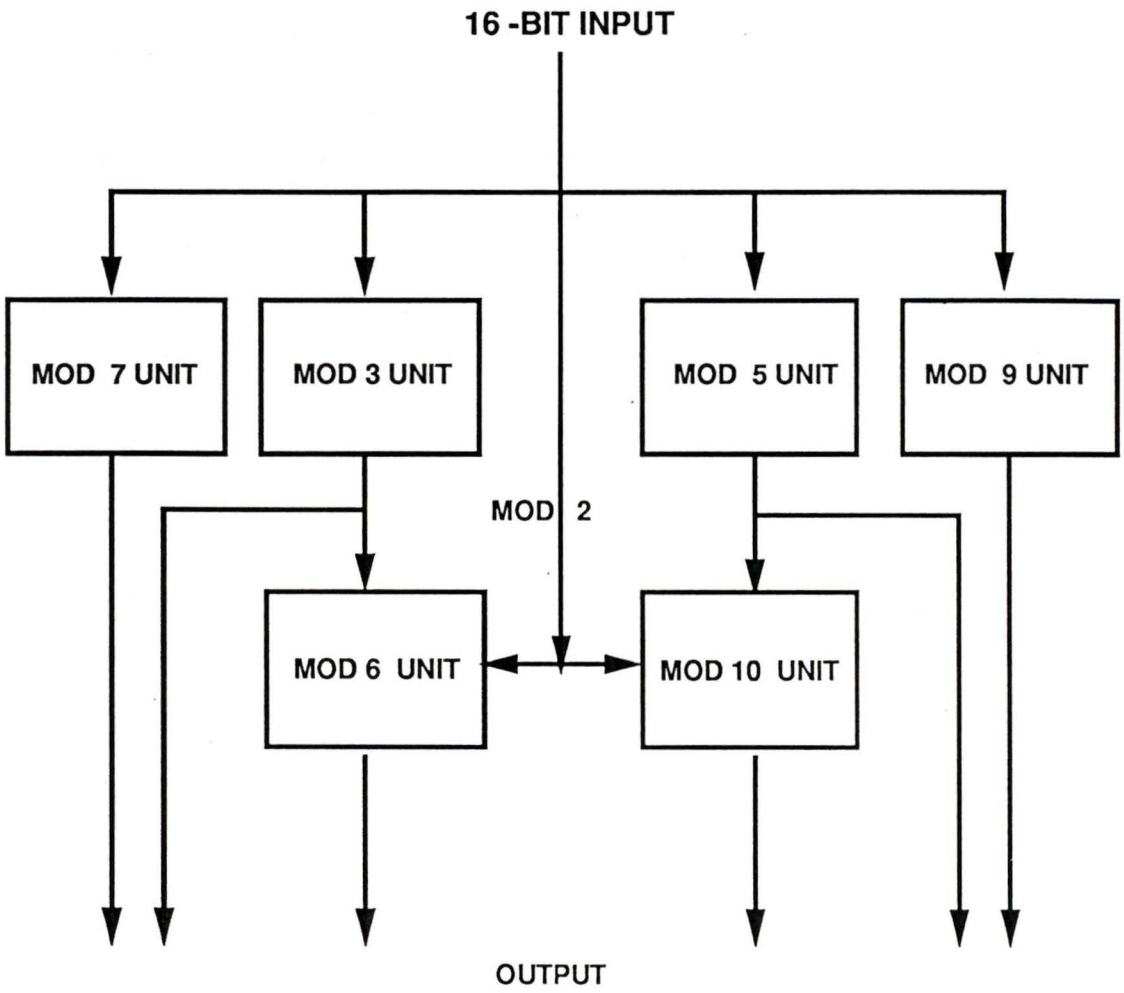


Figure 5.3: Logical organization of modulo extractor

5.4.1 X mod $2, 4, 8$

The modulo is of the form 2^k with $k = 1, 2, 3$. Using eqn.(5.5), we obtain the following results.

$$\langle X \rangle_2 = x_0 \tag{5.40}$$

$$\langle X \rangle_4 = x_1 x_0 \tag{5.41}$$

$$\langle X \rangle_8 = x_2 x_1 x_0 \tag{5.42}$$

By selectively masking the appropriate bits, X mod $2, 4, 8$ can be computed easily.

5.4.2 X mod 3

The design of $\langle X \rangle_3$ is based on eqn.(5.8) since the modulo is of type $2^k - 1$ with $k = 2$. The bit-sliced approach explained previously is adopted in designing $\langle X \rangle_3$ by partitioning X into four 4-bit numbers, X_0, X_1, X_2, X_3 such that $X = 2^{12}X_3 + 2^8X_2 + 2^4X_1 + X_0$. Then we can express

$$\begin{aligned}
\langle X \rangle_3 &= \langle 2^{12}X_3 + 2^8X_2 + 2^4X_1 + X_0 \rangle_3 \\
&= \langle \langle 2^{12}X_3 \rangle_3 + \langle 2^8X_2 \rangle_3 + \langle 2^4X_1 \rangle_3 + \langle X_0 \rangle_3 \rangle_3 \\
&= \langle \langle X_3 \rangle_3 + \langle X_2 \rangle_3 + \langle X_1 \rangle_3 + \langle X_0 \rangle_3 \rangle_3
\end{aligned}$$

The problem is thus reduced to the one of evaluating mod 3 for each of the four 4-bit numbers X_3, X_2, X_1, X_0 and the results are passed through the residue adders to get the resultant vector. The mod 3 unit for a 4-bit input is constructed using the recursive method as explained previously.

5.4.3 X mod $5, 7, 9$

When $m = 5$, the modulo is of type $2^k + 1$ with $k = 2$. Using eqn.(5.8) we get

$$\begin{aligned} \langle X \rangle_5 &= \langle Y_0 - Y_1 + Y_2 - Y_3 + Y_4 - Y_5 + Y_6 - Y_7 \rangle_5 \\ &= \langle (Y_0 + Y_2 + Y_4 + Y_6) - (Y_1 + Y_3 + Y_5 + Y_7) \rangle_5 \end{aligned} \tag{5.43}$$

The methodology for designing X mod 7 and X mod 9 is similar to X mod 3 and X mod 5 except that $k = 3$ in these cases.

5.4.4 X mod $6, 10$

We have stated earlier that the CRT can be implemented through the RMF and that the boolean function can be obtained from the residues of the prime factors. To compute X mod 6, we dissociate 6 into its prime factors 2 and 3. Using the logic circuits of $\langle X \rangle_2$ and $\langle X \rangle_3$, we can determine $\langle X \rangle_6$. Similarly $\langle X \rangle_{10}$ is obtained from $\langle X \rangle_2$ and $\langle X \rangle_5$. The CRT for the two cases is given by

$$\langle X \rangle_6 = \langle 3\hat{r_2} + 4\hat{r_3} \rangle_6 \tag{5.44}$$

$$\langle X \rangle_{10} = \langle 5\hat{r_2} + 6\hat{r_5} \rangle_{10} \tag{5.45}$$

where $\hat{r_2}$, $\hat{r_3}$ and, $\hat{r_5}$ are the residues of X with respect to 2,3 and 5 respectively. The residue mapping table similar to Table 5.1 can be formed for the above cases and the boolean logic can be easily derived.

Simulation Data for Modulo-Extractor						
Input(hex)	Output(hex)					
A	MOD3	MOD5	MOD6	MOD7	MOD9	MOD10
FFDC	1	0	4	1	7	0
FFDD	2	1	5	2	8	1
FFDE	0	2	0	3	0	2
FFDF	1	3	1	4	1	3
FFE0	2	4	2	5	2	4
FFE1	0	0	3	6	3	5

Table 5.2: Simulation data for Modulo-Extractor

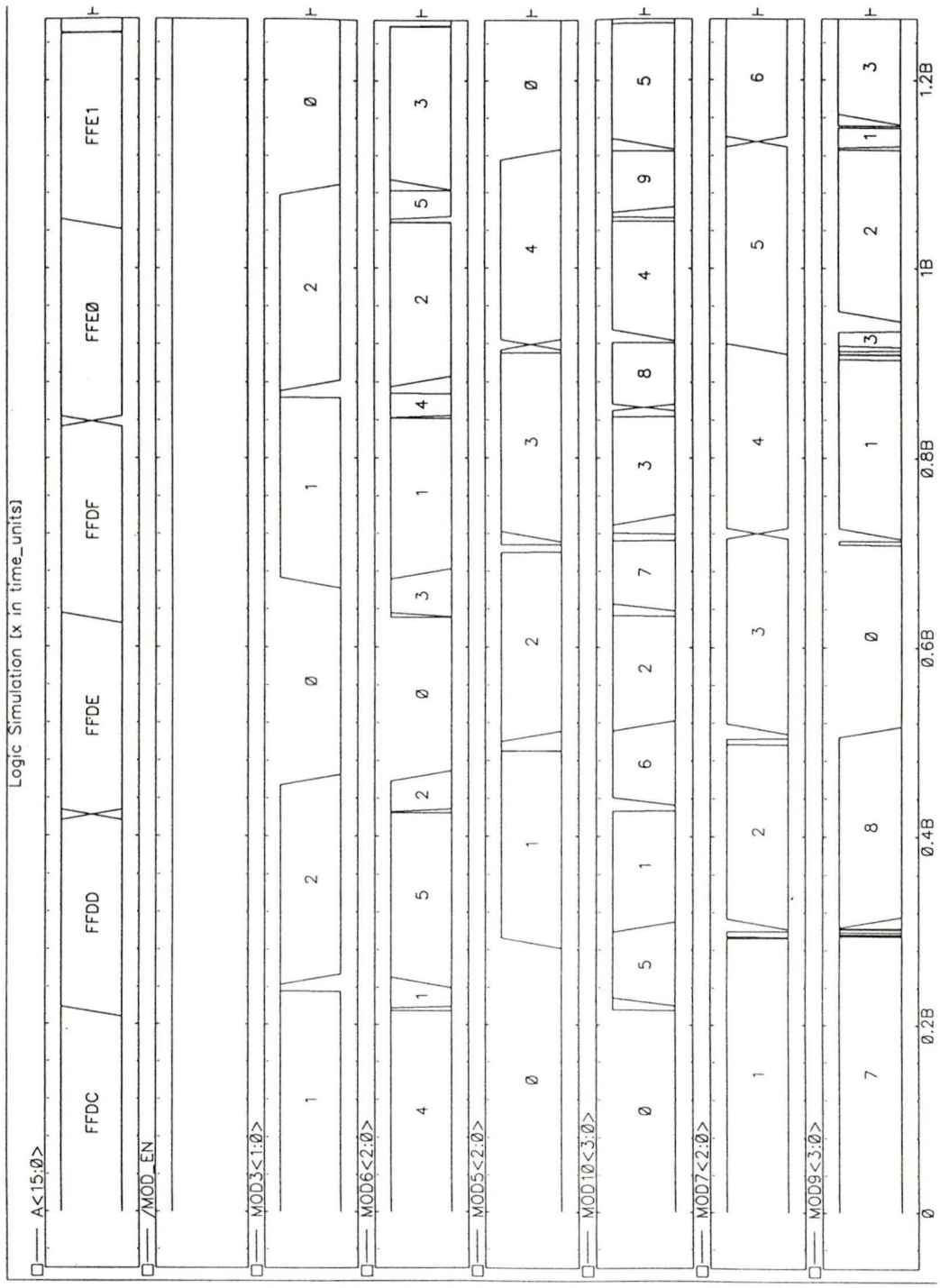
5.5 Implementation of Modulo-Extractor

A 16-bit modulo-extractor with moduli $2, 3, \ldots, 10$ has been implemented. in 3μ CMOS Double Layer Metal technology. The Cadence tool was used to design the circuit. The schematic diagrams of the components of the modulo extractor are given in Appendix B. Simulation results of the modulo-extractor is presented in Fig. 5.4 for the test vectors given in Table 5.2. As an illustration, the 16-bit input FFDD is applied to the circuit at time $T = 200ns$ and the output settles down at approximately $T = 290ns$. The MOD_EN pin is asserted during the normal operation of the modulo-extractor.

The statistics of the chip are given in Tables 5.3 and 5.4. The VLSI layout of the modulo extractor is shown in Fig. (5.5).

5.6 Discussion

The proposed structure has significant advantages in terms of speed and area over the model proposed in [85]. It has been claimed that the VLSI structure for



$B = 1\mu s$

Figure 5.4: SILOS Simulation of modulo-extractor (3 μ CMOS3DLM)

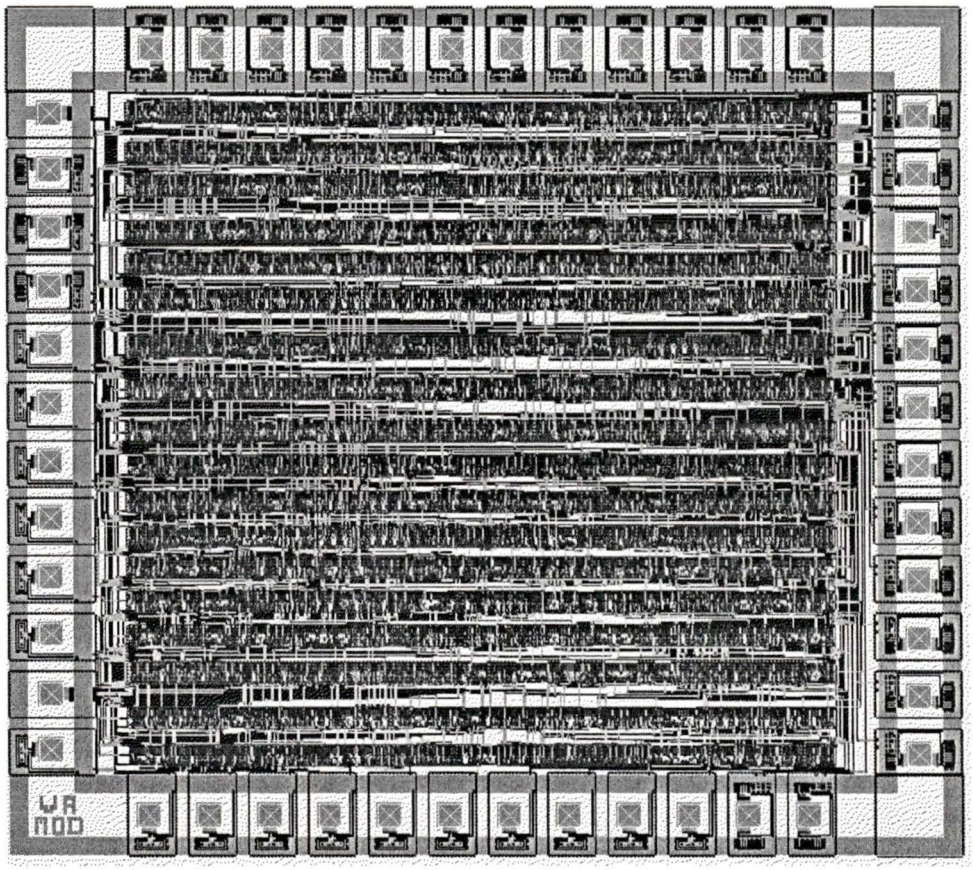


Figure 5.5: VLSI layout of 16-bit modulo-extractor(3 μ CMOS3DLM)

16-bit Modulo Extractor	
Technology	3μ NTE CMOS3DLM
Area of the chip	4545.1x4545.1 μ^2m
Area of the core	3681.1x3681.1 μ^2m
No. of pins	38
No. of standard cells	1392
No. of transistors	5156
Propagation delay (worst case)	95ns
Area*time	1.2873e09 nsμ^2m

Table 5.3: Chip statistics of the implemented 16-bit modulo extractor

Modulo Extractor	
Gate	Count
Inverters	461
Buffers	137
Nand2	507
Nand3	114
Nand4	44
Nor2	89
Nor3	27
Nor4	13

Table 5.4: Gate level statistics of the implemented 16-bit modulo extractor

computing X mod m has a response time of less than 200 ns for 32-bit numbers. In our model, the propagation delay for 16-bit operands is less than 100ns in the 3 μ technology. It has been shown that the increase in time complexity for the bit sliced method is proportional to $\log_2 \psi_0$ as the operand length increases. The implemented model is totally devoid of any multipliers and uses only high speed adders. This accounts for a drastic reduction in silicon real estate and propagation delay. It has been shown that the area-time complexity of the circuit is $\mathcal{O}(\psi_0 \log \psi_0)$. Besides the structure proposed is amenable to pipeline implementation. The demerits of our model is that it cannot be used for prime numbers which are not of the form $2^k - 1$ or $2^k + 1$. Besides the mapping table of an RMF for higher order modulus could be complicated and the hardware overhead is increased on account of the large number of inputs.

5.7 Conclusion

The technological advantages of VLSI coupled with the regular, modular structure of residue arithmetic have made RNS a viable proposition for implementation. In this thesis, a structure for the VLSI design of $\langle X \rangle_m$ has been discussed. This circuit is particularly useful for computing $\langle X \rangle_m$ for small values of m with X being a 16 or 32-bit number. Expressions for the asymptotic area-time complexity of the model has been derived. Technology scaling should provide a quantum increase in performance and throughput. The circuit was simulated in the 1.2 μ CMOS4S technology and it yielded a worst case propagation delay of less than 20 ns. A refined version of the modulo extractor for computing X mod $1, 2, 3, 4$ where X is a 16-bit number has been incorporated in the Port Selector circuit of the Hypercycle router.

Chapter 6

Design of a Random Number Generator

6.1 Introduction

In the context of the Hypercycle router, we need to design and implement a random number generator for the BTOR scheme which portrays a reasonable degree of randomness and uniform distribution and which produces random numbers at a rate that matches the throughput of the router. Software synthesis of random numbers from the host node is an option that deserves consideration but the degradation in performance (speed) greatly offsets the quality of random numbers produced.

It is therefore of paramount importance to make random numbers available during each clock cycle for the router to pass decisions on the next port selection. Hence to maximize the throughput of the router the RNG needs to be implemented in hardware. Since VLSI has been chosen as the medium for implementation, the area-time complexity of the proposed RNG should be commensurate with the requirements of the Hypercycle router chip. The algo-

rithm used for the RNG implementation should exhibit a low degree of area-time complexity. The tradeoff between the algorithm and area-time complexity has a direct bearing on the quality of numbers produced. These factors merit consideration in designing the RNG. In this chapter, the various methods for implementing the RNG in hardware is discussed. A random number generator of the mixed congruential type is proposed for the BTOR scheme. The proposed RNG is compared with other types of generators through statistical tests and the performance/cost ratio is analyzed. Finally the VLSI implementation of the chosen RNG in CMOS3DLM and CMOS4S technologies is considered along with simulation results.

6.2 Random Numbers

Any algorithm that generates random numbers is characterized by two properties.

1. *Periodicity*

2. *Randomness.*

A sequence generator usually cycles through a finite number of states before it repeats itself. In principle, a generator with a fairly long period is desired in many applications. Since the numbers of the series are determined through a deterministic process and are predictable, the term *Pseudo-random* is used frequently.

We give a formal definition for the meaning of pseudo-random numbers.

Definition 6.2.1 *A sequence of pseudo-random numbers [97] is a deterministic sequence of numbers in* [0,1] *having the same relevant statistical properties as a sequence of truly random numbers.*

The fundamental requirement of any random sequence is that each number of the series should have an equal probability of taking one of the possible values and must be statistically independent from one another.

Most algorithms for pseudo random number generation are suitable for software implementation. However a number of methods are available for constructing RNG in hardware also. This has been made possible partly by the recent advances in VLSI technology. Hardware circuits from traditional counters, Linear Feedback Shift Registers (LFSR), congruential generators to the modern RNG based on cellular automata have been used as PRNGs in several applications such as coding theory [94], signature analysis and testability [95, 96], simulation and modeling [97, 98], ALU of microprocessors, parallel computers and supercomputers [101, 102, 103].

6.2.1 Shift Register Generators

The linear feedback shift registers (LFSR) constitute an important class of shift register generators. In this section, we consider a particular case of LFSR called the autonomous linear feedback shift registers (ALFSR) operating in GF(2). They consist of a series of flipflops connected in a feedback loop through exclusive-or gates with no external inputs. The basic structure of a n bit ALFSR is shown in Fig. 6.1.

The symbols h_i multiply the output of the various stages and feed the exclusive-or gate. Since we are dealing in the GF(2) domain, the multipliers h_i assume the following interpretation. When $h_i = 1$, there is a feedback from the i^{th} stage and if $h_i = 0$, there is no connection. The state of the shift register

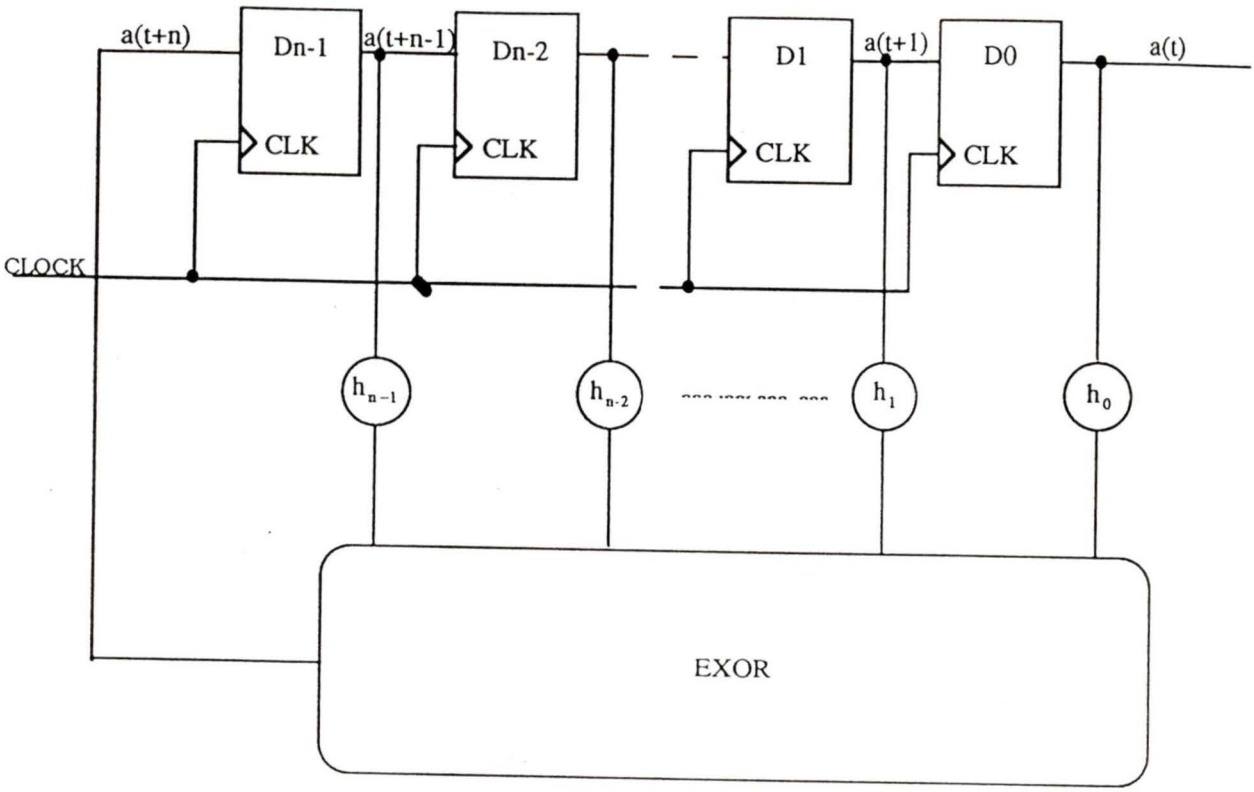


Figure 6.1: General structure of n-bit ALFSR

generator at any time $t+n$ is given by

$$a(t+n) = \sum_{i=0}^{n-1} h_i a(t+i) \bmod 2 \tag{6.1}$$

The above equation is expressed in the form of a polynomial over an independent x and whose coefficients are in GF(2) given by

$$f(x) = x^n + h_{n-1}x^{n-1} + \ldots + h_1 x + h_0 \tag{6.2}$$

The polynomials give an insight into the feedback paths present in the RNG. The generating function corresponding to maximum length $(2^n - 1)$ ALFSR is called the *characteristic polynomial* which in this case is "primitive" [94]. As

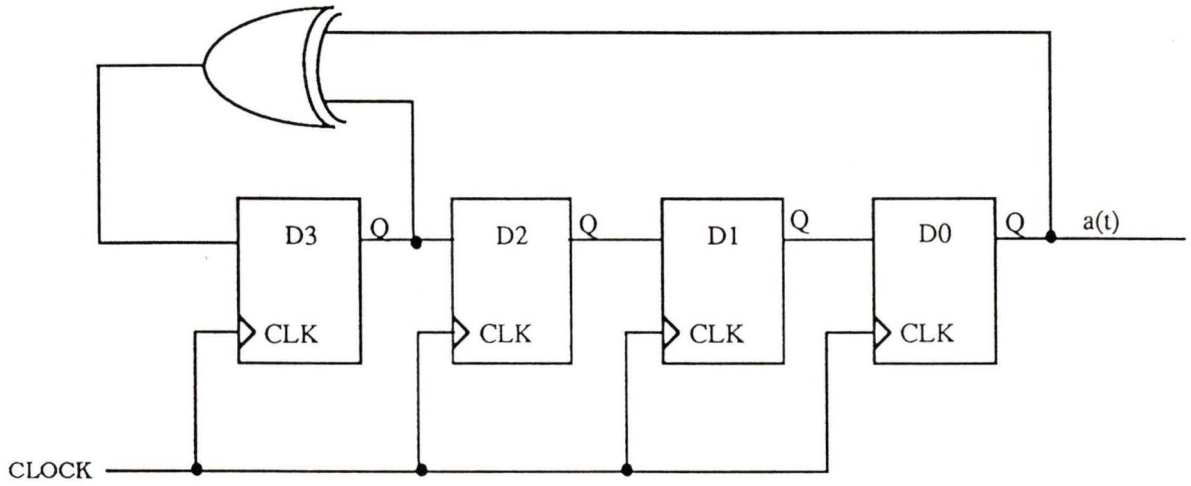


Figure 6.2: Schematic diagram of a 4-bit ALFSR

an illustration, consider the 4-bit ALFSR shown in Fig. 6.2. The feedback connections to the exor gate is provided such that h_0 and h_3 are set to 1 while $h_1, h_2 = 0$. The characteristic polynomial for the above configuration is given by

$$f(x) = x^4 + x^3 + 1 \tag{6.3}$$

The theory behind the characteristic polynomials and shift register generators is given in [94, 95]. From the tables given in [95], an ALFSR can be tailored to match the period and size of the generator as desired. As can be inferred from Fig. 6.2, the fundamental building blocks needed for a n bit ALFSR implementation include n D-type flip flops and exclusive-or gates and the hard-

ware is therefore simple to construct. The shift register produces a new pseudo random number during each clock cycle and the clock period is dependent on the flipflop propagation delay and exclusive-or gate delays. These factors make the LFSR an ideal candidate for VLSI implementation. However the quality of random numbers produced is not exemplary [103] and this overshadows their otherwise superior implementation merits. It has been reported in [103] that random number generators based on cellular automata are superior to LFSRs in this respect. Another obvious disadvantage is the personalization of the ALFSR for a fixed polynomial in that the sequence cannot be altered.

6.2.2 Congruential Methods

The most successful RNG in contemporary use is based on the congruence relationship of the mathematical theory of numbers suggested by Lehmer [104]. The generation of a random number is usually conceived from the recursive relation $r_{i+1} = f(r_i)$, with r_0 being the seed of the RNG. We shall consider different types of congruential generators.

6.2.2.1 Mixed Congruential Method

The Mixed Congruential (MC) method generates a sequence of random numbers in accordance with the recurrence relation given by

$$r_{n+1} = (ar_n + c) \bmod m \tag{6.4}$$

where a,c,m are positive integers with $a \leq m$ and $c \leq m$. Other congruential methods are basic derivatives of the MC method. They are

1. **Multiplicative Congruential Method:**
   This is a special case of equation(6.4) with $c = 0$.

2. **Additive Congruential Method:**

   Equation(6.4) reduces to this form when $a = 1$ with c being replaced by another number such as a previous random number in the sequence. The random numbers are generated by the equation

$$r_{n+1} = (r_n + r_{n-l}) \bmod m \tag{6.5}$$

   where l is the displacement of the sequence.

The maximum period length which is one of the important features of the RNG is always less than or equal to m depending on the choices of the parameters involved in equation(6.4). We state an important theorem in the context of the mixed congruential RNG that gives the necessary conditions for determining the maximum period of the sequence.

Theorem 6.2.1 *A congruential generator [105] has full period m iff*

1. *gcd(c,m) = 1.*
2. $a \equiv 1 \bmod p$ *for each prime factor p of m.*
3. $a \equiv 1 \bmod 4$ *if 4 divides M.*

Proof: The formal proof of the above theorem is given in detail in [105] pp. 16-20.

6.2.3 Design of Mixed Congruential Random Number Generator

In this section, we analyze the design of a mixed congruential RNG of the form

$$u_{n+1} = f(u_n) \bmod m \tag{6.6}$$

$$= (u_{n-1} + c) \bmod m \tag{6.7}$$

and discuss the selection of parameters for this RNG. This is essentially similar to eqn.(6.4) with $a = 1$. The value of $a = 1$ has been chosen in order to offset multiplication complexity and thereby reduce the consumption of silicon real-estate. Another important design parameter is the choice of m which singularly determines the period of the RNG. A number of generators with $m = 2^n$ have been used in the past and extensive references on the same can be found in [106, 107]. The evaluation of mod 2^n is simple and direct with minimum hardware and this is corroborated by the residue arithmetic theory explained in the previous sections. Despite its simplicity, there has always been shortcomings with these type of generators [108] and the most notorious and infamous among them is the IBM/360 product RANDU [110]. Furthermore, it has been shown in [110] that a choice of $a = 16,807$ and $m = 2^{31} - 1$ called the *Mersenne Prime* has been widely accepted as a standard for 32-bit computers since according to the authors, the generator has a full period, is sufficiently random and its characteristics have been exhaustively tested and verified.

A 16-bit RNG was deemed sufficient for the Hypercycle Router Engine to match the chosen configuration and the available silicon area. A value of $m = 2^{16} - 1$ was chosen in our case. The theory behind the modulus operation for $m = 2^n - 1$ and its associated hardware, explained in the preceding sections, is applied for the design of the RNG at hand. As advocated in [110], the number

16,807 is stored as c to generate the random numbers. It should be noted that the coefficient c is made programmable in the design and can be altered when needed to get different sequences. Simulation results of the RNG with $a = 1, c = 16807$ and $m = 2^{16} - 1$ have been found to be satisfactory. The maximum cycle length of $2^{16} - 1$ is guaranteed through the theorem for these parameters. Given a sequence of random numbers $r_1, r_2, \ldots, r_n$, in the range [0,m-1], we can normalize r_i with respect to $m - 1$ such that $u_i = \frac{r_i}{m-1}$ and assume u_i to be a uniformly distributed sequence in [0,1]. A normalized plot

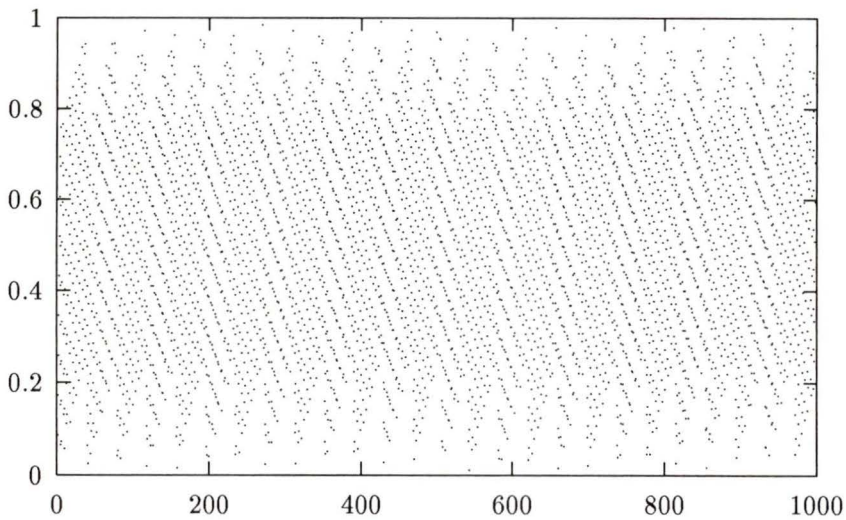


Figure 6.3: Normalized plot of 1000 random numbers generated by mixed congruential method with $a = 1$,$c = 16807$,$m = 65535$

of the distribution of 1000 random numbers generated through the MC method is given in Fig. 6.3. Before the details of implementation of the RNG are considered, the performance and test results of the proposed generator will be the discussed in the next section.

6.3 Testing Random Number Generators

Over the past, several statistical procedures have been proposed for testing whether a sequence of numbers constitutes a sample of random numbers or not. The output of a RNG must meet certain defined requirements to be classified as a random sequence.

Any generator can be tested empirically by applying statistical tests for independence and uniformity for a large sequence. Knuth [105] has made an elegant and exhaustive analysis on the topic of testing RNG. Many of these tests are statistical in nature and perform a counting/searching operation on a given series of numbers and the results are compared with a known distribution like chi-square. Based on these tests, one can make judgements on the probability that the obtained values depart from what is expected of a truly random sequence. It is also possible to determine the statistical distribution of the random numbers. We restrict ourselves to a few of the important tests since these are deemed sufficient to categorize the randomness of the generator. A probability of 90 - 95 % for the chi-square tests is fixed as the metric for judging the various RNGs in this analysis. The statistics compiled from the various tests on the proposed RNG is compared and contrasted with the results obtained from a cross section of random numbers generated from the *rand()* function of Unix 4.1c OS and a 16-bit LFSR whose characteristic polynomial is given by

$$f(x) = x^{16} + x^5 + x^4 + x^3 + 1 \tag{6.8}$$

Note that this polynomial has a period of $2^{16} - 1$ and hence matches the length of the proposed RNG.

6.3.1 Equidistribution test

The equidistribution test or frequency test is used to determine whether the numbers are uniformly distributed between 0 and 1. To determine the distribution of the random sequence, the range of the given set of n numbers is divided into r categories of equal probability p and the number of occurrences k_i in each category is counted. The Pearson chi-square statistic is given by

$$\chi^2 = \sum_{i=1}^{r} \frac{(k_i - np)^2}{np} \tag{6.9}$$

with $\nu = r-1$ degrees of freedom. The probability for a random variable X with a hypothetical chi-square distribution that would exceed the observed value is computed from the chi-square table. The test was repeated on several arbitrary samples for a given degree of freedom and size. The results of the equidistribution test is listed under Table (6.1). An analogous test was conducted on the

Categories	Size	Degrees of Freedom	Probability
c	n	ν	$P(X \geq \chi^2)$
5	400	4	99.39
3	1000	2	98.71
10	100	9	97.81
4	100	3	97.06
5	100	4	92.46
64	300	63	90.15
8	100	7	83.30

Table 6.1: Chi-square statistics of the proposed RNG

random numbers generated from Unix and the results are shown in Table 6.2. The equidistribution test for the LFSR is listed in Table 6.3.

Categories	Size	Degrees of Freedom	Probability
c	n	ν	$P(X \geq \chi^2)$
5	400	4	81.77
3	1000	2	94.95
10	100	9	88.32
4	100	3	98.33
5	100	4	89.74
64	300	63	87.61
8	100	7	92.24

Table 6.2: Chi-square statistics of Unix RNG

The random sequences of our model and the LFSR are closer to the expected distribution than the Unix case. Furthermore, all the three methods fall within the assumed range hence the distribution can be taken to be uniform.

Categories	Size	Degrees of Freedom	Probability
c	n	ν	$P(X \geq \chi^2)$
5	400	4	95.43
3	1000	2	87.57
10	100	9	97.80
4	100	3	97.06
5	100	4	93.85
64	300	63	88.11
8	100	7	99.54

Table 6.3: Chi-square statistics of LFSR

6.3.2 Run test

The basic tenet of the run test is to examine the given sequence for monotone subsequences, that is segments of increasing or decreasing length. The algorithm used for this test keeps track of the number of occurrences (count_r) of subsequences of length r for $1 \leq r \leq 5$ and for length 6 or more. It has been derived in [105] that χ^2 for the run test is given by

$$\chi^2 = \frac{1}{n}\sum_{i=1}^{6}\sum_{j=1}^{6}(\text{count}_i - nb_i)(\text{count}_j - nb_j)A_{ij} \tag{6.10}$$

where n is the length of the sequence and

$$A_{ij} = \frac{1}{n}\begin{pmatrix} 4529.9 & 9044.9 & 13568 & 18091 & 22615 & 27892 \\ 9044.9 & 18097 & 27139 & 36187 & 45234 & 55789 \\ 13568 & 27139 & 40721 & 54281 & 67852 & 83685 \\ 18091 & 36187 & 54281 & 72414 & 90470 & 111580 \\ 22615 & 45234 & 67852 & 90470 & 113262 & 139476 \\ 27892 & 55789 & 83685 & 111580 & 139476 & 172860 \end{pmatrix} \tag{6.11}$$

$$b_i = \begin{pmatrix} \frac{1}{6} & \frac{5}{24} & \frac{11}{120} & \frac{19}{720} & \frac{29}{5040} & \frac{1}{840} \end{pmatrix} \tag{6.12}$$

The variable χ^2 should have a chi-square distribution with six degrees of freedom for large values of n. Table 6.4, 6.5 and 6.6 list the computed value of χ^2 and the corresponding probability of chi-square for the proposed RNG, Unix and LFSR respectively give the result of the run test.

Run test			
Samples	n	χ^2	$P(X \geq \chi^2)$
S_1	20000	1.992263	92.04
S_2	10000	1.990413	92.05
S_3	5000	1.988742	92.07
S_4	2000	1.990453	92.06
S_5	1000	1.997267	91.99
S_6	500	1.977693	92.19
S_7	100	1.975825	99.17

Table 6.4: Run test conducted on random samples for the proposed RNG

Run Test			
Samples	n	χ^2	$P(X \geq \chi^2)$
S_1	20000	0.000456	
S_2	10000	0.001004	
S_3	5000	0.002284	greater than
S_4	2000	0.000628	99.99 %
S_5	1000	0.014367	
S_6	500	0.008344	
S_7	100	0.073494	

Table 6.5: Run test conducted on random Unix random numbers

The probability of 99.99 % for the Unix random numbers is too high to be acceptable and hence the Unix RNG may be be deemed to have failed the run test for the data shown above. In a similar vein, the LFSR performs poorly in the run test and this is corroborated by an average probability greater than 98 %. The probability of about 92 % in our case is quite reasonable in contrast to the other two cases.

Run Test			
Samples	n	χ^2	$P(X \geq \chi^2)$
S_1	20000	1.109119	98.11
S_2	10000	0.759162	99.31
S_3	5000	1.109987	98.10
S_4	2000	0.736777	99.36
S_5	1000	0.546102	99.71
S_6	500	0.496879	99.79
S_7	100	0.519099	99.76

Table 6.6: Run test conducted on LFSR

6.3.3 Serial correlation Test

Given a sequence of n random numbers u_i, the serial correlation coefficient which is a measure of the dependency of u_{i+1} on u_i is given by

$$C = \frac{n(u_0u_1 + u_1u_2 + \ldots + u_{n-2}u_{n-1} + u_{n-1}u_0) - (\sum_{i=0}^{n-1} u_i)^2}{n(u_0^2 + u_1^2 + \ldots + u_{n-1}^2) - (\sum_{i=0}^{n-1} u_i)^2} \tag{6.13}$$

If the correlation coefficient is zero or very small, then the numbers of the random sequence can be deemed to be independent of each other. On the other hand a value of ± 1 implies a high degree of linear dependence. The rule of thumb is that a good value of C should lie between $\mu_n - 2\sigma_n$ and $\mu_n + 2\sigma_n$ where

$$\mu_n = -\frac{1}{n-1} \tag{6.14}$$

$$\sigma_n = \frac{1}{n-1}\sqrt{\frac{n(n-3)}{n+1}} \tag{6.15}$$

The formula for the mean and standard deviation given in eqns(6.14) and (6.15) respectively have been derived based on the assumption that the sequence

Serial Correlation Test				
Samples	n	S.Correlation	$\mu_n - 2\sigma_n$	$\mu_n + 2\sigma_n$
S_1	100	-0.129784	-0.2081	0.1879
S_2	200	-0.126856	-0.1457	0.1357
S_3	400	-0.136762	-0.1023	0.0973
S_4	500	-0.139379	-0.0913	0.0873
S_5	1000	-0.144409	-0.0642	0.0622

Table 6.7: Serial correlation test on random samples in the proposed RNG

Serial Correlation Test				
Samples	n	S.Correlation	$\mu_n - 2\sigma_n$	$\mu_n + 2\sigma_n$
S_1	100	-0.060389	-0.2081	0.1879
S_2	200	-0.032111	-0.1457	0.1357
S_3	400	-0.049737	-0.1023	0.0973
S_4	500	-0.034569	-0.0913	0.0873
S_5	1000	-0.010722	-0.0642	0.0622

Table 6.8: Serial correlation test on Unix random numbers

Serial Correlation Test				
Samples	n	S.Correlation	$\mu_n - 2\sigma_n$	$\mu_n + 2\sigma_n$
S_1	100	0.427389	-0.2081	0.1879
S_2	200	0.437589	-0.1457	0.1357
S_3	400	0.496211	-0.1023	0.0973
S_4	500	0.507319	-0.0913	0.0873
S_5	1000	0.595832	-0.0642	0.0622

Table 6.9: Serial correlation test on LFSR

has a *normal* distribution. Table 6.7 gives the correlation coefficient for several samples along with the expected ranges. The statistics for the test on Unix random numbers and LFSR are given in Tables 6.8 and 6.9 respectively. The test suggests that the correlation coefficient for the Unix random numbers is closer to zero and lies within the required range as compared to our RNG. The behaviour of the LFSR is bad in the sense that the the correlation coefficient permeates far beyond the expected range. This clearly underlines the high degree of correlation existing between the numbers of the sequence.

6.3.4 Numerical distribution test

Generation of random observations from a sequence of uniformly distributed random numbers that follow known distributions give a good insight into the statistical properties of the random sequence. These observations are particularly important to problems involving random numbers in simulation and modeling, queueing theory, Monte Carlo techniques and so on. The important probability distributions include *exponential, normal, gamma, chi-square, F-distribution and t-distributions.* It would be pertinent to investigate and compare the distribution of random sequences generated by the PRNG with the standard distributions mentioned above. The exponential distribution function is considered in this analysis as it appears to fit the random numbers generated in our case as shown below.

6.3.4.1 Exponential distribution

The general method is to equate the cumulative distribution function $F(x) = P(X \leq x)$ where X is the random variable, to the random decimal number and solve for x. The cumulative distribution function for the exponential

distribution is given by

$$P(X \leq x) = 1 - e^{-\alpha x}, x \geq 0 \tag{6.16}$$

where $1/\alpha$ is the mean of the distribution.

In the *crude Monte Carlo* technique, random observations are obtained from the distribution under consideration and then the average of these observations is computed to estimate the mean. The random observations corresponding to an exponential distribution is given by

$$x_i = -\log(1 - u_i) \text{ for } i = 1, 2, \ldots n \tag{6.17}$$

with $\alpha = 1$. The *Complementary random numbers* method is an improved version of the crude Monte Carlo technique in that the random observations are made on both u_i and its complement $1 - u_i$ and the average is computed.

	Mean Random Observation x	
Samples	Crude Monte Carlo Method	Complementary Random Draw Method
S_1	0.984129	1.006529
S_2	0.994417	0.999321
S_3	0.993098	1.000400
S_4	0.992890	1.017303
S_5	0.992560	1.000665

Table 6.10: Random observation of samples of 1000 random numbers obtained from the proposed RNG using Monte Carlo technique with expected mean = 1

Samples of 1000 random numbers were used in the Monte Carlo and complementary random number methods and the computed mean of the observations for different samples are listed in Table 6.10. The test was repeated on the random samples obtained from Unix and LFSR. The results are shown in Tables 6.11 and 6.12.

	Mean Random Observation x	
Samples	Crude Monte Carlo Method	Complementary Random Draw Method
S_1	1.021756	1.014808
S_2	0.983135	1.010201
S_3	1.001775	0.984368
S_4	1.058347	1.004911
S_5	0.944015	1.009685

Table 6.11: Random observation of samples of 1000 random numbers using Monte Carlo techniques for Unix random numbers, expected mean = 1

	Mean Random Observation x	
Samples	Crude Monte Carlo Method	Complementary Random Draw Method
S_1	0.883411	0.997462
S_2	1.0771890	0.991352
S_3	1.019583	0.998667
S_4	0.998035	1.003655
S_5	1.037759	1.002267

Table 6.12: Random observation of samples of 1000 random numbers using Monte Carlo techniques for LFSR, expected mean = 1

It is worth mentioning here that the sample averages are closer to the actual mean of 1.0, for the proposed RNG and that of Unix, separated by a distance smaller than $1/\sqrt{n} = 1/\sqrt{1000} = 0.0316$ which is the standard deviation of the distribution at hand. From the tables it may be concluded that the mean random observation corresponding to an exponential distribution is excellent in both cases. However for the case of the LFSR, the crude Monte Carlo method gives results which are beyond the expected deviation. From the results of the above tests, we may draw a line on the performance of the various RNGs con-

sidered in this analysis and rank them in order of decreasing credibility, namely Unix RNG, the proposed RNG and finally the LFSR. The merits of the Unix random numbers stem from the fact that the RNG used is the multiplicative congruential generator with a period of $2^{31} - 1$.

6.4 Implementation of the Mixed Congruential RNG

While considering the implementation issues of the RNG, it is important to validate and ascertain the randomness of the numbers produced. From the results of the various tests conducted on the random number generator it can be inferred that the quality of the random numbers produced is reasonable as compared to other generators like the Unix MCRNG. Since the RNG is to be implemented in VLSI as one of the constituent modules of the Hypercycle router, area and speed of the circuit warrant serious considerations in designing the layout. A survey was made on the pros and cons of the various types of RNG including the linear congruential, multiplicative congruential, LFSR and other types of generators for implementation. Despite the high quality random numbers produced by the MCRNG, the silicon real estate needed for its implementation is large and far from the prescribed limits. Furthermore, multiplication imposes a bottleneck on the throughput of the circuit. The LFSRs occupy less area but are of inferior quality as demonstrated in the tests discussed previously.

A prototype of a 16-bit mixed congruential random number generator of the form $r_{n+1} = r_{n-1} + c \bmod m$ has been implemented in VLSI in 3μ CMOS3DLM technology with $m = 2^{16} - 1$. The block diagram for the proposed RNG is given in Fig. 6.4.

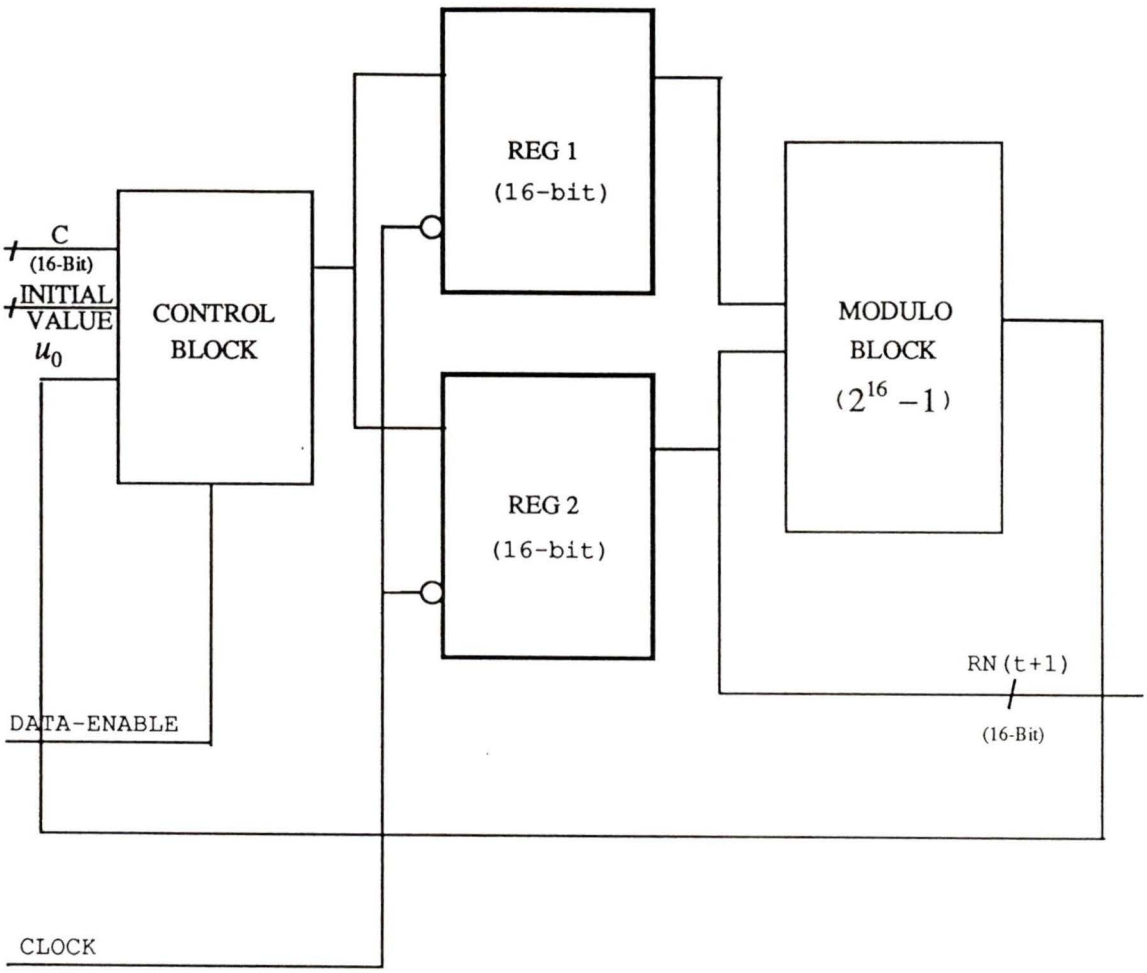


Figure 6.4: Block diagram of Random Number Generator

As can be seen, there are two 16-bit registers for storing the values of c and the initial value (seed) of the RNG. The two words are clocked into the respective registers by asserting the DATA-ENABLE control pin. The registers are enabled at the negative edge of the clock. After the data settles in the registers, they are fed into the modulo unit which adds the two numbers and evaluates modulo $2^{16}-1$. It is worth mentioning here that the design of modulo

$2^{16} - 1$ is based on eqn.(5.8) with $k = 16$ as explained in chapter 5 on residue arithmetic. The modulo block consists of two high speed 16-bit carry-select adders constructed using 8-bit CLAs. Decoding logic for the detection of zero condition is also incorporated.

The output of the modulo block constitutes the next random number in the sequence and is strobed into register 2 during the next clock pulse.

Random Number Generator)	
Technology	CMOS3DLM
Area of the chip	5169.2x5169.2 $\mu^2 m$
Area of the core	4305.2x4305.2 $\mu^2 m$
No. of pins	52
No. of standard cells	1764
No. of transistors	7146
Propagation delay (worst case)	100ns
Area*time	1.853e09 ns$\mu^2 m$
Packaging	68-Pin PGA

Table 6.13: Chip statistics of the implemented RNG

The DATA-ENABLE pin is made low during normal operation of the chip. The block diagrams of the various components of the RNG is illustrated in Appendix C. The SILOS simulation results of the RNG for the 3 μ technology is shown in Fig. 6.6 and this corresponds to an initial value of 1 for the seed while $c = 16807$. The register QN illustrated in Fig. 6.6 latches the next random number of the sequence while register QQ is used for storing the coefficient c. The speed of operation of the circuit is dictated by the propagation delay of the modulo unit and the registers and this sets the upper bound on the clock frequency. Simulation results of the RNG show a worst case propagation delay of 100ns or equivalently 10 Mhz is the maximum frequency of operation. The

MCRNG	
Gate	Count
Inverter	455
Buffer	246
Nand2	738
Nand3	90
Nand4	40
Nor2	108
Nor3	36
Nor4	19

Table 6.14: Gate level statistics of the implemented RNG

schematic capture of the above circuit was made using the Cadence design tool. The VLSI layout of the RNG is shown in Fig. 6.5. The chip consists of a total of 52 pins with 48 data pins, 2 power pins and one clock and control pins. The implementation details of the RNG are summarized in Tables 6.13 and 6.14.

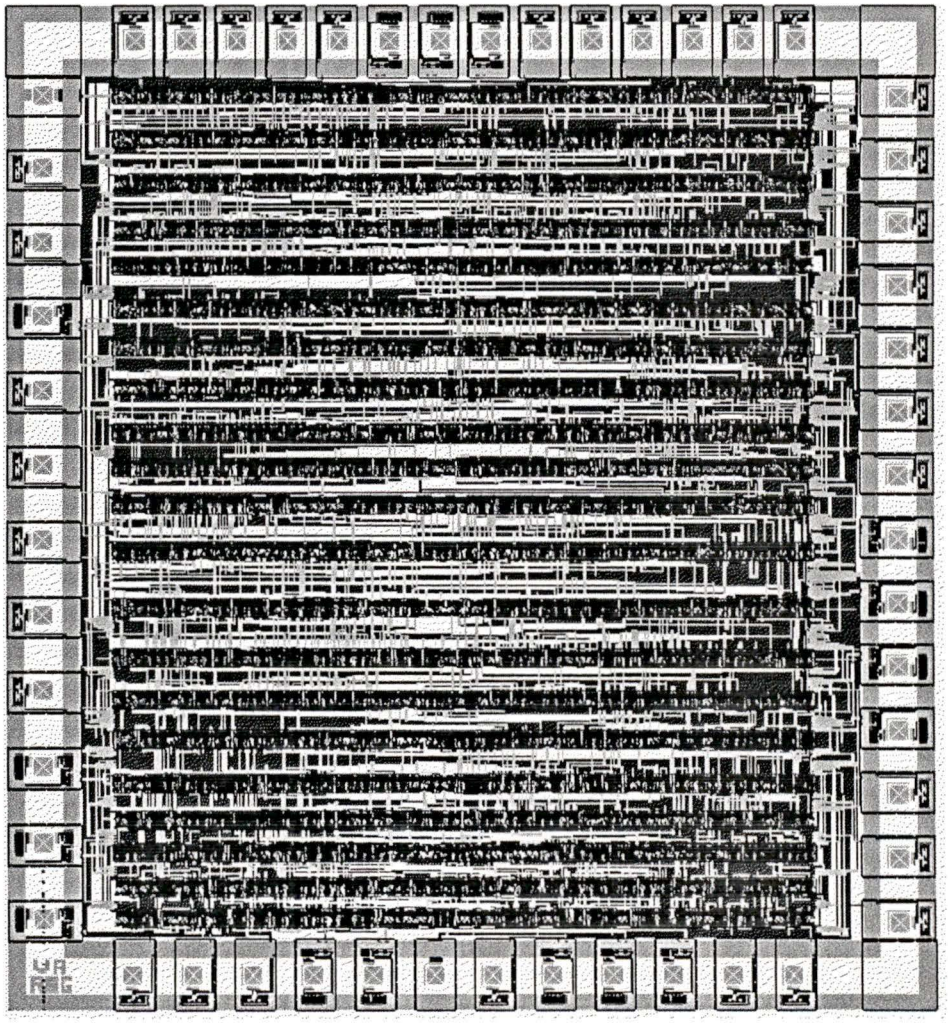
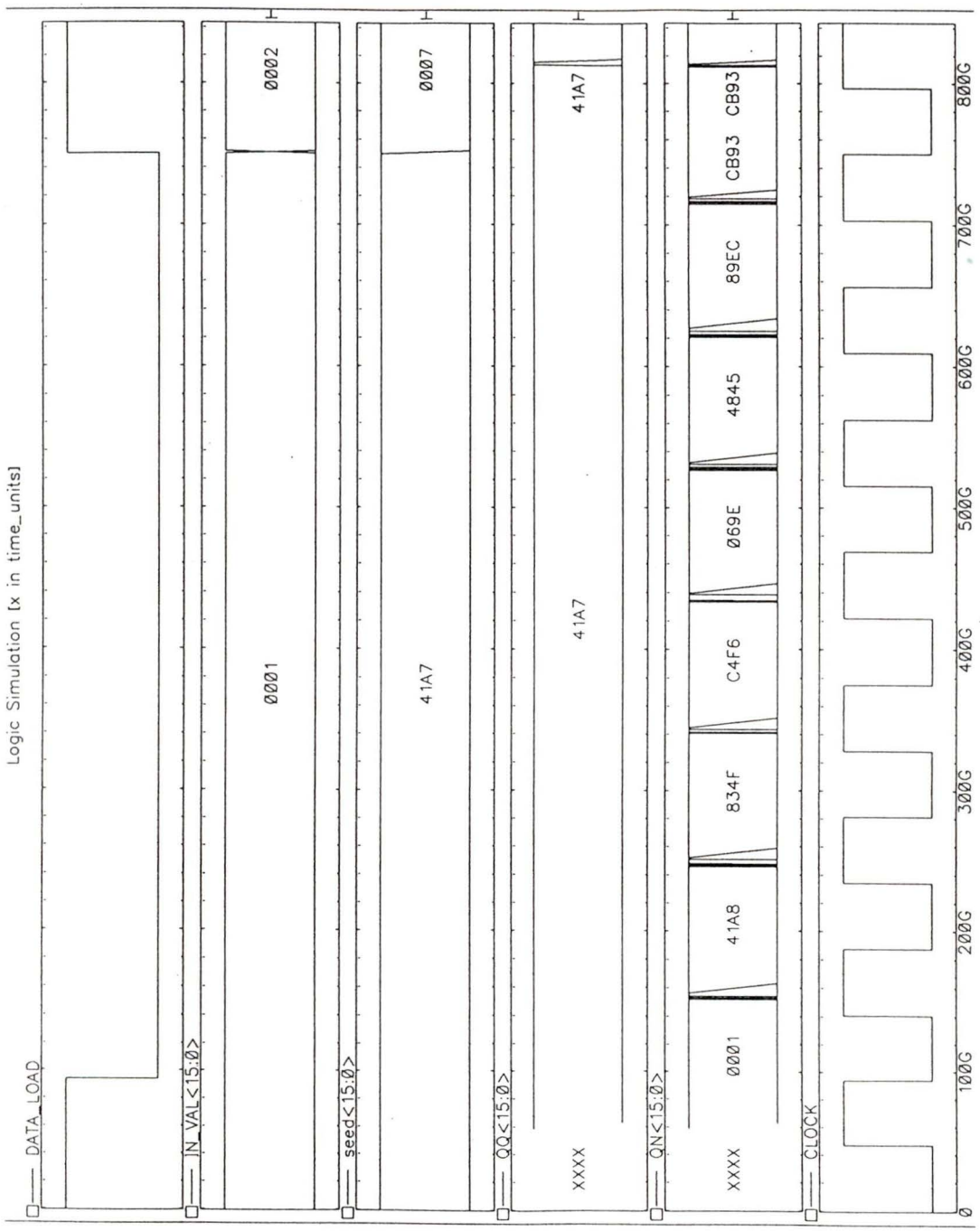
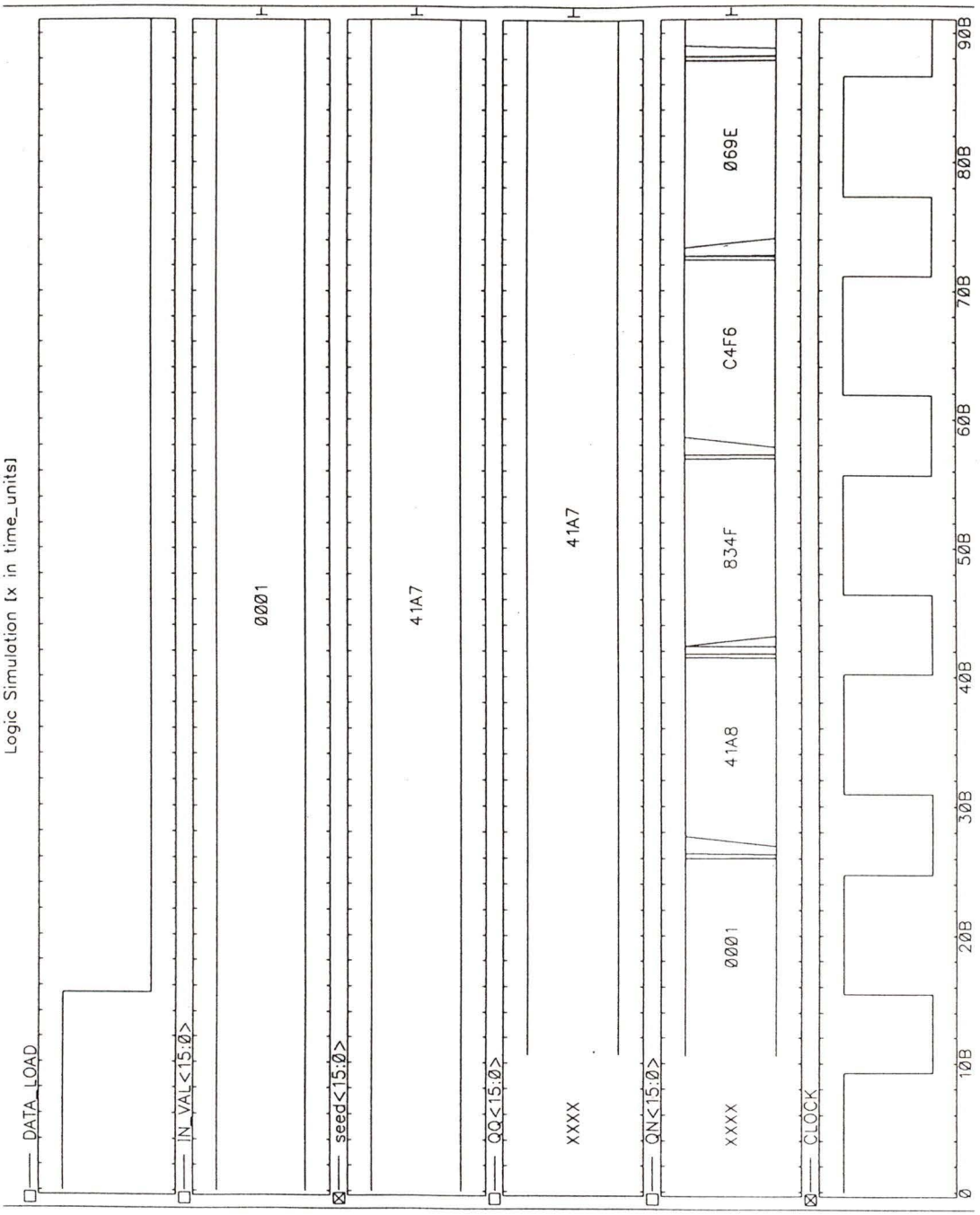


Figure 6.5: VLSI layout of the implemented RNG (3 μ CMOS3DLM)



$G = 1ns$

Figure 6.6: SILOS simulation of RNG (3 μ CMOS3DLM)



$B = 1ns$

Figure 6.7: SILOS simulation of RNG (1.2 μ CMOS4S)

MCRNG	
Technology	1.2 μ CMOS4S
Area of the core	2136.00 x 2136.00 μ^2m
Propagation delay (worst case)	15ns
Area*Time	6.8437e07 ns $\mu^2 m$

Table 6.15: Statistics of the RNG in CMOS4S

6.5 Conclusion

The performance of the RNG in terms of area and speed was studied by simulating the circuit in CMOS4S technology and the statistics for the same are furnished in Table 6.15. The SILOS simulation results of the RNG for the 1.2 μ technology is given in Fig. 6.7. It is interesting to observe that the circuit generates random numbers at a clock period of 15ns. Further more the ratio between the area-time product of the 3 μ and 1.2μ technology is approximately 27:1. This dramatic scaling in area-time made possible by the technological strides in VLSI has given a cutting edge to the proposed RNG in that the maximum operating frequency is about 66 Mhz. It goes without saying that the high throughput of the RNG as evidenced by the performance results, is ideally suited for the router chip incorporated in the 1.2 μ CMOS4S technology.

In this chapter, we analyzed various types of random generators for the router from the performance-cost point of view. A mixed congruential RNG was proposed and it was tested statistically and the performance as compared to other generators was satisfactory. The above RNG is used in the design of the Port Selector for the Hypercycle router.

Chapter 7

Design of the Line Selector Circuit

7.1 Introduction

In the Port Selector module of the Hypercycle router, the output from the RNG-modulo extractor block determines the next random port to be selected from the validated ports as computed by the NPG and PV. Since in our design the HCR can route messages to any of the four dimensions and as the maximum number of available ports is 16, the Port Selector should be capable of selecting randomly one out of the four possible next ports from the 16 available ports. The problem of line selection illustrated in Fig. 7.1 can be formulated as follows.

Given n lines, each of which can assume a logical '1' or '0', there are a maximum of r lines that can be set to '1' with $r < n$. It is necessary to select the xth line which is a '1' such that $0 \leq x \leq r$. We shall call x, the order of the line to be selected. The line selector circuit has widespread applications in sorting and searching, switching theory, and coding theory [112]-[113]. The line selector considered here is similar to the minimum-comparison selection discussed in [112] which selects the tth largest from a set of n elements. While

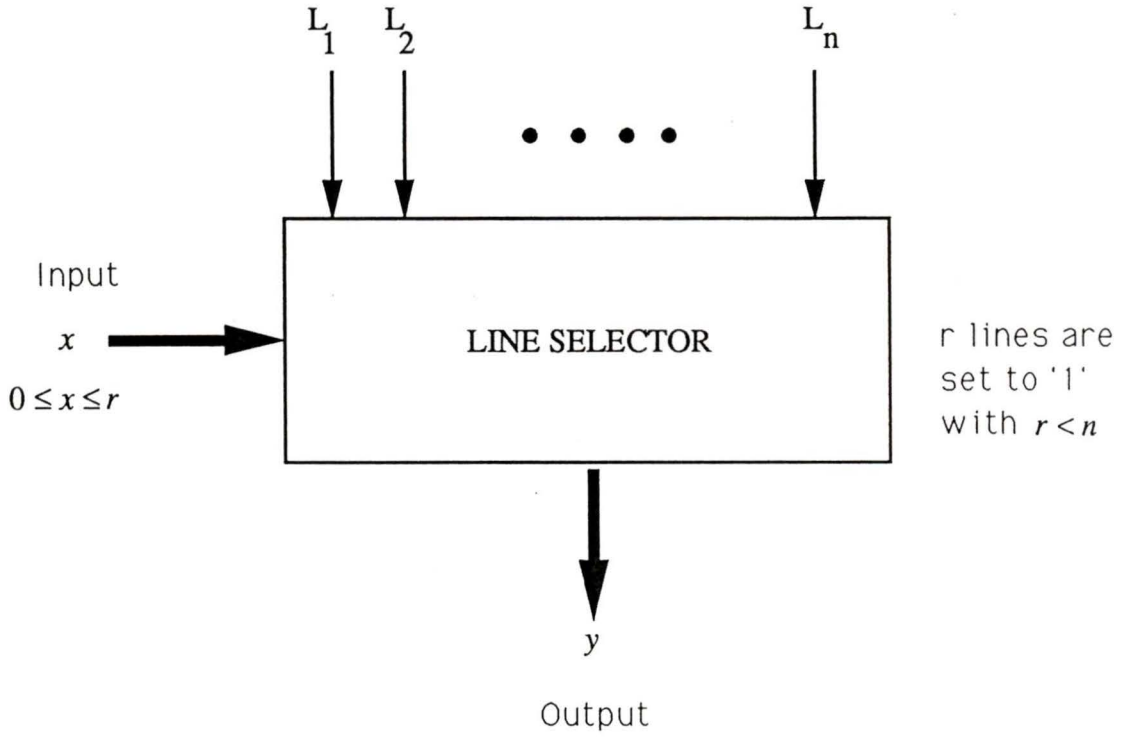


Figure 7.1: Block diagram of Line Selector Circuit

addressing the implementation issues of the line selector, it is important to devise an algorithm which is not only efficient but also reduces the hardware cost, size, power requirements and improves the throughput of the system. In this chapter, a *binary search* method for implementing the line selector for the PS is discussed. The area and time complexities of the proposed design are also derived and its hardware implementation is discussed.

7.2 Binary Search Method

The basic tenet of the binary search method is the partition of the problem into two or more sub problems of lesser complexity than the original one and to recursively apply the same technique till the solution is found. We shall assume that the number of lines n is a power of two for the sake of simplicity. The algorithm proceeds as follows:

The n lines are divided into two groups of n/2 lines each. The number of lines in the first group having the value of '1' is determined and this is compared to the requested order x. If this value is less than or equal to x, it implies that the sought line is indeed located within the first group of n/2 lines. Otherwise, it is present in the second group of n/2 lines. The search has therefore been narrowed to a particular group of n/2 lines and the algorithm repeats itself with the order x being the original order if the sought line is in the first group, or the difference between the original order x and the number of ones encountered in the first group, otherwise. The algorithm terminates after $\log_2 n$ steps at which point, the number of lines in each group has been reduced to exactly one.

7.3 Complexity Calculations

While considering the limiting cases of the above algorithm it is worth mentioning here that for $r = 1$, the problem is reduced to a simple encoder. On the other hand, for the case $r = n$, the sought line is given by x itself. The structure of the line selector for the configuration $n = 16, r = 4$ is shown in Fig. 7.2. The circuit comprises $\log_2 n$ levels which is equivalent to the number of steps in the algorithm. The first level incorporates a single n/2 bit ACS unit comprising of an adder, comparator and subtractor while the k^{th} level incorporates

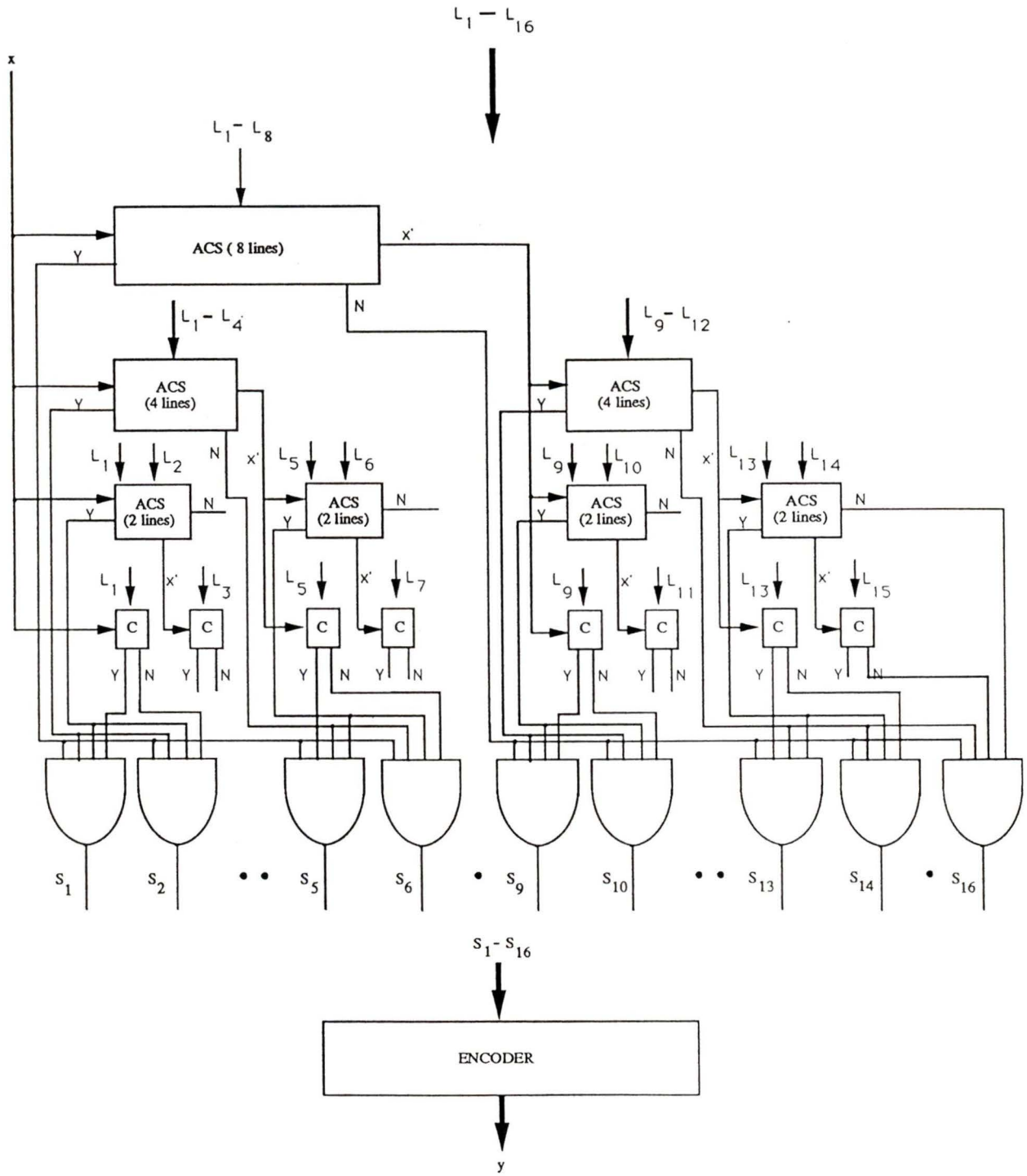


Figure 7.2: Logical diagram of Line Selector for $n = 16$, $r = 4$ using binary search algorithm

2^{k-1} $n/2^k$ bit ACS units. The ACS units are used for calculating the difference between x and the number of *ones* encountered thus far. As one proceeds down the hierarchy, the complexity of these circuits decreases logarithmically.

In establishing the area and time complexity of the proposed line selector, the following assumptions are made regarding the complexity of the used ACS blocks. Given that the ACS blocks incorporate an adder, subtractor and a comparator, they can be constructed using the *straight-forward* approach or *divide-and-conquer* approach as given in [93]. Thus the complexities of the ACS blocks are;

Straight-forward approach in constructing an ACS block:

An m-bit ACS unit is defined to have an area $\alpha_s(m) = m * \alpha(1) = \mathcal{O}(m)$ where $\alpha(1)$ is the area occupied by the unit block. Similarly the time complexity is given by $t_s(m) = m * t(1) = \mathcal{O}(m)$ where $t(1)$ is the time taken by the unit block.

Divide-and-conquer approach in constructing an ACS block:

The area complexity is given by $\alpha_d(m) = m * log_2 m * \alpha(1) = \mathcal{O}(m log_2 m)$ corresponding to the product of the width and height of the tree while the time complexity is given by $t_d(m) = log_2 m * t(1) = \mathcal{O}(log_2 m)$.

7.3.1 Area complexity

The area complexity, A_s, of the proposed line selector using the straight-forward approach in constructing the ACS blocks is given by

$$\begin{aligned} A_s &= \sum_{k=1}^{\log_2 n} 2^{k-1} \alpha_s(\frac{n}{2^k}) \\ &= (\frac{n}{2}) \log_2 n \ \alpha(1) \end{aligned}$$

$$= \mathcal{O}(n \log_2 n) \tag{7.1}$$

The area complexity, A_d, using the divide-and-conquer approach in constructing the ACS blocks is given by

$$\begin{aligned} A_d &= \sum_{k=1}^{\log_2 n} 2^{k-1} \alpha_d(\frac{n}{2^k}) \\ &= \sum_{k=1}^{\log_2 n} 2^{k-1} \frac{n}{2^k} \log_2(\frac{n}{2^k})\ \alpha(1) \\ &= \sum_{k=1}^{\log_2 n} \frac{n}{2} \log_2(\frac{n}{2^k})\ \alpha(1) \\ &= \mathcal{O}(n(\log_2 n)^2) \end{aligned} \tag{7.2}$$

7.3.2 Time complexity

The time complexity, τ_s, of the proposed line selector using the straight-forward approach in constructing the ACS blocks is given by

$$\begin{aligned} \tau_s &= \sum_{k=1}^{\log_2 n} t_s(\frac{n}{2^k}) \\ &= \sum_{k=1}^{\log_2 n} (\frac{n}{2^k})\ t(1) \\ &= n\frac{\frac{1}{2}(1 - \frac{1}{2^{\log n}})}{(1 - \frac{1}{2})}\ t(1) \\ &= n(\frac{n-1}{n})\ t(1) \\ &= (n-1)\ t(1) \\ &= \mathcal{O}(n) \end{aligned} \tag{7.3}$$

The time complexity τ_d using the divide-and-conquer approach in constructing the ACS blocks is given by

$$\begin{aligned}
\tau_d &= \sum_{k=1}^{\log_2 n} t_d(\frac{n}{2^k}) \\
&= \sum_{k=1}^{\log_2 n} \log_2(\frac{n}{2^k})\ t(1) \\
&= (\sum_{k=1}^{\log_2 n} \log_2 n - \sum_{k=1}^{\log_2 n} k)\ t(1) \\
&= ((\log_2 n)^2 - \frac{\log_2 n(log_2 n + 1)}{2})\ t(1) \\
&= (\frac{1}{2}(\log_2 n)^2 - \frac{1}{2}\log_2 n)\ t(1) \\
&= \mathcal{O}((\log_2 n)^2) \qquad (7.4)
\end{aligned}$$

In addition, one can calculate the number of ACS units $U(n)$ used in our circuit as well as an upper bound of the delay $\tau(n)$ in terms of the maximum delay within the ACS units as $U(n) = \sum_{k=1}^{\log_2 n} 2^{k-1} = n - 1$ and $\tau(n) = \log_2 n$. These complexities compare well with the complexities cited in [114] for a selector network which moves the t largest of n distinct inputs into t specific outputs where they are allowed to appear in any order. The asymptotic behavior for the number of comparators $\hat{U}_t(n)$ in that selector network is given as $\hat{U}_t(n) = n\lfloor\log_2(t+1)\rfloor + \mathcal{O}((\log n)^{\lfloor\log_2 t\rfloor})$ while the minimum delay time is $\hat{\tau} = \log_2 n + \lfloor\log_2 t\rfloor \log\log n + \mathcal{O}(\log\log\log n)$.

7.4 Implementation

The reduced hardware complexity coupled with the attendant gains in regularity and expandability makes the binary search algorithm ideally suited for VLSI implementation. The line selector circuit discussed in this thesis has been implemented in VLSI in 1.2 μ CMOS technology. The ACS units have been constructed in a hierarchical manner using the divide-and-conquer approach discussed previously.

Each ACS block outputs two control signals, named Y and N, which determine whether the location of the line searched is within the group of lines feeding the ACS unit or not. If the line is present in the group, then $Y = 1; N = 0$ otherwise the sought line is not in the group and $Y = 0; N = 1$. A series of 1-bit comparators is present at the leaves of the decision tree. The corresponding minterms of the Y, N signals are passed through a set of AND gates to get the outputs $S_1 - S_{16}$. Only one of the outputs, S_i, is asserted for a given set of inputs and this indicates the position of the selected line i. Simulation results of the line selector have yielded a worst case propagation delay of less than 15 ns.

7.5 Conclusion

The line selector circuit discussed in this chapter for the configuration corresponding to $n = 16$ and $r = 4$ has been used in the design of the Port Validator block of the Hypercycle router.

Chapter 8

Implementation of Hypercycle Router

8.1 Introduction

Having explained the structure of Hypercycle graphs in Chapter 2, the *Backtrack-to-the-Origin-Retry-routing* scheme, system architecture and sub-components of the router in Chapters 3, 4, 5, 6 and 7, we are now in a position to integrate the sub-blocks and build the Hypercycle router. In this chapter, we focus primarily on the implementation of the router in the 1.2 μ CMOS4S technology and discuss in depth the organization of the router, circuit description, timing diagrams, simulation results and implementation details of the Hypercycle router.

8.2 Organization

The Hypercycle router considered for implementation was designed with the following assumptions:

1. The maximum dimension r of the graph was taken to be 4.
2. The maximum number of available ports was taken to be 16.
3. Each NPG can have a maximum population of 15 nodes.

These assumptions were made after careful consideration of the available silicon area and hardware complexity. The organization of data, control and computing blocks of the 16-port Hypercycle router is illustrated in Fig. 8.1.

8.2.1 Data Path

The router requires 8 pieces of information for its proper operation. They include:

1. Connectivity vector
2. Source or current address
3. Destination address
4. Population
5. Offset of previous dimensions
6. Available ports
7. Random number coefficient
8. Initial value of random number generator.

The connectivity vector ρ_i, current address ξ_i, destination address β_i, population M_i and offset PP_i are required by each NPG in the i^{th} dimension. Each of the above inputs in dimension i is 4-bit in length except ρ_i which is a 3-bit number. Since there are 4 dimensions, 4 modules of NPG are utilized and for the sake of uniformity, the above data operands have been set to 16 bits. The number of available ports is a 16-bit operand with each bit corresponding to a port. In addition, two 16-bit data are required for the initial value and seed of the random number generator in the Port Selector.

To determine the I/O count of the chip, the eight 16-bit operands require 128 pins. The high I/O pin count may be infeasible and inefficient to realize and this necessitates multiplexing of data. With 3 control pins, the eight operands can be multiplexed into the 8 registers of the router as seen in Fig. 8.1. This effects a drastic reduction in the pin count and is amenable for implementation in standard IC packages. The degradation in performance is minimal since parameters such as connectivity, current address, population, offset, and RNG initial value need to be input only once during startup while only the destination address needs to be updated every time a new message needs to be routed. The assignment of internal registers to the input data are shown in Table 8.1. It is worth mentioning here that register No. 7 is a special purpose register in that it has been designed such that it can store the next random number generated internally within the Port Selector during each cycle, provided the data for initializing the RNG is not loaded into it externally. The output of the router consists of

1. The next port address
2. Destination reached signal
3. Break signal.

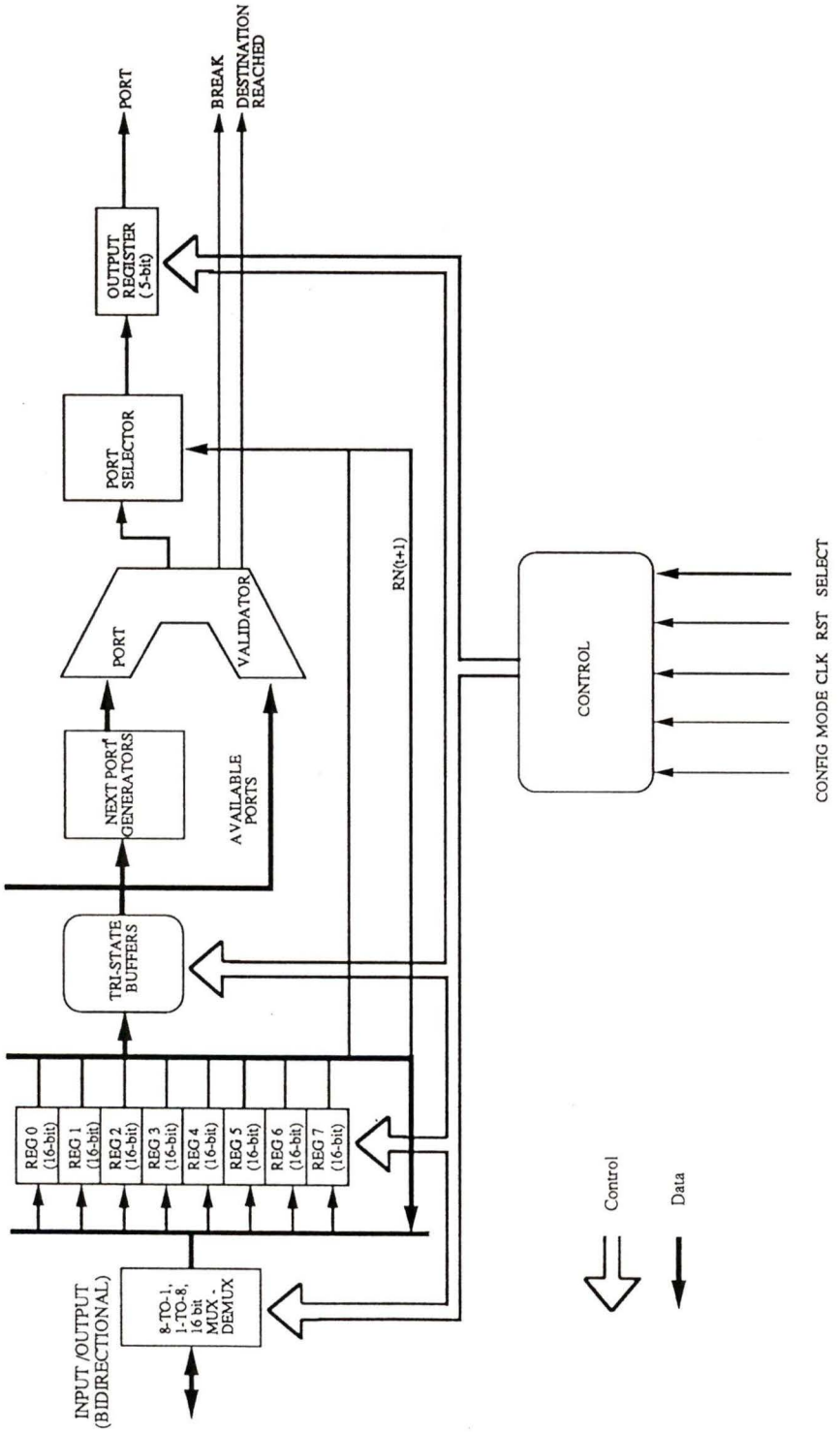


Figure 8.1: Logical organization of Hypercycle Router

Operand	*Notation*	*RegisterNo.*$(16-bit)$
Connectivity	$\rho = \rho_1\rho_2\rho_3$	0
Source address	$\xi = \xi_1\xi_2\xi_3\xi_4$	1
Destination address	$\beta = \beta_1\beta_2\beta_3\beta_4$	2
Population	$M = m_1m_2m_3m_4$	3
Previous Port (Offset)	$PP = PP_1PP_2PP_3PP_4$	4
Available ports	AP	5
Random number coefficient	coefficient	6
RNG initial value	IN-VAL	7

Table 8.1: Input registers of the Hypercycle router

The next port address is a 5-bit data since there are 16 ports numbered 1 through 16 and it is latched in a 5-bit register at the end of each decision cycle which will be used by the controller to switch the cross-bar. The destination reached signal or the break signal identify the corresponding condition of the routing process. The controller can then initiate a strategy specific action. For instance in the BTOR scheme, upon receiving a break signal, the partially completed path is dissolved.

8.2.2 Computing Blocks

We have discussed at length the architecture of the NPG, Port Selector and the Port Validator in the previous chapters including their sub-components such as random number generator, modulo-extractor and line selector circuits. They have been integrated into the router with little or no modifications to suit the general architecture. Four modules of the 15-node NPG are used corresponding to 4 dimensions. Also the modulo extractor has been designed to compute X mod $1 \ldots 4$ and X is the 16-bit random number generated internally by the mixed congruential random number generator. Tristate buffers have been in-

corporated to provide isolation between the registers and computing blocks and to prevent spurious output during the load cycle.

8.2.3 Data Signals

The data signals of the router consist of a 16 bit bidirectional data bus (I/O $\langle 15:0 \rangle$). The output signals from the router include PORT $\langle 4:0 \rangle$, which constitutes the next port address (5 bits), destination reached signal (DTR, 1-bit) and the break signal (BK, 1-bit).

8.2.4 Control Signals

The essential control signals for the router consist of the following:

Configuration (CONFIG, 1-bit): This controls the operation of the router. Typically it identifies the router in two phases, either the Load phase (corresponding to a logical '0') or the execute phase (corresponding to logical '1'). The configuration control signal is generated by the controller. More details regarding the operating modes of the router will be explained in the subsequent sections.

Mode (MODE, 1-bit): The Mode control can be used to access the internal registers of the router. When it is set in the *write* mode (corresponding to logical '0', data is written into the internal register as determined by the select lines of the multiplexor. When MODE is equal to logical '1', the router is set in the *read* mode and the internal states of the registers can be scanned corresponding to the selected register. The testability feature incorporated in the router is the *Random Access Scan* path which will be discussed in the next chapter.

Select (A,B,C, 3-bits): The lines A, B and C select the appropriate register to be accessed.

Clock (CLK, 1-bit): The clock pin is used to strobe in data into the registers when the router is in the load phase with mode control equal to 0 (write). The data is latched during the positive edge of the clock pulse.

Reset (RST, 1-bit): When the reset pulse is active high, all the registers of the router including the output are reset. For normal operation, the reset pin is made low.

The basic control signals for accessing the I/O registers in the router are given in Table 8.2.

8.3 Operating Modes

Each decision cycle of the Hypercycle router is composed of two phases, namely

1. *Load phase*
2. *Execute phase.*

In the *Load phase*, the router is initialized with data needed for its operation during system configuration. In this phase, the controller loads the router with information such as the current address, destination address, population, connectivity, offset, RNG coefficient and initial values of the RNG. Note that the above data is needed only once during configuration. For normal and subsequent operations only the destination address is required.

In the *Execute phase*, the data loaded into the router is processed and the resulting next port address (if an available one is found) together with the

Phase	Configuration	Mode	Select	Operation	Register
	CONFIG	MODE	ABC		
Load	0	0/1	000	write/read	Reg 0
	0	0/1	001	write/read	Reg 1
	0	0/1	010	write/read	Reg 2
	0	0/1	011	write/read	Reg 3
	0	0/1	100	write/read	Reg 4
	0	0/1	101	write/read	Reg 5
	0	0/1	110	write/read	Reg 6
	0	0/1	111	write/read	Reg 7
Execute	1	1	000	read	Reg 0
	1	1	001	read	Reg 1
	1	1	010	read	Reg 2
	1	1	011	read	Reg 3
	1	1	100	read	Reg 4
	1	1	101	read	Reg 5
	1	1	110	read	Reg 6
	1	1	111	read	Reg 7

Table 8.2: Register access in Hypercycle Router chip

destination reached or possible break are then presented on the output pins PORT$\langle 4:0 \rangle$, DTR, BK respectively.

The total execution time for each decision is therefore dictated by the time taken for the load cycle and the execute cycle. Note that the load cycle time is maximum during system power up or when the configuration of the computer network is altered. Under normal conditions, configuration of the network happens only during power up when the necessary information is loaded into the router. We shall use the worst case delay for the load cycle as the one arising from the loading of a new destination address. The worst case propagation delay of the router is given by the sum of the worst case delays of the load and

Simulation Data for Hypercycle Router		
Operand Type		Data(Hex)
Input	Connectivity	2222
	Source	1111
	Destination	3333
	Population	4444
	Offset	0369
	Available Port	FFFF
	Seed	41A7
	Initial Value	0001
Output (Expected)	Port	05
	DTR	0
	BK	0

Table 8.3: Logic simulation data for Hypercycle Router

execute phase. The *throughput* (TP) of the router is defined as the number of decision cycles executed per second and this is given by

$$TP^w = \frac{1}{T_l^w + T_e^w} \quad \text{operations/sec} \tag{8.1}$$

where T_l^w and T_e^w are the worst case propagation delays of the router during the load and execute phases of each decision cycle.

The timing diagrams of the router in both the normal and register read modes are illustrated in Fig. 8.2 and 8.3. A snapshot of the simulation results of the router is shown in Fig. 8.4 for the vectors given in Table 8.3. Input data is loaded from time $T = 0$ to $T = 900$ ns into the various registers of the router as explained previously. At $T = 950$ ns, the next port address (ph_port) is set to 05(Hex) when configuration control (CON) transits from logical '1' to '0'. The DTR and BK signal settle at logical '0' at 890ns and 920 ns respectively.

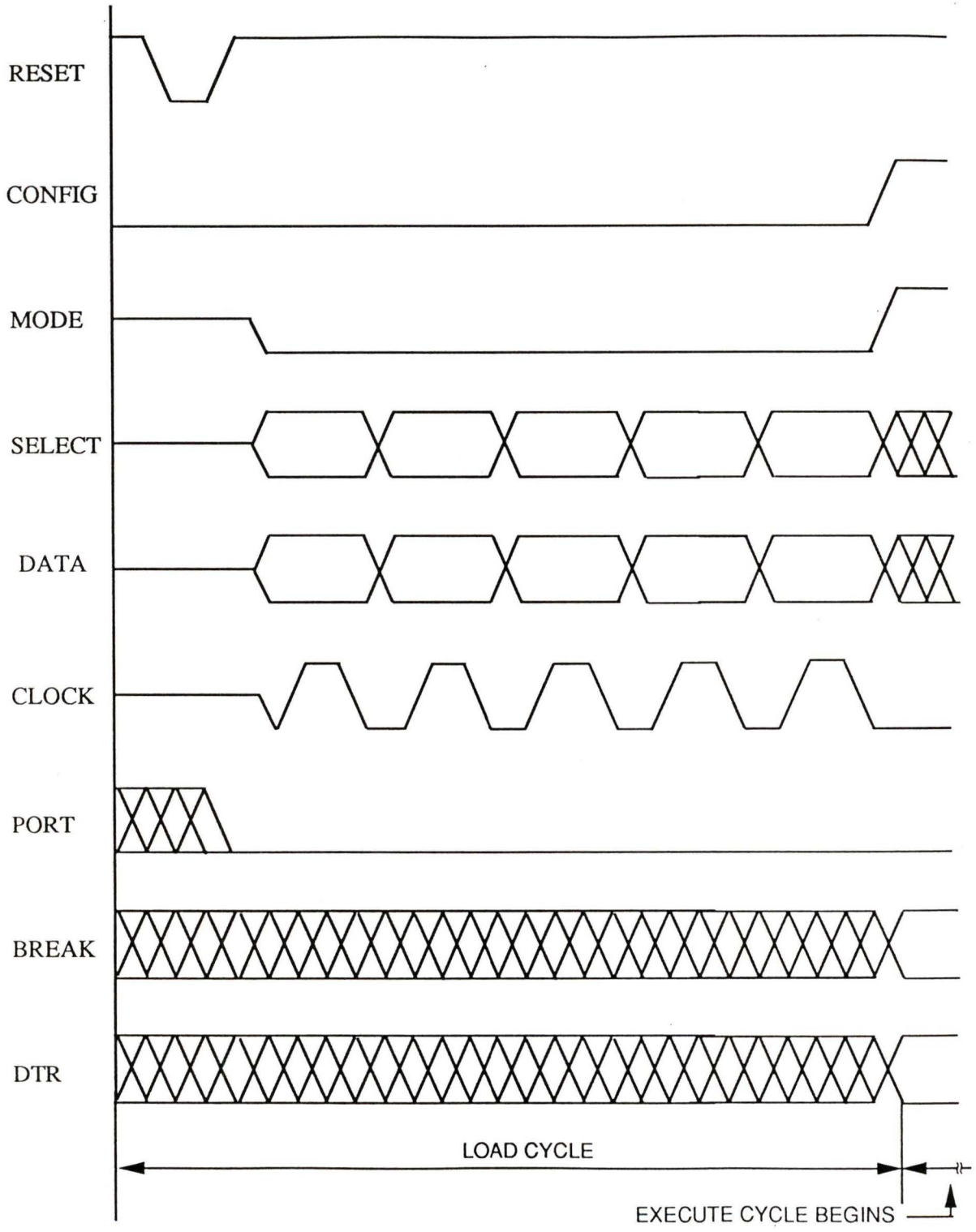


Figure 8.2: Timing diagram of the Hypercycle router (normal mode)

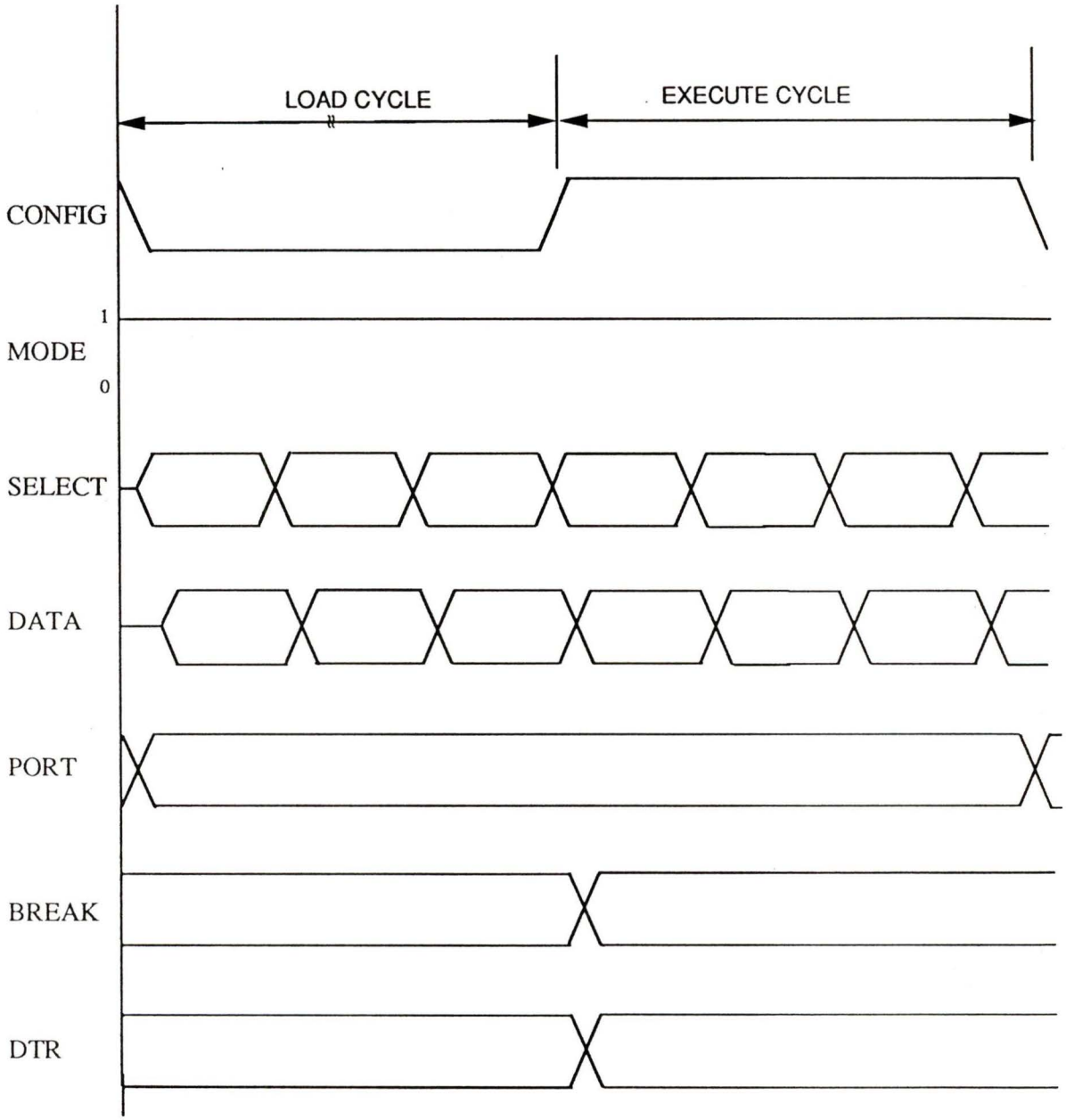


Figure 8.3: Timing diagram of the Hypercycle router (Read mode)

$B = 1ns$

Figure 8.4: SILOS Simulation of Hypercycle router

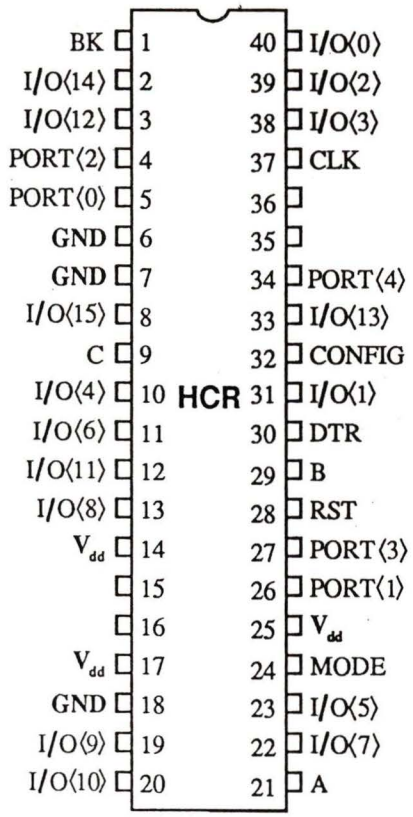


Figure 8.5: Pin layout of the Hypercycle Router (40-Pin DIP)

8.4 Pin Configuration

The Hypercycle router is housed in a 40-pin DIP utilizing 36 pins. Of the 36 pins, 23 are data pins, 7 are control pins and 6 of them are power pins. The pin layout of the router is show in Fig.(8.5). The bonding diagram of the Hypercycle router fabricated by CMC as a multi-project-chip is shown in Appendix D along with the schematic diagrams.

Hypercycle Router	
Technology	1.2μ NTE CMOS4S
Area of the chip	5263.60x5263.60 $\mu^2 m$
Area of the core	4581.90x4581.90 $\mu^2 m$
No. of pins	36
No. of standard cells	5951
No. of transistors	25842
Propagation delay	50ns
Area*time	1.0497e09 ns$\mu^2 m$

Table 8.4: Chip statistics of the implemented Hypercycle Router

8.5 VLSI Implementation

The router was implemented in the 1.2 μ CMOS4S technology using the semi custom approach with the help of the Cadence tool and the CMOS4S standard cell library [73]. As explained previously, the I/O pin count of the entire system was carefully studied and it was optimized to fit in a 40 Pin DIP. The general guidelines for its implementation suggested in [73, 75] were followed. As many as 6 power pads were incorporated in the design to account for metal migration and noise spikes. Of these, 2 pairs of power pads were exclusively used for the core area. Power bus width was set to 15 DSM and the methodology of tree distribution of drivers was followed. All registers in the circuit are built using D-type Master slave flipflops to avoid race conditions. Bidirectional pins played a key role in reducing the I/O count. The micrograph of the Hypercycle router is shown in Fig. 8.6. The statistics of the chip are given in Tables 8.4 and 8.5.

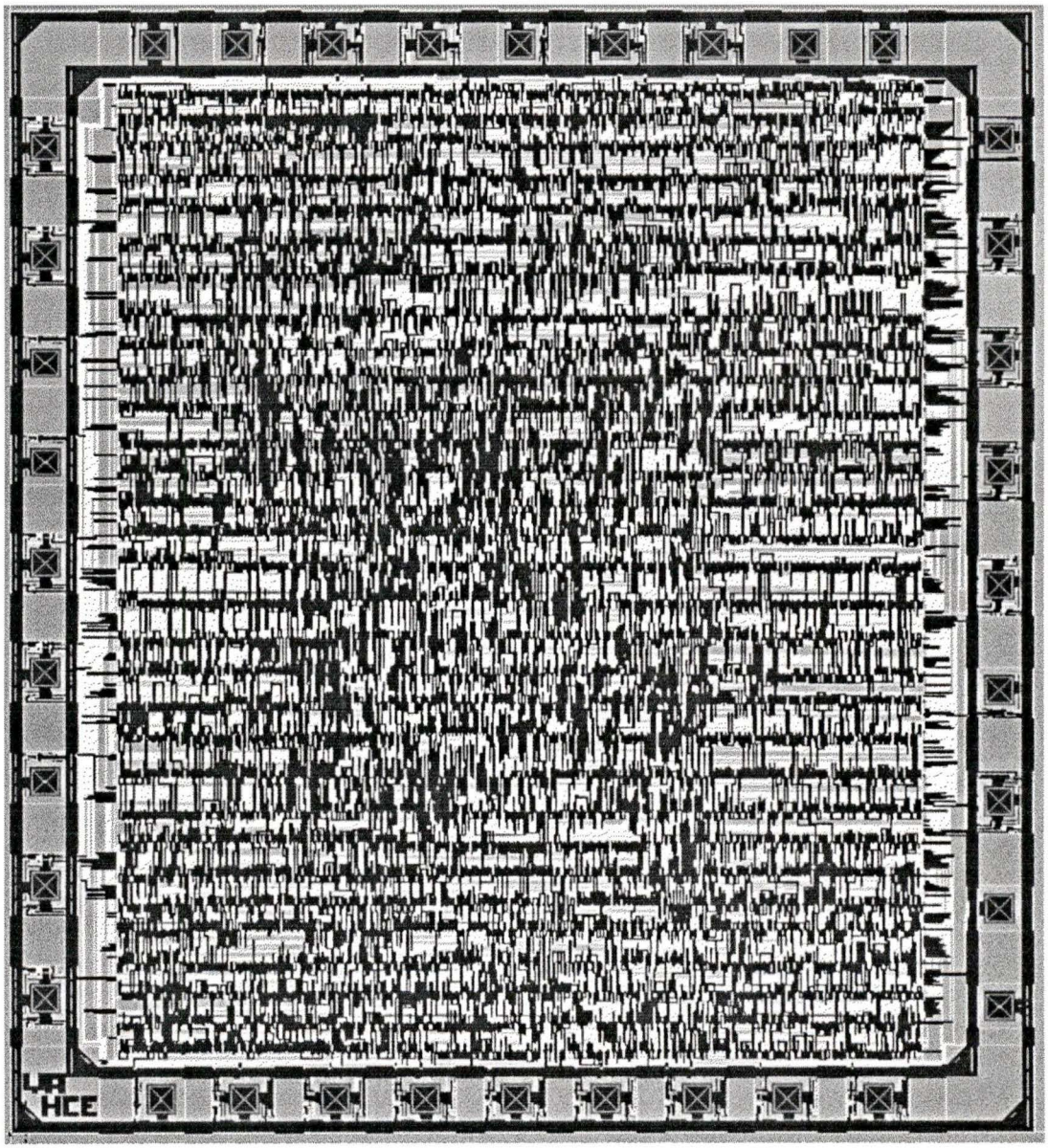


Figure 8.6: Micrograph of the Hypercycle Router (1.2 μ CMOS4S)

Hypercycle Router	
Gate	Count
Inverters	1844
Buffers	779
Tri-Buffers	96
D-FlipFlops	135
Nand2	1704
Nand3	372
Nand4	373
Nor2	425
Nor3	108
Nor4	115

Table 8.5: Gate level statistics of the implemented Hypercycle Router

8.6 Conclusion

In this chapter, the implementation of the Hypercycle router which was the primary objective of this thesis was discussed. The configuration chosen for implementation in 1.2 μ CMOS4S technology was a 4-dimension, 16-port, 15 node per dimension Hypercycle router. In the next chapter, we consider the testing and design verification of the implemented chips.

Chapter 9

Testing and Verification

9.1 Introduction

The progress in contemporary IC technology has brought about a quantum leap in integration, higher speed, reduced area and a drastic reduction in the cost of chips. But the problem of determining in a cost effective way, whether the component or chip has been manufactured correctly and free from defects still remains an enigma to the research community [118].

The process of testing and design verification is therefore an important facet of the IC design cycle in that it attempts to determine whether the prototype is fault free, whether it is consistent and meets its expected goals, whether it delivers the required performance under actual field conditions and whether or not it can be committed to large scale manufacturing.

The IC being designed may be an ASIC such as a gate array, PLA, standard cell or a full custom design. In all these design methodologies, the design verification falls under two categories. The first is the preliminary verification performed through CAE simulation tools while the second phase is the hardware verification process during which the designer determines whether the prototypes meet the design requirements.

The prototypes are usually verified by inserting them into the target board or by using a test fixture to analyze the device. Testing of ICs should be stringent to the maximum possible extent during hardware verification.

The inclusion of testability circuits can significantly affect the cost of the product because of increased area and propagation delays but it greatly simplifies the testing process and hence the overall testing/validation costs. It is difficult to quantify the cost of testing since it spans over a wide variety of issues including test generation, test evaluation and test application. Designing VLSI circuits for testability is the most efficient way to reduce testing costs and assuring high chip reliability. *Design-for-testability* (DFT) is beneficial in that it reduces the overall design cycle time and test costs without sacrificing the quality of the product and leads to higher productivity. The general methods include scan design, partitioning of chips, random and built-in-self-test methods. In this chapter, we detail the testing and testability aspects of the implemented chips besides giving details on their performance, yield and AC/DC characterization.

9.2 Testing

The increased complexity of VLSI has made testing an important and often vital phase of the design cycle. Rising costs of chips run counter to the reduction in cost of designing and producing chips. Testing can be defined as the process of searching a circuit for faults that may have been caused by design errors, fabrication errors or due to external phenomena [119].

Testing has two major facets, namely

1. Test generation

2. Test verification.

Test generation is the process of enumerating stimuli for a circuit which will demonstrate its correct operation. Test verification is the process of validating/proving that the set of tests are effective to this end. It is a widely accepted fact that cost of the test generation and fault simulation is proportional to the cube of the number of gates in the chip [118]. In other words

$$T = kN^3 \tag{9.1}$$

where T is the computer run time, N is the number of gates in the chip and k is a proportionality constant. The testing of a digital system or an IC is quintessential for achieving high reliability and maintainability of the device [119]. The testing process requires understanding the basic design of the circuit, test objectives, test methodologies, the software involved in generating the test vectors and last but not the least the test equipment that is used to verify the chips.

9.2.1 Test Objectives

One of the simplest methods of testing a combinational logic circuit is to exhaustively test the circuit by applying all possible input combinations. This is feasible only for combinational circuits with a small number of inputs.

One need not elaborate the difficulty of testing sequential circuits since it is necessary to validate the output besides the state transition for each input vector. This is compounded by the fact that there is no closed formula for describing the patterns and length of an exhaustive test for a sequential circuit.

It has been shown in [121] that for a sequential circuit with M latches, an upper bound on the number of test patterns is given by $(N-1)N^N$ where $N \leq 2^M$. The circuit must therefore be tested for all possible inputs and initial states.

Scan designs are being increasingly used for testing sequential circuits. In this method, the feedback loops of the registers are broken so that it is possible to control and observe the inputs and outputs of the combinational logic within the sequential circuit. Scan designs make the generation of tests for sequential circuits easier and greatly reduce the number of transitions in the state transition table that must be verified.

9.2.2 Fault Models

Fault models are necessary to establish and study the effects of faults in the circuit. This model can be used to simulate the circuit and determine how the circuit behaves in the presence of faults. There is an endless list of the various types of faults that occur in an IC. These include component defects, wafer defects, stuck-at-faults, stuck-open-faults, bridging faults, transient faults and so on. The single stuck-at-model is the most commonly used method for modeling failures in logic circuits. It is assumed that the fault in the circuit will cause one of the lines to be permanently stuck-at-1 or 0.

9.2.3 Fault Simulation

Fault simulation determines the behaviour of the circuit for each of the different faults. Due to the presence of a sizable number of potential faults in any large circuit, it is usually assumed that only a single fault is present in the circuit at any given time. A typical procedure for developing test data is to fault simulate the circuit for a particular fault model, compute the fault coverage and generate corresponding test patterns for the tester.

9.3 Test Pattern Generation

Test generation techniques attempt to systematically derive a set of test vectors that exposes possible faults in the logic circuit. There are several algorithms for generating test vectors for combinational circuits and these include the Boolean Difference method, literal propositions, D-algorithm [125] and Path Oriented Decision Making algorithms (PODEM) [126]. PODEM is perhaps the most widely algorithm on account of its efficiency and speed over D-algorithm. They use the principle of *path sensitization* in which a path is selected from the site of a potential fault to the primary output and the fault is propagated along (forward trace). The primary signals and hence the test vectors that produce the signal values produced during the forward drive is calculated and this phase is called the *Backward Drive.*

9.3.1 CATPAG

CATPAG [127] is an automatic test pattern generator that generates vectors for combinational logic circuits using the PODEM algorithm. The vectors generated by CATPAG are selected to detect single stuck-at-faults in the circuit under test. This package is interfaced to the Cadence design tool and can accept SILOS netlist descriptions of the circuit. The user can specify the various parameters that control the functioning of CATPAG such as specifying the percentage of fault coverage, the amount of backtrace and memory. The output of CATPAG consists of the actual fault coverage achieved and a set of test vectors that can be used for testing the IC. If the acquired response differs from the expected response, it can be concluded that the circuit is faulty.

9.3.2 Test Application

Test application is the process of applying test vectors and measuring the responses of the Device-under-test (DUT) using sophisticated instruments called Automatic Test Equipment (ATE) which play a key role in the testing of ICs. The ATE used for testing the implemented chips is the IMS XL-60 tester. We briefly run through the characteristics of this equipment in the next section.

9.3.3 IMS XL-60

The IMS XL-60 [128] is a fully integrated test system that has complete capability for characterizing and testing a device. It has provisions for testing chips up to 244 pins and can handle a wide variety of chip packages such as DIP, PGA, PLCC. The highlights of the tester are the following.

1. It is a fully integrated test system with complete capabilities for verification and characterization of the device.
2. It can measure data rates up to 60 Mhz.
3. The built in command language interface and pattern conversion can be readily used to set the operating conditions and for downloading test vectors from any simulator.
4. It can perform AC analysis including the measurement of propagation delays, set-up and hold times.
5. The DC parametric analyzer is useful for characterizing the device under different loads and for measurement of voltage/current levels.

One obvious limitation of the IMS XL-60 tester is its low test vector depth. The tester can handle only 16K vectors at a time.

Chip Name	CMC Code	No. of Devices	Package	Technology	Fabrication Time
NPG	IC3VAHCR	5	40-Pin DIP	CMOS3DLM	6 months
MOD	IC3VAMOD	10	68-Pin PGA	"	"
RNG	IC3VARNG	10	"	"	"
HCR	ICAVAHCE	5	40-Pin DIP	CMOS4S	5 months

Table 9.1: Statistics of the fabricated chips

9.4 Fabrication

The Next Port Generator, Modulo-Extractor and Random Number generator designed in CMOS3DLM technology were accepted for fabrication by the CMC on September 17^{th}, 1990 its multiproject chip run MPC9004C3. The chips were expected to be delivered on December 11^{th}, 1990 but unforeseen fabrication delays postponed the shipping date. The chips were finally delivered on March 19^{th}, 1991. The Hypercycle router chip designed in CMOS4S technology was accepted by CMC on November 19^{th}, 1990 run MPC9003CA and were delivered on April 14^{th}, 1991. The number of chips fabricated by CMC for the various designs is shown in Table 9.1.

9.5 Test results of Next Port Generator

As was explained in Chapter 4, the NPG is a 546-cell CMOS combinational logic design that performs the routing operations of the Hypercycle networks. It consists of 19 inputs and 5 outputs. An exhaustive test for the NPG can be ruled out on account of the overhead involved in testing all 2^{19} combinations. To make the testing process more tractable, functional, fault and parametric

testing were done on the fabricated chips.

9.5.1 Functional Testing

The primary aim of functional testing is to ascertain and validate that the DUT is correct functionally and free from any errors. CATPAG was used to generate test vectors and these were verified by simulation. A C program was written that simulates the function of the Hypercycle router and generates the next port address. The simulator selectively removed the improbable cases which include the following:

1. The population $M = 0$
2. The connectivity $RHO = 0$ or $RHO > \lceil \frac{M}{2} \rceil$
3. The source address $XI > M$
4. The destination address $BETA > M$.

The number of test vectors obtained were 12,182 vectors. These patterns were applied to each of the 5 chips using the IMS XL-60 tester and the expected response was compared with the acquired response. The result of the functional test performed on the NPGs is shown in Table 9.2.

It can be inferred from functional testing that 3 out of 5 chips perform the intended functions as per design while 2 of them failed.

9.5.2 Fault Testing

CATPAG was used to generate test patterns for a single stuck-at-fault model and this yielded a fault coverage of approximately 78 % with 50,000 vectors. The chips were tested once again with these patterns and any mismatch will

Functional Test Summary - NPG		
No. of vectors - 12,182		
Chip No.	Comparison errors	Result
1	0	Pass
2	0	Pass
3	9253	Fail
4	8702	Fail
5	0	Pass

Table 9.2: Functional test results of the Next Port Generator

Stuck-at-fault Test Summary - NPG		
No. of vectors - 50,000		Fault Coverage - 78 %
Chip No.	Comparison errors	Result
1	0	Pass
2	0	Pass
3	42,468	Fail
4	40,156	Fail
5	0	Pass

Table 9.3: Stuck-at-fault test results of Next Port Generator

flag the presence of a stuck-at-fault. The results of the fault test are listed in Table 9.3.

As can be seen, the 3 chips (No. 1,2, & 5) which passed the functional test also passed the stuck-at-fault test and hence it may be concluded that the chip is defect free from stuck-at-faults for the given fault coverage.

Parametric Test Summary - NPG					
Measured values (Worst case)					
Chip No.	Propagation delay	V_{OH}	V_{OL}	I_{iL}	I_{iH}
1	137.5 ns	4.48 V	0.875 V	4na	8na
2	131.0 ns	4.50 V	0.836 V	4na	8na
5	127.2 ns	4.46 V	0.821 V	4na	8na

Table 9.4: Parametric test results of Next Port Generator

9.5.3 Parametric Testing

Parametric testing is concerned with ensuring that a fabricated structure achieves the specified performance in terms of time and accuracy. For the NPG, performance parameters were checked by running an AC, DC analysis and validating them within the tolerant limits of the device. The propagation delay of the circuit was measured by reducing the sampling time of the output to the point where the device fails. The voltage and current levels of the I/O pins were measured and these were found to be lie within the required ranges. Table 9.4 gives the parametric test summary of the 3 working chips.

Simulation results of the NPG yielded a worst case propagation delay of approximately 125ns. The measured delay is greater than this value and this can be attributed to the fact that simulation did not take into account the I/O pad delay and the parasitic resistances, capacitances of the interconnect layers.

The fault coverage could not be improved beyond this point due to the presence of redundant faults in the design. The failed chips were observed under the microscope and the following observations came to light.

- Scratches in the metallization were noted.
- Dust particles or specks were observed on top of the p-transistors causing

short circuits.

- A large region of transistors in one section of the core were affected possibly due to manufacturing flaws/wafer defects and the p-diffusion and Metal 1 appeared corrupted.

The chip yield in our case is defined as the ratio of the number of good chips to the total number of fabricate chips. The yield of the NPG chips is 60 %.

9.6 Testing of Hypercycle Router Chip

The increased complexity of contemporary VLSI chips provides the basis for inclusion of special hardware circuits for improving the testability of the chip. This approach termed as Design for testability (DFT) is not only restricted to the external pins but also to the internal nodes of the chip. DFT can be categorized into two parts;

1. *Ad hoc Testing*

2. *Structured Testing.*

Ad hoc method consists of partitioning the chip into several logical modules that can be easily tested. Extra pins or probe points may be added to increase the controllability and observability of the circuit.

Structured techniques were developed to introduce standardization in testing. They usually involve a set of design rules by which designs are implemented. The objective of the structured approach is to reduce the sequential complexity of the network and aid in test generation and verification by improving the controllability and observability of the circuit. The commonly used approaches

for structured testing include Built-In-Self Test (BIST), Level Sensitive Scan Design, Scan Path, Scan/Set Logic, Random Access Scan [63] and so on.

9.7 Random Access Scan

The Random Access Scan (RAS) technique is analogous to the LSSD in that it provides the ability to completely control and observe the registers of the circuit and thereby the sequential logic test problem is converted to a combinational one [123]. Each register of the circuit can be individually accessed through a decoder. Once a register is selected, data can be scanned in or out and hence the states of the circuit can be controlled and observed.

The RAS method has been incorporated in the design of the 1.2 micron Hypercycle router chip explained in Chapter 8. As can be seen from Fig. 8.1 in Chapter 8, each one of the eight 16-bit registers can be accessed through the bidirectional bus by setting the appropriate *SELECT* signals. The bidirectional pins are controlled by the MODE pin which determines whether the data is an input (MODE=0) or an output (MODE=1). The basic functions of the registers are enumerated below.

1. The RESET pulse clears all the flipflops/registers of the circuit.

2. During *Scan-in*, operands in the DATA bus are clocked into the selected register by setting the appropriate SELECT signal with *MODE* control set to '0' (write).

3. Similarly for *Scan-out*, the selected register is read out by setting *MODE* pin to '1' (read).

4. The 5-bit output register which holds the next port address can be observed at any time.

5. The internal states of the random number generator can be observed in register 7 since the successive random numbers are clocked into the same during each execution cycle.

The timing diagram for the test operation is given in Fig. 8.3. Logical partitioning of chip is achieved in step 5 mentioned above since the RNG can be isolated from the rest of the circuit and its performance can be monitored separately. In our implementation of the RAS, the bidirectional pins have been effectively and efficiently used in reading/writing data to the router through the 16-bit parallel-in/parallel-out registers. This approach has reduced the pin count substantially besides increasing the speed of testing since the data can be accessed in parallel.

9.7.1 Random Testing

Random Testing is a probabilistic method in which the test patterns for the DUT are selected at random in the hope that if the device is faulty, one of the selected vectors will sensitize the fault thereby resulting in an erroneous response at the output. The disadvantage of random testing is that it is difficult to ascertain the percentage of fault coverage and this is usually done with the help of a fault simulator. The sequential nature of the Hypercycle router can be fractured by considering the combinational blocks, namely, the NPG, PV and PS separately. The problem therefore reduces to that of generating vectors that will detect faults in the above combinational blocks [129].

The vectors generated by CATPAG for the NPG were utilized in testing the Hypercycle router. A simulator for the router was coded in C and it was used to validate the vectors chosen at random. The patterns so generated were manually edited so that they sensitized all possible paths and execute the

Random Testing - HCR		
No. of vectors - 100		
Chip No.	Comparison errors	Result
1	0	Pass
2	78	Fail
3	65	Fail
4	0	Pass
5	81	Fail

Table 9.5: Random Testing of the Hypercycle Router

AC Test Summary - HCR				
Chip No.	Set-Up Time	Hold Time	Load cycle	Execute cycle
1	3.9 ns	2.0ns	5.9ns	38ns
4	4.1 ns	2.0ns	6.1ns	40ns

Table 9.6: Measured propagation delays of Hypercycle Router

various functionalities of the chip. The results of random testing are listed in Table 9.5. The state of the RNG was observed for several cycles to verify that their functional correctness. The number of test vectors used was about 100. It should be mentioned that the test results for the router shown here are only preliminary and are not exhaustive.

The two working chips were subjected to high speed testing. The results are shown in Table 9.6.

The yield of the Hypercycle router chips is 40 %. The fault coverage of the Hypercycle router could not be computed due to the nonavailability of sophisticated fault simulators/testability analyzers.

9.8 Conclusion

The challenge of testing VLSI circuits is becoming more formidable with tremendous increase in gate-count, pin-count, smaller feature size, higher performance and higher complexity all contributing to the surmounting testability problem. Test generation and test evaluation become cumbersome and at times computationally infeasible.

In this chapter, testing and verification of the NPG and Hypercycle Router chips were considered. The test results that we have at hand are quite encouraging in that the yield of the chips is about 50 %. In essence, we have proved through testing and hardware verification that the implemented chips indeed perform their functions as per specifications for the test patterns presented.

It is natural to inquire about the test results of the other two chips, namely the modulo extractor and RNG. Due to the nonavailability of a 68-pin PGA socket for the IMS, the testing of these chips has been postponed till resources become available. The only conjecture, we can make at this point is that the yield of the chips will be similar to that of the NPG considering the fact that the NTE's CMOS3DLM process is fairly well standardized and established. over the years.

Furthermore, the Hypercycle router needs to be tested in a comprehensive manner for more patterns since the test results stated here are only preliminary. Unforeseen hardware failures in the IMS test equipment that occurred on April 30^{th}, 1991 have stalled the testing process. This has been augmented by the non-functionality of the SILOS fault simulators, testability and SCOAP analyzers of the Cadence design tool and therefore an idea of the approximate fault coverage could not be obtained. Intensive testing of the router chip besides the untested ones will be taken up in the future.

Chapter 10

Conclusion

10.1 Summary

A very important consideration in the design of distributed computer systems is the interconnection network. A plethora of interconnection networks have been proposed in the past for multi-computer systems. In this thesis, our attention was focussed on a novel class of multidimensional graphs called Hypercycles which are amenable for designing interconnection networks for concurrent computers. The Hypercycles exhibit a regular, cyclic and expandable topology and are generalization of the r-ary, n-cube networks.

Hypercycles combine the best features of the expandable r-ary n-cube and the compact n-dimensional Hypercube structures. The regular topology permits the implementation of simple routing algorithms like the BTOR scheme considered here. Hypercycle graphs can be used to synthesize networks that grow incrementally. All nodes have the same degree regardless of the size of the graph. This makes them attractive for a wide spectrum of applications. They show great promise for the future in the design of interconnection networks compared to binary n-cube in light of their expandability, fault tolerance capability and topological flexibility.

10.2 Synopsis

The heart of this thesis is the hardware realization of the Hypercycle router for the BTOR scheme. The Hypercycle Router was implemented in 1.2 μ CMOS4S technology and the entire IC design cycle from concept, theory, specification, design, implementation and testing was traversed. To complement and validate the design, prototypes of the sub-modules of the router namely, the Next Port Generator, Modulo-Extractor and Random Number Generator were implemented in 3 μ CMOS3DLM technology.

In Chapter 1, the evolution of parallel computers was traced and the need for efficient interconnection networks for multiprocessing system was discussed.

In Chapter 2, the basic definitions and terminologies of Hypercycle graphs were introduced. The characteristics of the Hypercycles were described in depth.

In Chapter 3, the BTOR scheme was discussed and the architecture of Hypercycle router engine was formulated. In Chapters 4, 5 & 6, we considered the VLSI implementation of the NPG, modulo-extractor and 16-bit MCRNG respectively. In Chapter 7, a *binary search* algorithm for implementing the line selector was discussed along with the area-time complexity analysis.

In Chapter 8, the design of the Hypercycle Router was considered and the circuit was implemented in 1.2 μ CMOS4S technology.

Test results of the implemented chips namely, the NPG and HCR were discussed in Chapter 9.

10.3 Discussion

One can address several issues for further improvement. Most importantly, the Hypercycle router can be redesigned using full-custom approach. This will result in increased area-time efficiency compared to the implemented prototype. In this thesis, analytical expressions corresponding to the *greedy strategy* (eqns. 2.18 a, c, e) were used in designing the NPGs. For future work, the *liberal strategy* could be incorporated into the design and an appropriate hardware be synthesized.

Further improvements could be brought about in the design of the random number generator. A 31-bit multiplicative congruential random number generator which has been claimed to be an industry standard generator [110] can be used in the router but at the cost of silicon area and speed. However these demerits could be annulled in full custom design and further scaling in technology. Novel algorithms for multiplication in VLSI and modulo arithmetic principles outlined in Chapter 5 could be combined to produce an efficient generator. Furthermore cellular automata based RNGs should be given due consideration. The test results for the Hypercycle Router are only preliminary and it requires more comprehensive testing and characterization besides the other untested chips as well. These issues will be given due consideration for the future.

The router designed in this thesis can be operated at a frequency of approximately 20 Mhz. The propagation delay of the router can be considered to be insignificant compared to the overall transmission time of messages over the network (at the rate of 1 bit per 100 ns for wire transmission, for instance). The router has been isolated from the controller and other parts of the Hypercycle router engine in view of the fact that it can be used in conjunction with other routing schemes and also from the point of view of available silicon area.

This gives sufficient motive for bringing about various innovations in the router engine. The remainder of the project is the complete implementation of the controller and hardware synthesis of the entire Hypercycle router engine for the BTOR scheme.

10.4 Suggestions for Further Research

The development of high performance Interconnection networks is an evolutionary rather than a revolutionary process. Hypercycles are a new class of graph that have been conceived very recently for designing interconnection networks for distributed computer systems. They have opened new and exciting avenues for further investigation. Routing strategies other than the BTOR scheme considered here such as *random backtracking* which is a cross between the Hyperswitch type of routing and the BTOR scheme, adaptive routing algorithms, virtual channel, virtual networks etc can be analyzed and their performance be validated for the Hypercycle networks.

Fault tolerance in Hypercycles is a broad area that warrants further research. The BTOR scheme in particular can be simulated by injecting faults in the network and studying the performance of the algorithm. Similarly, fault tolerance can be studied for other routing algorithms as well.

Hypercycle graphs can be analyzed from an optimization point of view. Cost-effective design is the hallmark and basic tenet of any system. To increase the throughput of interprocessor communication, the network has to be optimized. Expressions for Hypercycle topology based on average distance, diameter and degree given in Chapter 2 can be optimized using classical minimization algorithms to arrive at an optimal graph structure for the application at hand.

The problem of load balancing and task allocation is yet another area which needs investigation. Task allocation is the method of assigning tasks of a parallel program among the processors of a parallel computer in such a manner that it minimizes the interprocessor communication cost while maintaining the computational load balance among the various processors. Various allocation schemes like *first fit, best fit, cluster partitioning* [130, 131, 132] etc can be mapped on the Hypercycles and a control unit be designed for the same. This hardware can be included as part of the controller circuit in the Hypercycle router engine.

The technological revolution in VLSI and the development of high speed digital computers provides an ideal setting for the practical realization of a Hypercycle based multi-computer system.

Bibliography

[1] Kai Hwang and A. Briggs, *Computer Architecture and Parallel Processing*, McGraw Hill series, 1987.

[2] W.D. Hillis, *The Connection Machine: A Computer Architecture Based on Cellular Automata*, Physica, 1984.

[3] Kai Hwang and Douglas Degroots, *Parallel Processing for Supercomputers and Artificial Intelligence*, McGraw Hill series, 1989.

[4] A. S. Tanenbaum, *Computer Networks*, 2nd edition, Prentice-Hall, New Jersey, 1988.

[5] H. T. Kung, "Why systolic architectures", *IEEE Computer*, No.15, 1982, pp. 37-45.

[6] S. Y. Kung, "VLSI Array Processors", Prentice-Hall, 1988, *IEEE Computer*, No.15, 1982, pp. 37-45.

[7] Olaf M. Lubeck, "Supercomputer Performance: The Theory, Practice and Results" in Marshall C. Yovits (ed.), *Advances in COMPUTERS*, vol. 27, Academic Press, 1988, pp. 315.

[8] Tse-yun Feng, "A Survey of Interconnection Networks," *IEEE Computer,* pp. 12-27, December 1981.

[9] Leonard Uhr, *Multi-Computer Architectures for Artificial Intelligence*, Wiley-Interscience Publication, 1987.

[10] P. H. Enslow, *Multiprocessor and Parallel Processing*, John Wiley, 1974.

[11] T. Lang and H. S. Stone, "A Shuffle exchange network with simplified control", *IEEE Transactions on Computers*, vol. C-25, Jan. 1976, pp. 55-56.

[12] D. Lawrie,"Access and alignment in an array processor", *IEEE Transactions on Computers*, vol. C-24, Dec. 1975, pp. 1145-1155.

[13] J. H. Patel,"Processor-Memory Interconnection for Multiprocessors", *Proc. Sixth Annual Symposium on Computer Architecture*, April 1979, pp. 168-177.

[14] L. Goke and G. Lipovski, " Banyan Networks for partitioning multiprocessor systems", *Proc. 1st Annual Computer Architecture Conference*, 1973, pp. 21-28.

[15] J. P. Hayes, T. N. Mudge, Q. F. Stout et al., "Architecture of a Hypercube Supercomputer", *Proceedings of the 1986 International Conference on Parallel Processing*, 1986, pp. 653-660.

[16] Y. Paker, *Multi-Microprocessor Systems*, Academic Press, 1983.

[17] R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger Ltd, 1981.

[18] A. L. Leiner, W. A. Notz et al., "PILOT - A new multiple computer system", *Journal of ACM*, vol. 6. No. 3, July 1959, pp. 313-335.

[19] G. H. Barnes et al., "The ILLIAC IV computer", *IEEE Transactions on Computers*, vol. C-17, Aug 1968, pp. 144-151.

[20] D. P. Siewiorek, V. Kini, H. Mashburn, S. R. McConnel and M. M. Tsao, "A Case study of C.mmp, C.m* and C.vmp - Part I - Experiences with fault-tolerance in multiprocessor systems", *Proceedings of IEEE*, vol. 66, No. 10, pp. 1178-1199, Oct 1978.

[21] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)", Technical Report CS-79-103, CMU, Pittsburgh, Apr. 1979.

[22] L. D. Wittie, "Communication Structures for Large Networks of Microcomputers", *IEEE Transactions on Computers*, C-30, No. 4, pp. 264-273, April 1981.

[23] M. Weiser etal, *Status and Performance of the ZMOB Parallel Processing System*, Proceedings of COMPCON, Spring 1985.

[24] P.C. Treleaven, *The New Generation of Computer Architecture*, Proceedings of the 10th Annual International Symposium on Computer Architecture, June 1983.

[25] B. Awerbuch, O. Goldreich, "A trade-off between Information and Communication in Broadcast Protocols", *Journal of ACM*, June 1983, pp. 238-256.

[26] *Butterfly Parallel Processor Overview*, BBN Labs., version 1, March 1986.

[27] J. R. Goodman and C. H. Sequin, "Hypertree: A Multiprocessor Interconnection Topology", *IEEE Trans. on Computers*, vol C-30, No. 12, Dec. 1981, pp. 923-933.

[28] K. Hwang and J. Ghosh, "Hypernet: A Communication-Efficient Architecture for constructing Massively Parallel Computers", *IEEE Trans. on Computers*, Dec. 1987, pp. 1450-1466.

[29] J. Rattner, "Concurrent Processing: A New Direction in Scientific Computing", *Proc. of the 1985 National Computer Conference*, AFIPS Press, vol. 54, 1985, pp. 764-771.

[30] C. L. Seitz, "The Cosmic Cube", *Communications of the ACM*, vol. 28, Jan. 1985, pp. 22-33.

[31] M. W. Ferrante, "Cyberplus and Map V Interprocessor Communications for Parallel and Array Processor Systems" in W. J. Karplus (ed), *Multiprocessors and Array Processors*, San Deigo, Jan. 1987, pp. 45-54.

[32] L. N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network", *IEEE Trans. on Computers*, vol. C-33, No.4, April 1984, pp. 323-333.

[33] L. N. Bhuyan and D. P. Agrawal, "Design and Performance of Generalized Interconnection Networks", *IEEE Trans. on Computers*, vol. C-32, No.12, Dec. 1983, pp. 1081-1090.

[34] J. A. Stankovic, "A Perspective on Distributed Computer Systems", *IEEE Trans. on Computers*, vol. C-33, No.12, Dec. 1984, pp. 1102-1115.

[35] M. Imase, T. Soneoko, "Connectivity of Regular Directed Graphs with Small Diameters", *IEEE Trans. on Computers*, vol. C-34, No.3, March 1985, pp. 267-273.

[36] K. Padmanabhan, D. H. Lawrie, "A Class of Redundant Path Multistage Interconnection networks", *IEEE Trans. on Computers*, vol. C-32, No.12, Dec. 1983, pp. 1099-1108.

[37] S.P. Dandamudi, D.L.Eager, *Hierarchical interconnection networks for multiprocessor systems*, IEEE Trans. On Computers, June 1990.

[38] D. P. Bertsekas, J. H. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, New Jersey, 1989.

[39] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, New Jersey, 1974.

[40] J. O. Tuazon, J. C. Patterson et al.,"Caltech/JPL MARK II Hypercube Concurrent Processor", *Proceedings of the 1985 International Conference on Parallel Processing*, Aug 1985, pp. 666-673.

[41] J. C. Patterson, J. O. Tuazon, D. Leiberman et al.," The Mark III Hypercube Ensemble Concurrent computer", *Proceedings of the 1985 International Conference on Parallel Processing*, Aug 1985, pp. 71-73.

[42] R. D. Rasmussen, N. J. Dimopoulos et al.,"Advanced General Purpose Multicomputer for Space Applications", *Proceedings of the 1987 International Conference on Parallel Processing*, Aug 1987, pp. 54-57.

[43] R. D. Rasmussen, N. J. Dimopoulos, G. S. Bolotin et al.,"MAX: Advanced General Purpose Real-Time Multicomputer for Space Applications", *Proceedings of the IEEE Real Time Systems Applications*, Dec 1987, pp. 70-78.

[44] N. J. Dimopoulos, R. D. Rasmussen, G. S. Bolotin et al., "Hypercycles, Interconnection Networks with simple Routing Strategies", *Proceedings of the*

Canadian Conference on Electrical & Computer Engineering, Nov. 1988, pp. 577-580.

[45] Don Radvan, "Performance Evaluation and Router Design for the Backtrack-to-the-origin-and-retry-routing-scheme of the Hypercycle based interconnection networks", M.A.Sc Thesis, University of Victoria, July 1990.

[46] N. J. Dimopoulos, D. Radvan, K. F. Li, " Backtrack-to-the-Origin-and-Retry-routing for Hypercycle based interconnection networks", *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, June 1989, pp. 173-177.

[47] William J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in multiprocessor Interconnection networks", *IEEE Trans. on Computers*, vol. C-36, No. 5, MAY 1987, pp. 547-553.

[48] N. S. Szabo and R. I. Tanaka, Residue Arithmetic and its Applications to Computer technology, McGraw Hill, 1967.

[49] J. M. Mcquillan, "Routing Algorithms for Computer Networks - A Survey", *Proceedings of National Telecommunications Conference*, 1977, pp. 28:1-1 - 28:1-6.

[50] M. Schwartz, *Computer-Communication Networks - Design and Analysis*, Prentice-Hall, New Jersey, 1977.

[51] T. H. Cormen and C. E. Leiserson, "A Hyperconcentrator Switch for Routing Bit-Serial Messages", *Journal of Parallel and Distributed Computing*, vol. 10, Oct 1990, pp.193-194.

[52] William J. Dally, "Performance Analysis of k-ary n-cube Interconnection networks", *IEEE Trans. on Computers*, vol. 39, No. 6, June 1990, pp. 775-785.

[53] S. L. Johnson and C. T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Trans. on Computers*, vol. 38, No. 9, Sept. 1989, pp. 1249-1268.

[54] H. Yoon, K. Y. Lee and M. T. Liu, "Performance Analysis of multibuffered Packet-Switching Networks in Multiprocessor Systems", *IEEE Trans. on Computers*, vol. 39, No. 3, March. 1990, pp. 319-327.

[55] N. F. Maxemchuk and M. Zarki, "Routing and Flow Control in High-Speed Wide Area Networks", *Proceedings of IEEE*, vol. 78, No. 1, Jan. 1990, pp. 204-220.

[56] C. L. Seitz et al., "The Hypercube communication chip", Dept of Computer Science, CalTech, Display File 5182:DF:85, March 1985.

[57] P. Kermani et al., "Virtual cut through: A new computer communication technique", *Computer Networks*, vol. 3, 1979, pp. 267-286.

[58] D. C. Grunwald, D. A. Reed, "Networks for Parallel Processors: Measurements and prognostications", *Journal of ACM*, 1988, pp. 610-619.

[59] J. M. Mcquillan, "Adaptive Routing Algorithms for Distributed Computer Networks", Ph.D. Dissertation, Harvard University, 1974.

[60] H. T. Kung, B. Sproull, G. Steele, *VLSI Systems and Computation*, Computer Science Press, Maryland, 1981.

[61] *Selected Reprints on Logic Design for Testability*, Tutorial, IEEE Computer Society Press, 1984.

[62] R. G. Bennets,*Design of Testable Logic circuits*, Addison-Wesley, Reading-Mass, 1984.

[63] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[64] *CMOS VLSI Design Principles*, VLSI design group, University of Manitoba, August 1990.

[65] Marco Annaratone, *Digital CMOS Circuit Design*, Kluwer Academic Publishers, 1986.

[66] D. A. Pucknell and K. Eshragian, *Basic VLSI Design*, Prentice-Hall, 1985.

[67] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing, 1980.

[68] J. P. Uyemura, *Fundamentals of MOS Digital Integrated Circuits*, Addison-Wesley Publishing, 1988.

[69] A. Mukherjhee, *Introduction to nMOS and CMOS VLSI Systems Design*, Addison-Wesley Publishing, 1980.

[70] M. Shoji, *CMOS Digital Circuit Technology*, Prentice-Hall, 1988.

Queen's University, 1990.

[71] *SPICE User's Manual*, ver 3c1, University of California, Berkeley, 1990.

[72] *The CMOS3DLM Cell library*, VLSI Implementation Centre report, CMC, March 1989.

[73] *The CMOS4S Standard Cell library*, VLSI Implementation Centre report, CMC, March 1990.

[74] *Layout specifications for the 1.2 micron Standard Cell library*, VLSI Implementation Centre report, CMC, April 1990.

[75] Jack Mowchenko, *Designing with CMC CMOS4S Standard Cell library*, 1990 Canadian Microelectronics Corporation Workshop, Queen's University, Kingston, Ontario, June 1990.

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

[77] Carver Mead and Martin Rem, *Minimum propagation delays in VLSI*, Technical Memorandum, CalTech, 1982.

[78] Dejan Mijuskovic, *Clock distribution in ASIC*, Micro-electronics Manual, July 1988.

[79] *The Cadence Edge Environment for CMOS4S*, VLSI implementation centre report, CMC, March 1990.

[80] *Cadence Reference Manual*, Cadence Design Inc, ver. 2.1f, California, 1989.

[81] *SILOS II User's Manual*, Simucad Inc., Palo Alto, California, 1988.

[82] *The TTL Data Book: vol. 2*, Texas Instruments, 1985.

[83] J. H. McClellan, Charles M. Rader, *Number Theory in Digital signal processing*, Prentice Hall, 1979.

[84] F. J. Taylor, "Residue Arithmetic: A tutorial with examples", *IEEE Computer Magazine*, pp. 50-62, May 1984.

[85] Giuseppe Alia, Enrico Martinelli, A VLSI Structure for X(mod m) Operation, *Journal for VLSI Signal Processing*, vol. 1, pp. 257-264, 1990.

[86] M. A. Bayoumi, G. A. Jullien, W. C. Miller, A VLSI model for Residue Number System Architectures, *INTEGRATION, the VLSI journal* 2, pp. 191-211, 1984.

[87] M. A. Bayoumi et al.,"Modes for VLSI implementation of residue number system arithmetic modulus", *Proc. IEEE 6th Symp. on Comp. Arithmetic*, pp. 174-182, June 1983.

[88] F.J. Taylor, "A VLSI residue arithmetic multiplier", *IEEE Trans. Computers.*, vol C-31, 1982, pp. 540-546.

[89] W. K. Jenkins and B.J. Leon, "The use of residue number systems in the design of finite impulse response digital filters", *IEEE Trans. Circuits and Systems.*, vol CAS-24, 1977, pp. 191-201.

[90] M. A. Soderstrand, "A high speed low-cost recursive digital filtering using Residue Arithmetic", *Proc. IEEE*, vol. 65, July 1977, pp. 1065-1067.

[91] C. H. Huang, D.G.Patterson et al, "Implementation of a fast digital processor using the residue number system", *IEEE Trans. Circuits and Systems*, vol CAS-28, 1981, pp. 32-38.

[92] Tremblay and Manohar, *Discrete Mathematical Structures with Application to Computer Science*, McGraw-Hill,1985.

[93] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1984.

[94] W. W. Peterson and E.J. Weldon, *Error Correcting Codes* ,MIT Press, 2nd edition, 1972.

[95] S. W. Goloumb, *Shift Register Sequences* ,Aegean Park Press, California, 1982.

[96] Edward Mcluskey, *Logic Design Principles* Prentice Hall, New Jersey,1986.

[97] Brian D. Ripley, *Stochastic Simulation*, John Wiley, New York, 1987.

[98] Richard S. Lehman, *Computer Simulation and Modeling - An Introduction*, John Wiley, New York, 1977.

[99] Frederick. S. Hillier and Gerald J. Leiberman, *Operations Research*, Holden-Day Inc, San Francisco, 1967.

[100] J. Kirkpatrick and Stoll, "A very fast shift register sequence Random Number Generator", *Journal of Computational Physics*, vol. 40,1981, pp. 1065-1067.

[101] Istvan Deak, "Uniform Random Number Generators for Parallel Computers", *Parallel Computing*, vol. 15, Sept 1990, pp. 155-164.

[102] A. De Matteis, and S. Pagnutti, "Long-range correlations in linear and and non-linear random number generators", *Parallel Computing.*, vol. 14, No. 2, June 1990, pp. 207-210.

[103] P. D. Hortensius, R.D. Mcleod and H.C. Card, "Parallel Random Number Generation for VLSI systems using Cellular Automata", *IEEE Trans. on Computers.*, vol. 38, No. 10, Oct 1988, pp. 1466-1473.

[104] D. H. Lehmer, "Mathematical methods in large scale computing units", *Annu. Comput. Lab, Harvard University*, 26, 1951, p 141-146.

[105] D. E. Knuth, *Semi Numerical Algorithms - The Art of Computer Programming*, vol. 2, Reading-Mass, San Jose, 1981.

[106] T. E. Hull, A. R. Dobell, "Mixed Congruential Random Number Generators for binary machines", *J. ACM .*, vol. 11, 1964, pp. 31-40.

[107] T. E. Hull, A. R. Dobell, "Random Number Generators", *SIAM.*, Rev 4, 1962, pp. 230-254.

[108] Donald M. Maclaren and George Marsaglia, "Uniform Random Number Generators", *Journal of the ACM*, vol. 12, No.1, Jan 1965, pp. 83-89.

[109] David G. Carta, "Two Fast Implementations of the "Minimal Standard" Random Number Generator", *Communications of the ACM.*, vol. 33, No. 1, Jan 1990, pp. 87-88.

[110] Stephen K. Park, Keith W. Miller, "Random Number Generators: Good ones are hard to find", *Communications of the ACM.*, vol. 31, No. 10, Oct 1988, pp. 1192-1201.

[111] IBM Corp., *Random Number Generation and Testing*, Reference Manual, C20-8011, New York,1959.

[112] D. E. Knuth, *The Art of Computer Programming*, vol. 3, 'Sorting and Searching', Addison-Wesley, Reading-Mass, 1981.

[113] E. R. Berlekamp,*Algebraic Coding Theory*, McGraw Hill, 1968.

[114] D. E. Knuth, *The Art of Computer Programming*, vol. 3, "Networks for Sorting", Addison-Wesley, Reading-Mass, 1981, pp. 235.

[115] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading-Mass, 1974.

[116] G. K. Kostopoulos, *Digital Engineering*, Wiley Inter-Science, 1975.

[117] N. J. Dimopoulos, D. Radvan, R. Sivakumar, 'Implementation of the Backtrack-to-the Origin-and-Retry routing of the Hypercycle based Interconnection Networks', *Proc. Canadian Conference on Elect. & Comp. Eng*, Ottawa, Sept. 1990, pp. 67.2.1-67.2.4.

[118] T. W. Williams, K. P. Parker, "Design for Testability - A Survey" *IEEE Trans. on Computers*, C-31, No.1, January 1982, pp. 2-15.

[119] Barry. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading-Mass, 1989.

[120] C. Timoc, F. Stott and L. Hess, "A Novel Method for Test Generation for VLSI", *Selected reprints on Logic design for Testability*, IEEE Society Press, 1984, pp. 187-194.

[121] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw Hill, New York, 1978.

[122] R. A. Rasmussen, "Automated Testing of LSI" *Selected reprints on Logic design for Testability*, IEEE Society Press, 1984, pp. 3-12.

[123] H. Ando, "Testing VLSI- Random Access Scan" *Proceedings of COMPCON S'80*, 1980, pp. 50-52.

[124] S. C. Seth and V. D. Agrawal, "Cutting chip-testing costs" *IEEE Spectrum*, 1985, pp. 38-45.

[125] J. P. Roth, "Diagnosis of automata failures: A calculus and a method" *IBM Journal of Research and development*, vol. 10, No.7, July 1966, pp. 278-291.

[126] P. Goel, "An implicit enumeration algorithm to generate tests fro combinational logic circuits" *IEEE Trans. on Computers*, C-30, No.3, March 1981, pp. 215-222.

[127] *CATPAG User's Manual*, ver 2.0.1, Report ICI-025R0, Canadian Microelectronics Corporation, Queen's University, 1990.

[128] *Logic Master XL Series Reference Manual*, IMS Inc., ver 4.4, 1989.

[129] T. W. William "Random Patterns within a Structured Sequential Logic Design" *Proceedings of Int. Test Conference*, 1977, pp. 19-26.

[130] N. J. Dimopoulos, R. Sivakumar, D. Radvan, "Routing and Processor Allocation on a Hypercycle based Multiprocessor", to be presented in *The 1991 International Conference on Supercomputing*, Cologne, Germany, June 17-21, 1991.

[131] F. Ercal, J. Ramanujam and P. Sadayappan, "Task allocation onto a Hypercube by Recursive Mincut Bipartitioning", *Journal of Parallel and Distributed Computing*, vol. 10, 1990, pp. 35-44.

[132] F. Dehne and M. Gastaldo, "A note on the load balancing problem for course grained Hypercube dictionary machines", *Journal of Parallel Computing*, vol. 16, 1990, pp. 75-79.

Appendix A

Schematic Diagrams of Next Port Generator

The schematic layout of the Next Port Generator along with the bonding diagram of the circuit implemented in 3 μ CMOS3DLM are shown in pages 177-184. The NPG is a 26 pin chip housed in 40-pin DIP. Its pin diagram is shown in Fig. A.1.

A.1 Pin descriptions

Beta⟨3⟩

BETA⟨3⟩ is a 4-bit input that denotes the destination address. The pin specifications correspond to the following.

BETA⟨3 : 0⟩	
Data	Pin No.
BETA⟨3⟩	21
BETA⟨2⟩	31
BETA⟨1⟩	28
BETA⟨0⟩	29

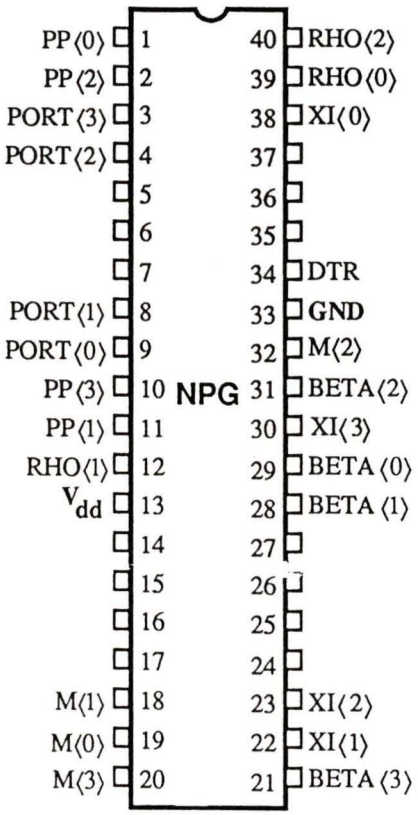


Figure A.1: Pin layout of NPG

XI⟨3 : 0⟩

XI⟨3 : 0⟩ is a 4-bit input that corresponds to the source address. The pin specifications are as follows:

XI⟨3⟩	
Data	Pin No.
XI⟨3⟩	30
XI⟨2⟩	23
XI⟨1⟩	22
XI⟨0⟩	38

M ⟨3 : 0⟩

M ⟨3⟩ is a 4-bit input that corresponds to the population of the network. The pin specifications are as follows:

M⟨3⟩	
Data	Pin No.
M⟨3⟩	20
M⟨2⟩	32
M⟨1⟩	18
M⟨0⟩	19

PP⟨3 : 0⟩

PP⟨3 : 0⟩ is a 4-bit input which signifies the offset of the previous dimensions to be added to the computed logical port. The pin listing for PP⟨3 : 0⟩ are:

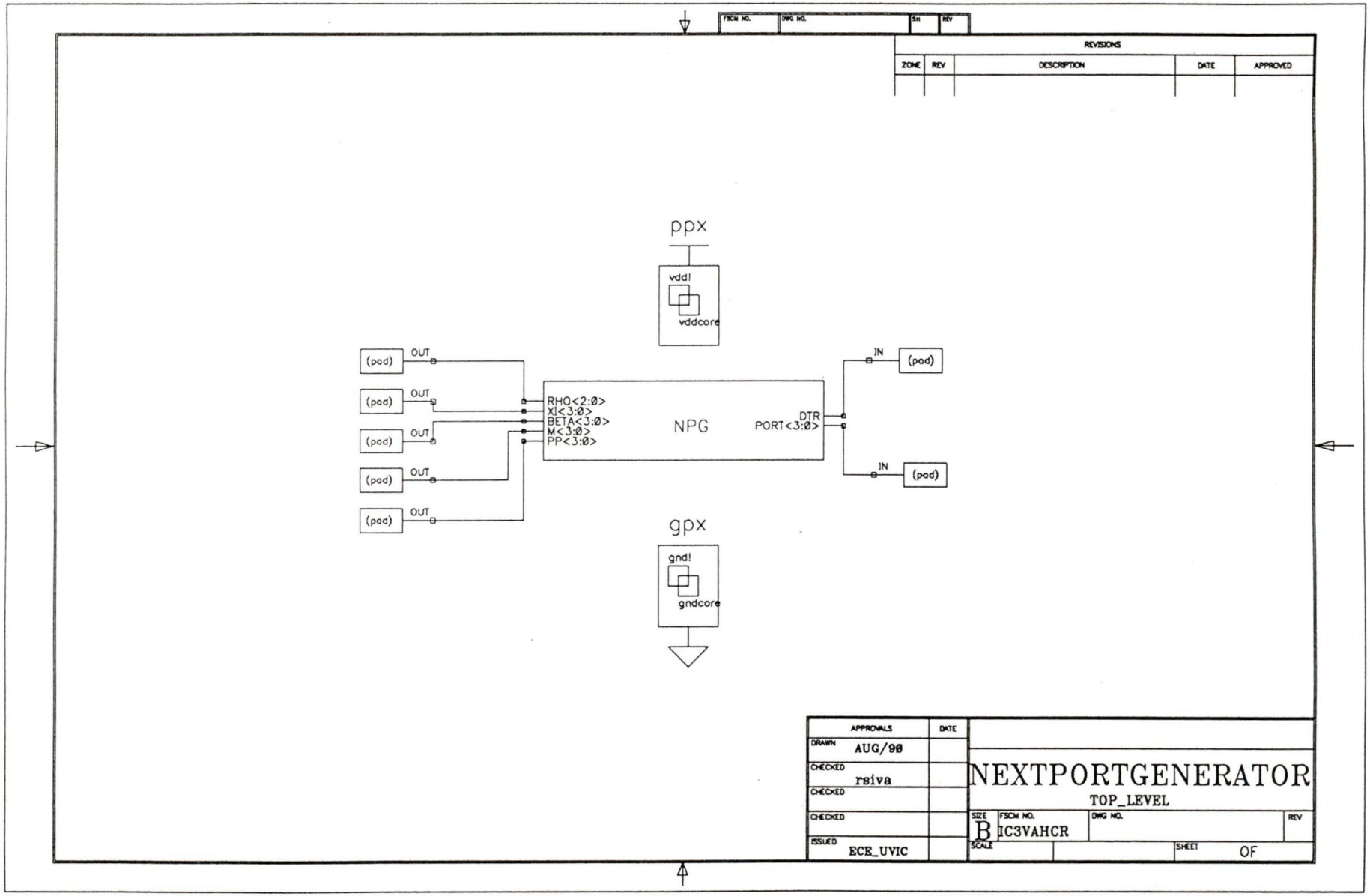
PP⟨3 : 0⟩	
Data	Pin No.
PP⟨3⟩	10
PP⟨2⟩	2
PP⟨1⟩	11
PP⟨0⟩	1

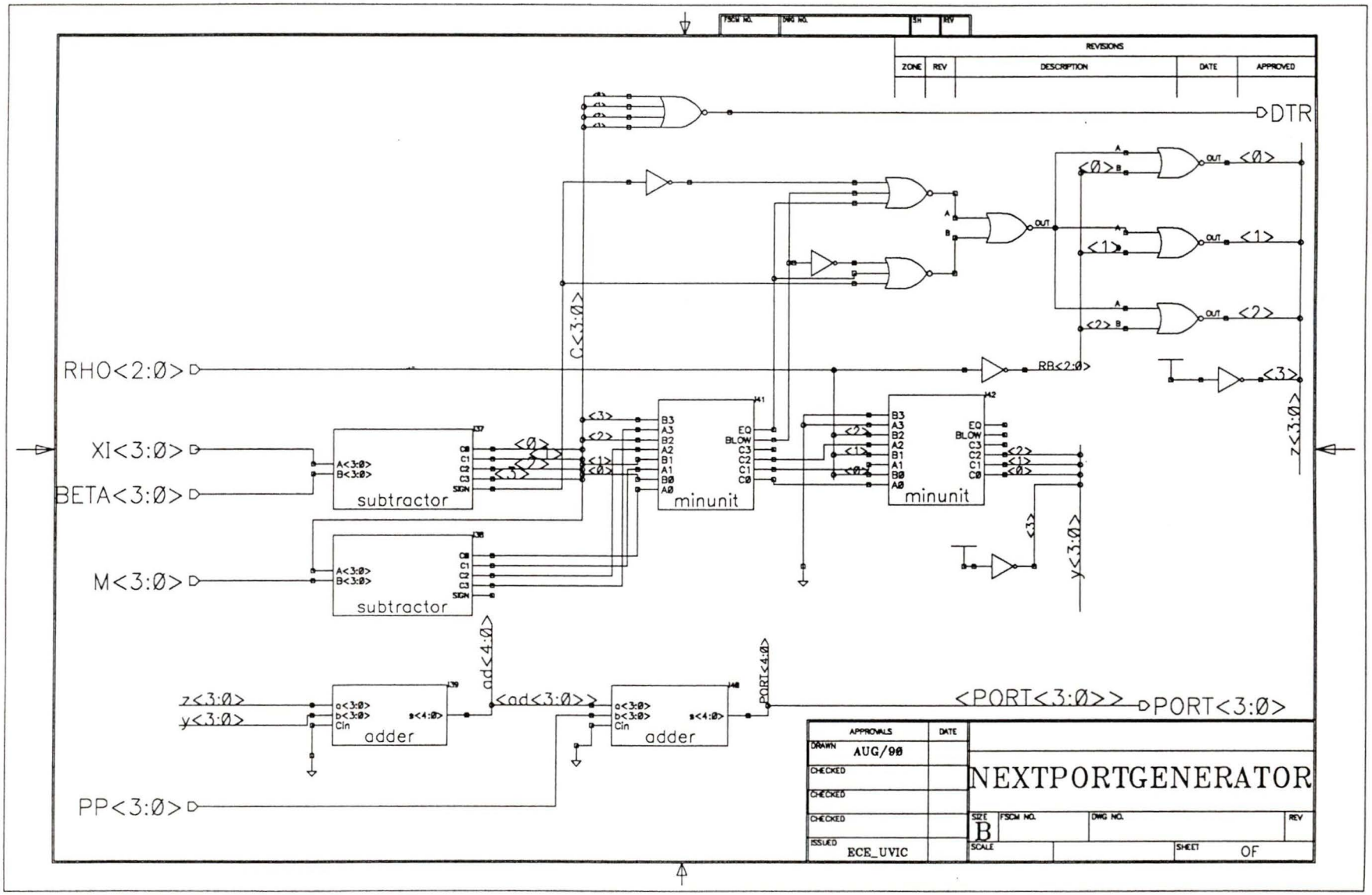
PORT⟨3 : 0⟩

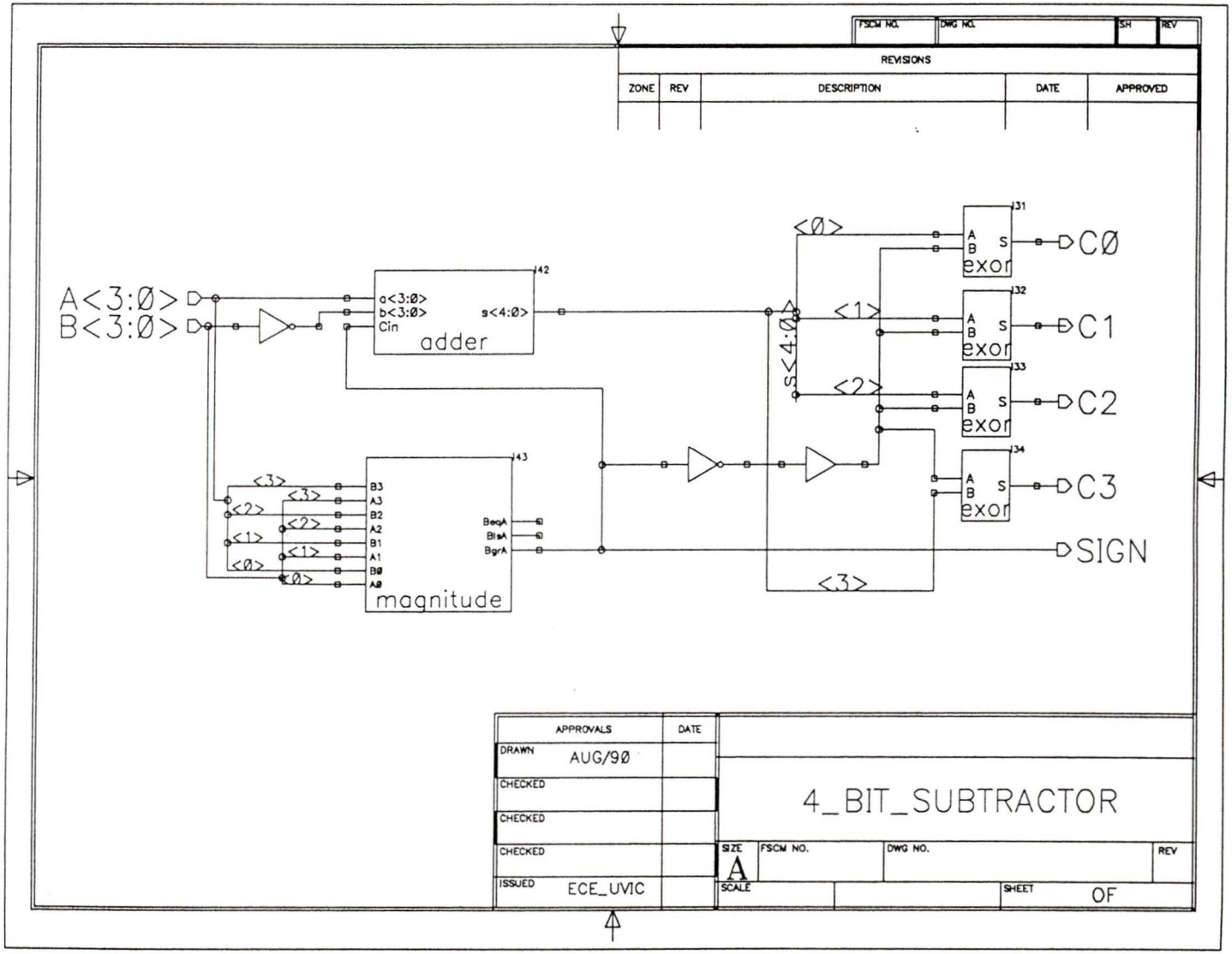
PORT⟨3 : 0⟩ is a 4-bit output which notes the next port to route the message. The pin listing for PORT⟨3 : 0⟩ are:

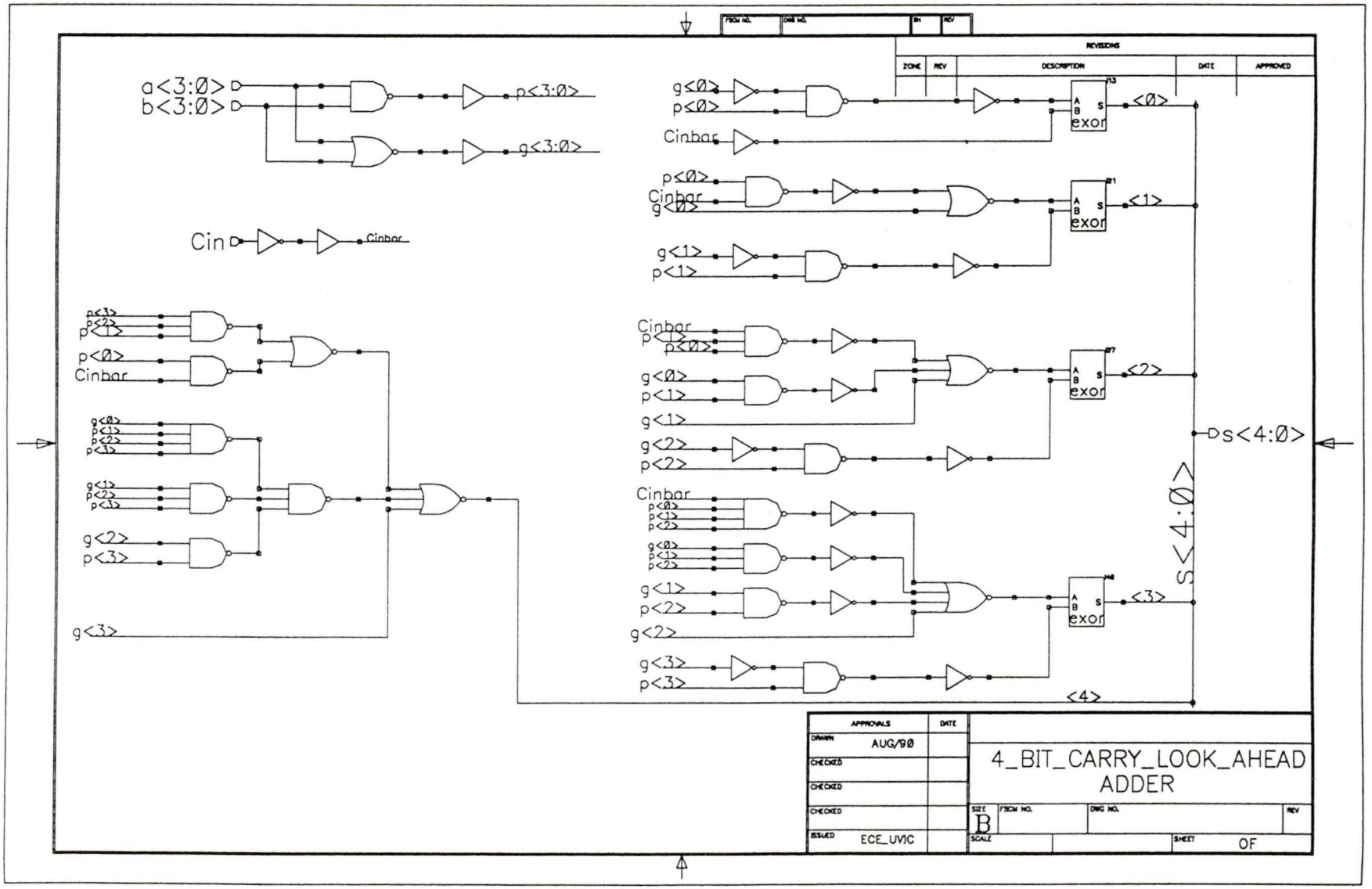
PORT⟨3 : 0⟩	
Data	Pin No.
PORT⟨3⟩	3
PORT⟨2⟩	4
PORT⟨1⟩	8
PORT⟨0⟩	9

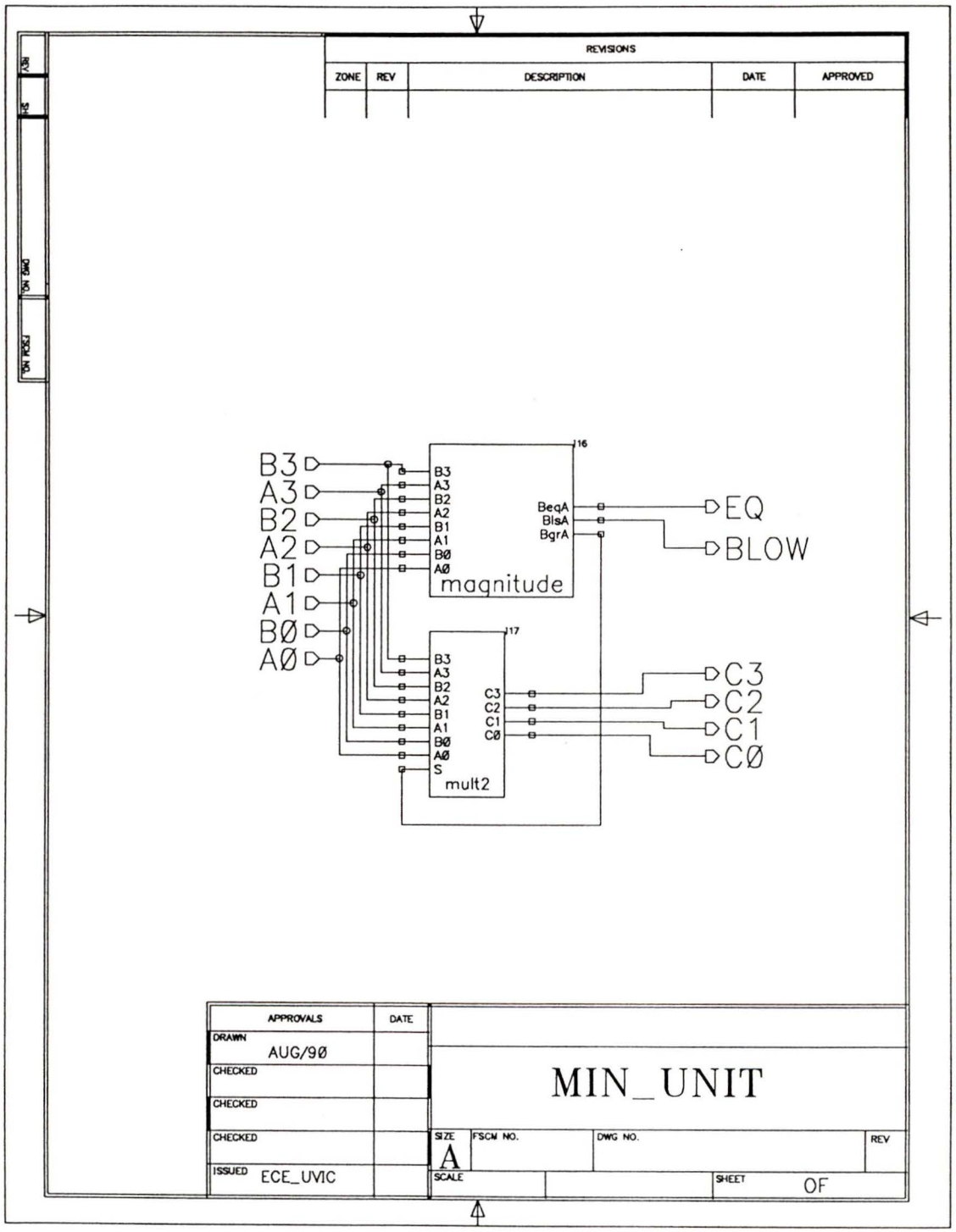
Finally, the DTR signal corresponds to pin 33, while V_{dd} and V_{ss} are assigned to pins 13 and 33 respectively.

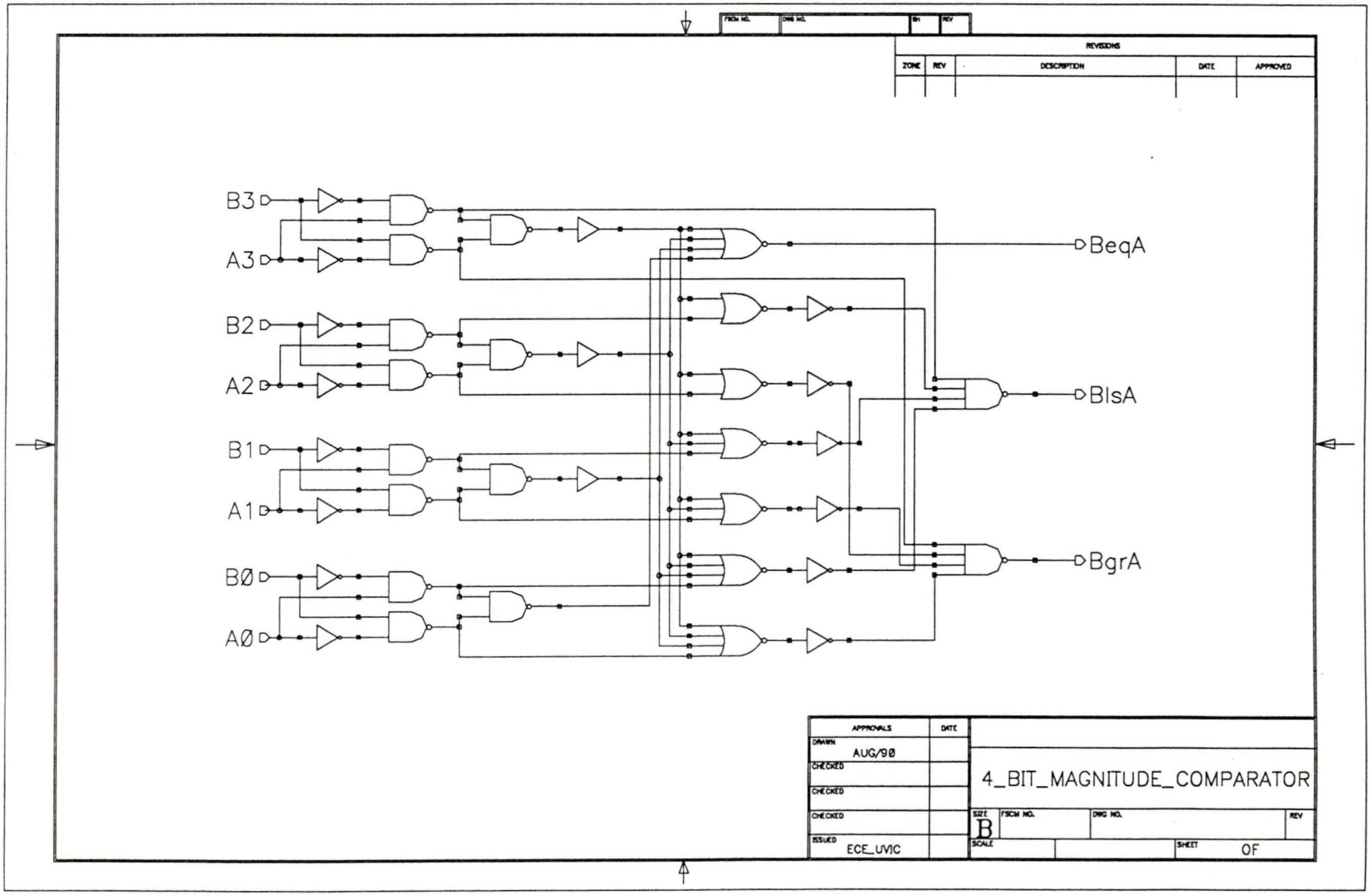

FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
ppx
vdd!
vddcore
(pad)
OUT
RHO<2:0>
XI<3:0>
BETA<3:0>
M<3:0>
PP<3:0>
NPG
DTR
PORT<3:0>
IN
gpx
gnd!
gndcore
APPROVALS
DATE
DRAWN
AUG/98
CHECKED
rsiva
ISSUED
ECE_UVIC
NEXTPORTGENERATOR
TOP_LEVEL
SIZE
B
IC3VAHCR
SCALE
SHEET
OF

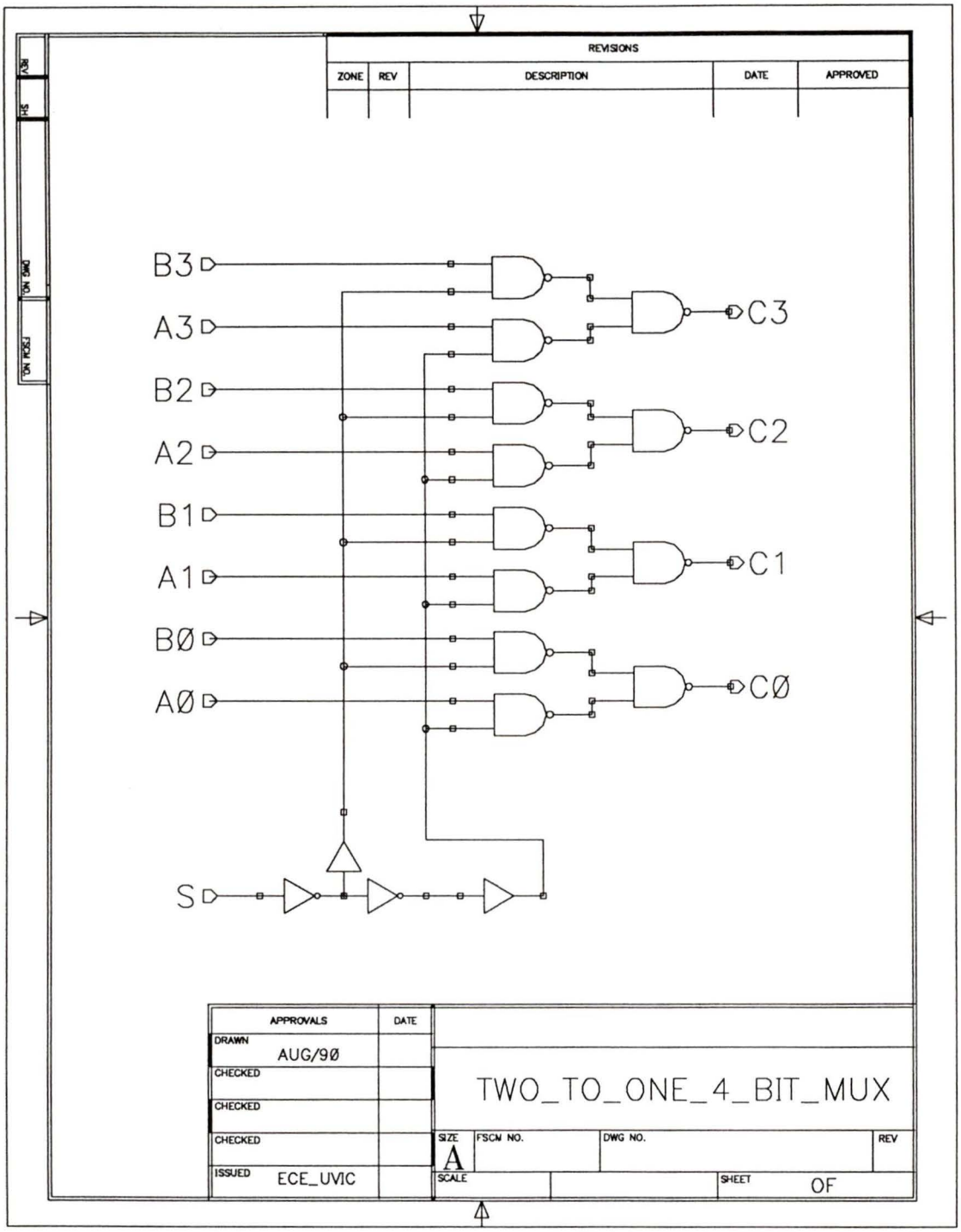

REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
DTR
RHO<2:0>
XI<3:0>
BETA<3:0>
M<3:0>
PP<3:0>
C<3:0>
RB<2:0>
z<3:0>
y<3:0>
subtractor
subtractor
minunit
minunit
adder
adder
ad<4:0>
<ad<3:0>>
PORT<4:0>
<PORT<3:0>>
PORT<3:0>
APPROVALS
DATE
DRAWN
AUG/90
CHECKED
CHECKED
CHECKED
ISSUED
ECE_UVIC
NEXTPORTGENERATOR
SIZE
B
FSCM NO.
DWG NO.
REV
SCALE
SHEET
OF


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
A<3:0>
B<3:0>
a<3:0>
b<3:0>
Cin
s<4:0>
adder
42
s<4:0>
<0>
<1>
<2>
<3>
31
32
33
34
exor
C0
C1
C2
C3
SIGN
B3
A3
B2
A2
B1
A1
B0
A0
BeqA
BltA
BgrA
magnitude
43
APPROVALS
DATE
DRAWN
AUG/90
CHECKED
CHECKED
CHECKED
ISSUED
ECE_UVIC
4_BIT_SUBTRACTOR
SIZE
A
FSCM NO.
DWG NO.
REV
SCALE
SHEET
OF


a<3:0>
b<3:0>
p<3:0>
g<3:0>
Cin
Cinbar
s<4:0>
exor
4_BIT_CARRY_LOOK_AHEAD ADDER
AUG/90
ECE_UVIC


REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
B3
A3
B2
A2
B1
A1
BØ
AØ
16
magnitude
BeqA
BlsA
BgrA
EQ
BLOW
17
mult2
C3
C2
C1
CØ
S
APPROVALS
DATE
DRAWN
AUG/90
CHECKED
CHECKED
CHECKED
ISSUED
ECE_UVIC
MIN_UNIT
SIZE
A
FSCM NO.
DWG NO.
REV
SCALE
SHEET
OF


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
B3
A3
B2
A2
B1
A1
BØ
AØ
BeqA
BlsA
BgrA
APPROVALS
DATE
DRAWN
AUG/90
CHECKED
CHECKED
CHECKED
ISSUED
ECE_UVIC
4_BIT_MAGNITUDE_COMPARATOR
SIZE
B
FSCM NO.
DWG NO.
REV
SCALE
SHEET
OF

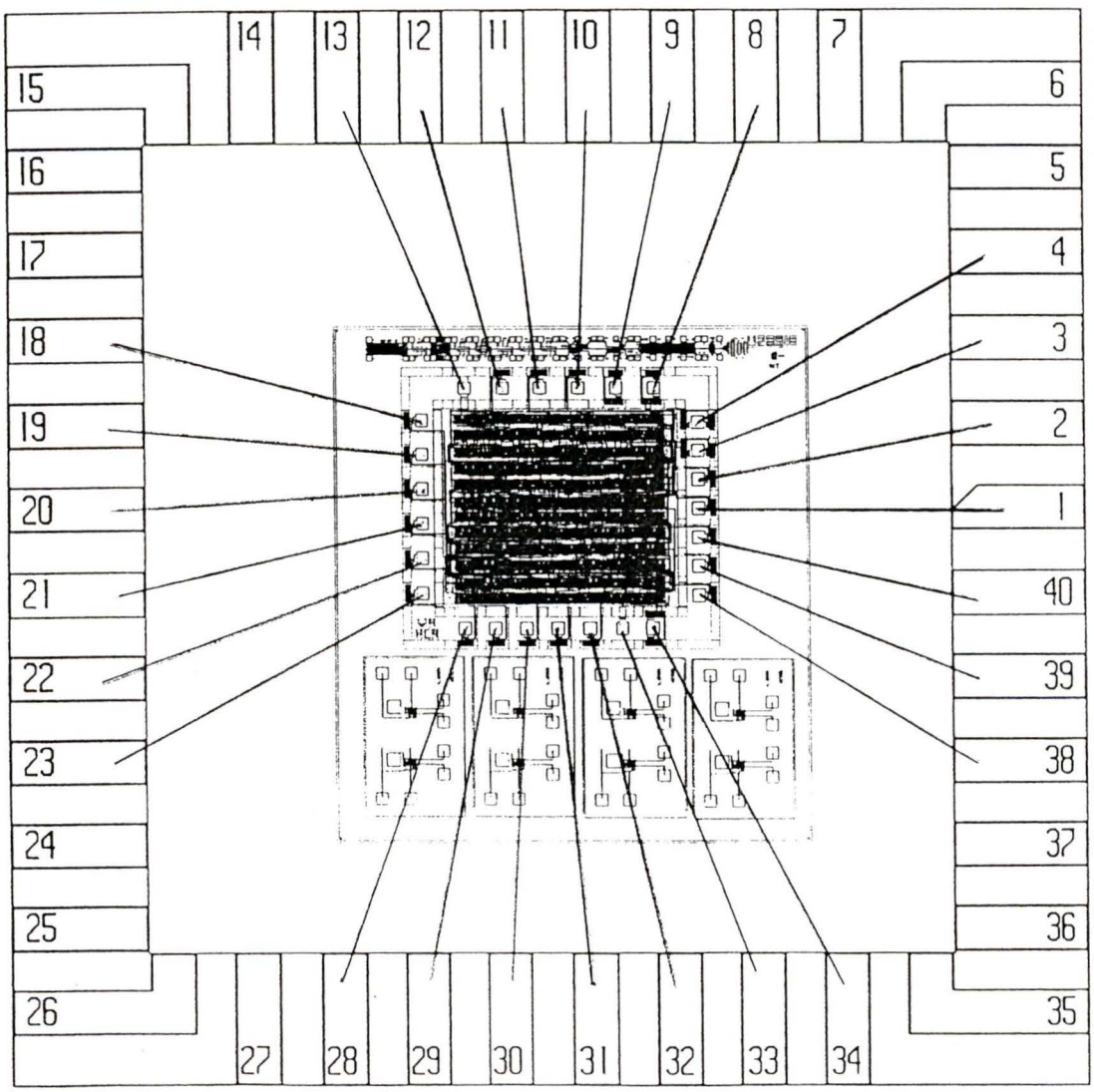

REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
B3
A3
B2
A2
B1
A1
BØ
AØ
S
C3
C2
C1
CØ
APPROVALS
DATE
DRAWN
AUG/90
CHECKED
CHECKED
CHECKED
ISSUED
ECE_UVIC
TWO_TO_ONE_4_BIT_MUX
SIZE
A
FSCM NO.
DWG NO.
REV
SCALE
SHEET
OF

CMC MULTIPROJECT BONDING DIAGRAM
RETICLE CODE U43W
BRAND ID LABEL IC3 / VAHCR
OTHER IDENTIFICATION FEATURES
WAFER NUMBERS
DESIGN FILE REFERENCE

PACKAGE	IDK40F1-192G			LID	C-493-175-35M		
WIRE ALLOY	99% Al/1% Si	DIA.	.001"	ELONG.	1.5 - 4%	T.S.	14-16 gms
	99% Al/1% Si		.00125"		1.5 - 4%		18-22 gms
D/A PREFORM	ALLOY 98% Au/2% Si	RECOMMENDED SIZE		W/B METHOD	U.S.		

BONDING DIAGRAM

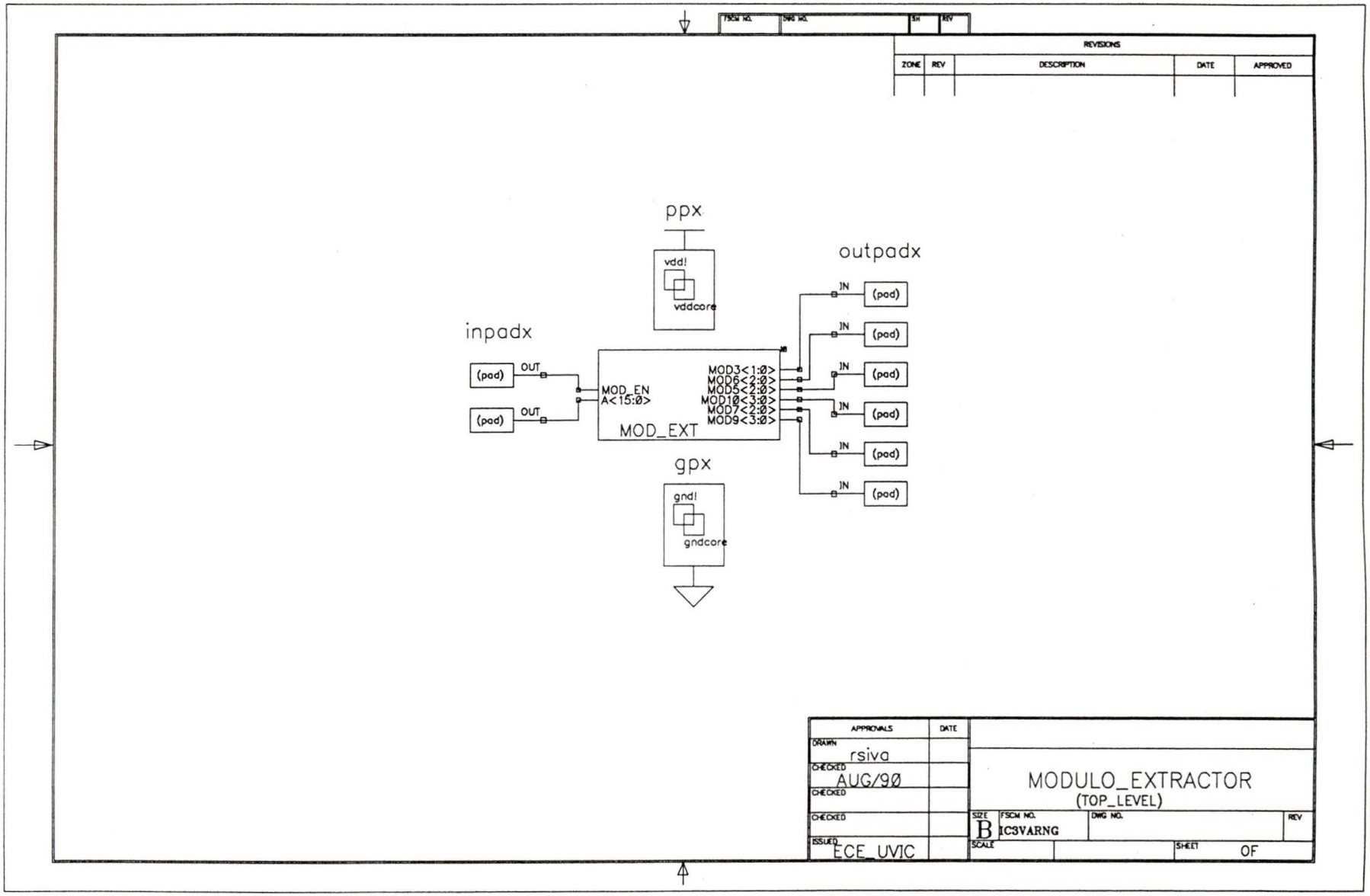
NOTES: 1. DIE ATTACH PAD SIZE: .400 X .400
2. ZERO GROUND

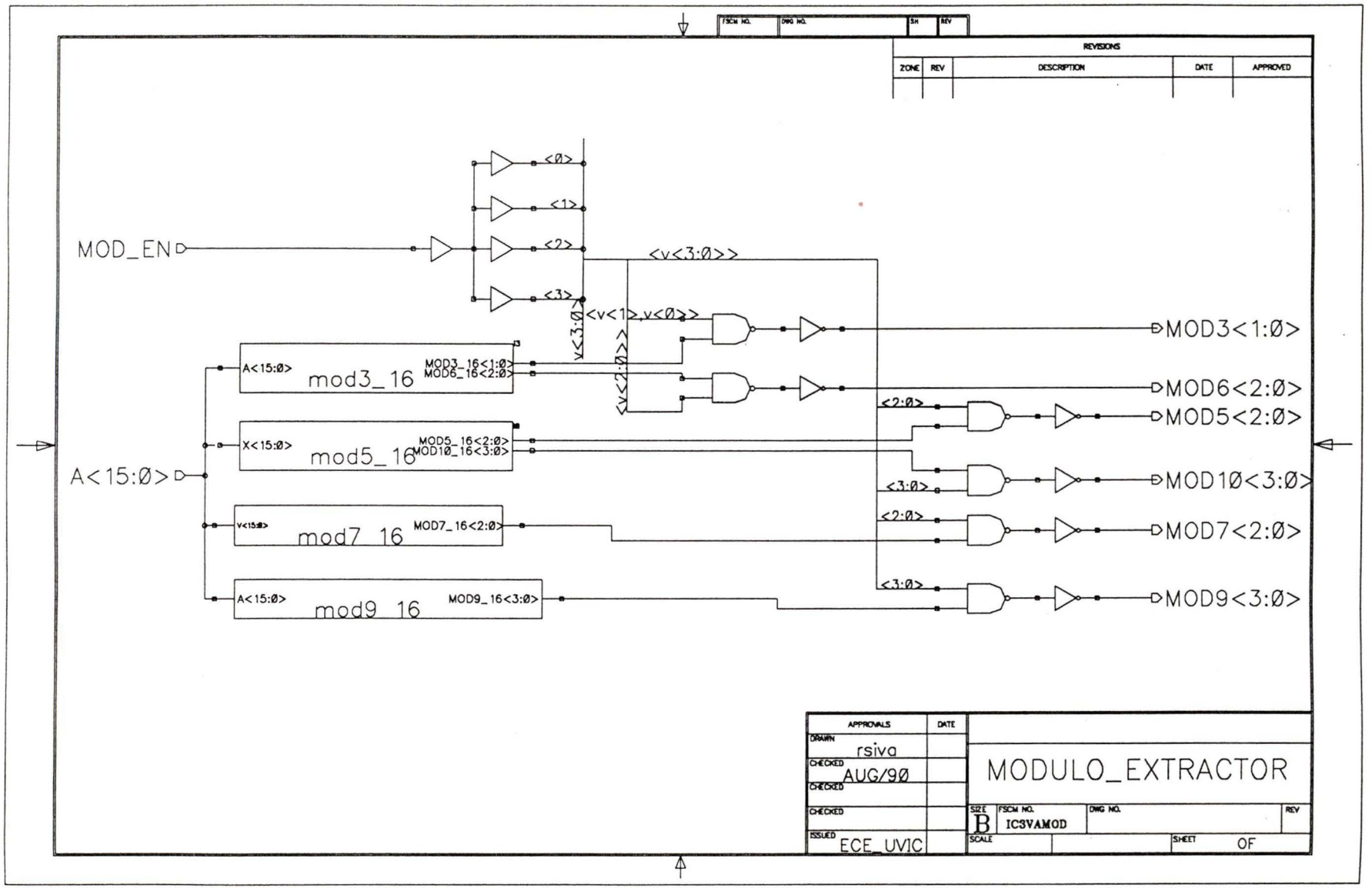


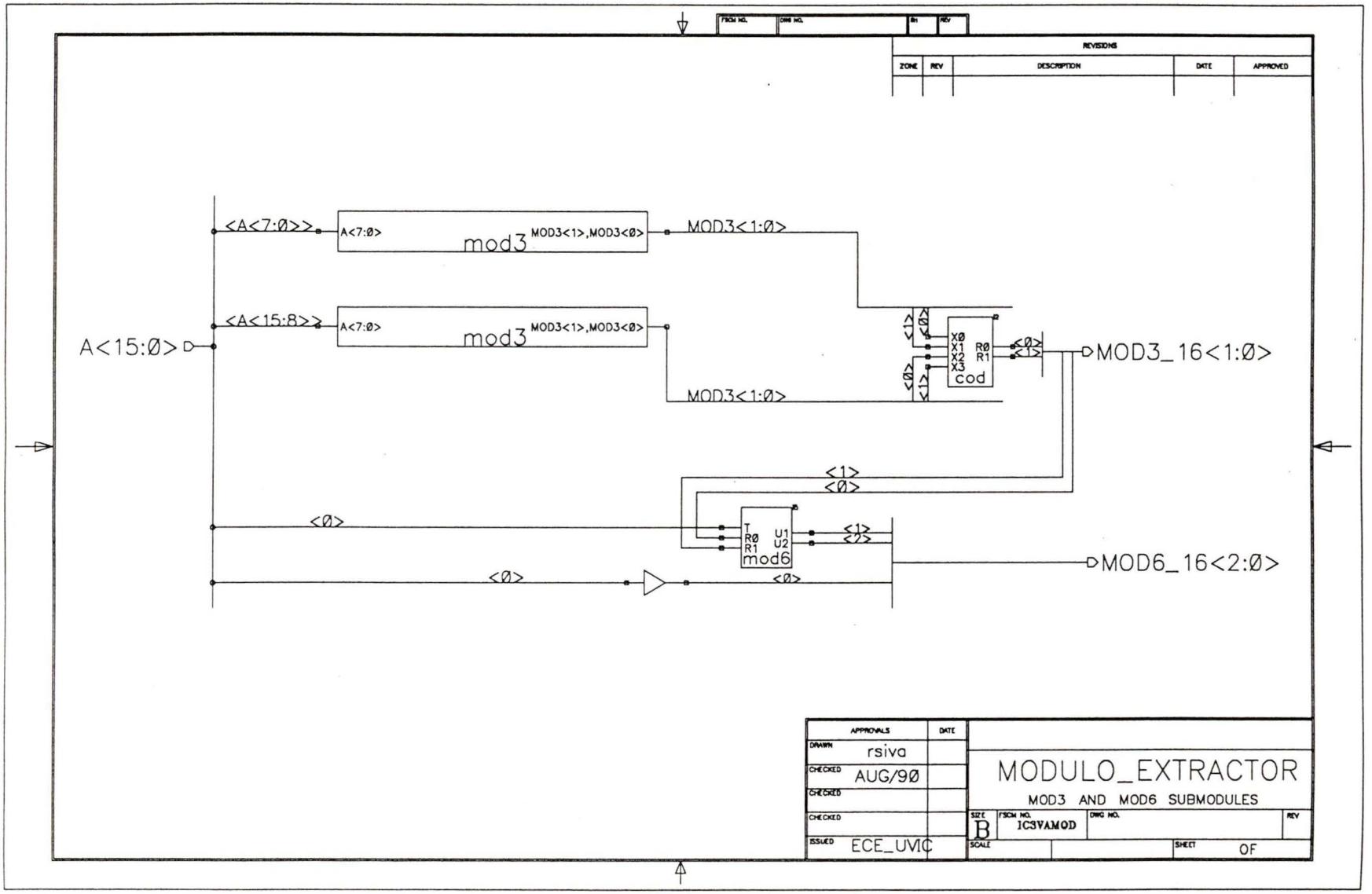
Appendix B

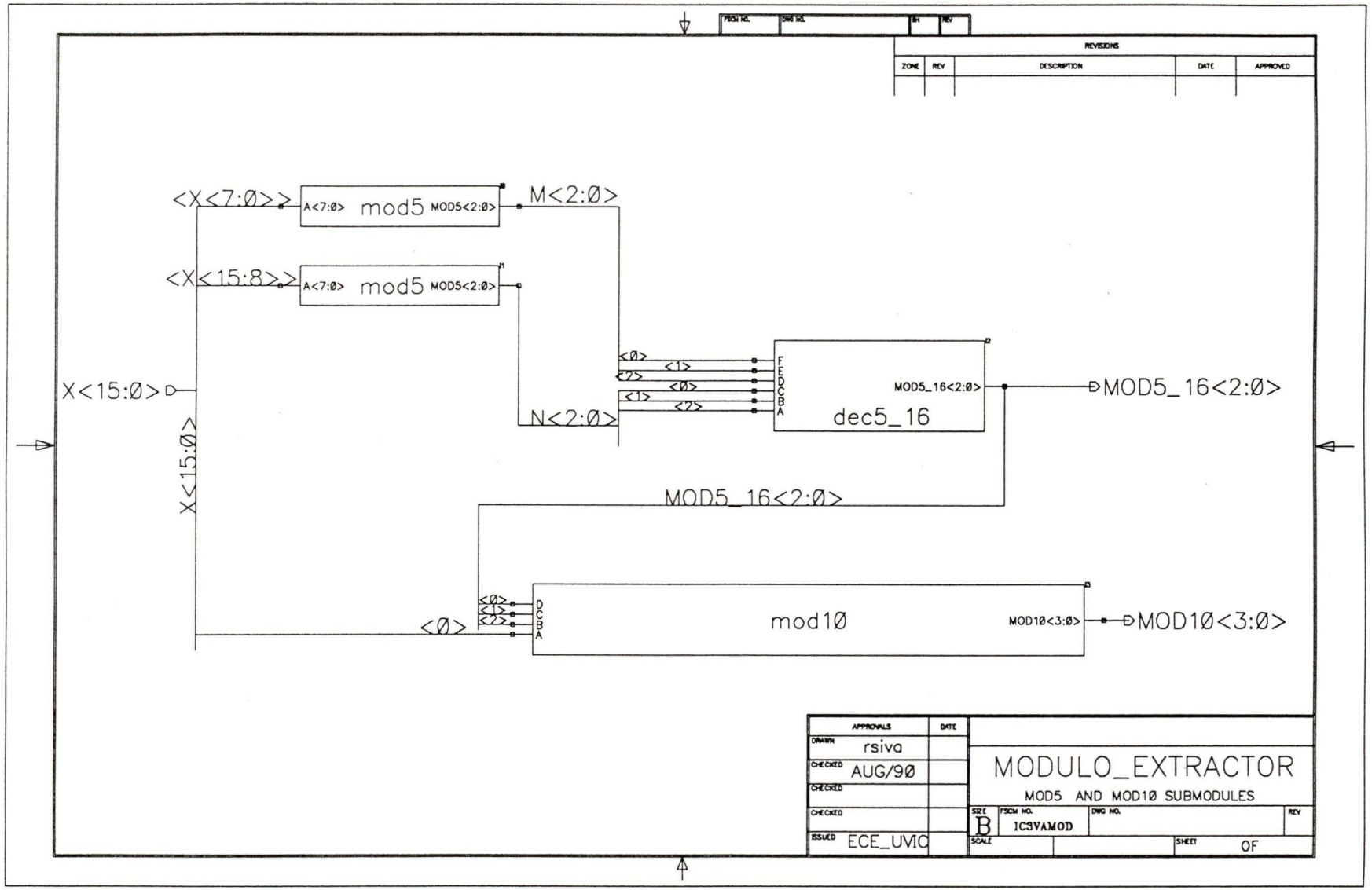
Schematic Diagrams of Modulo-Extractor

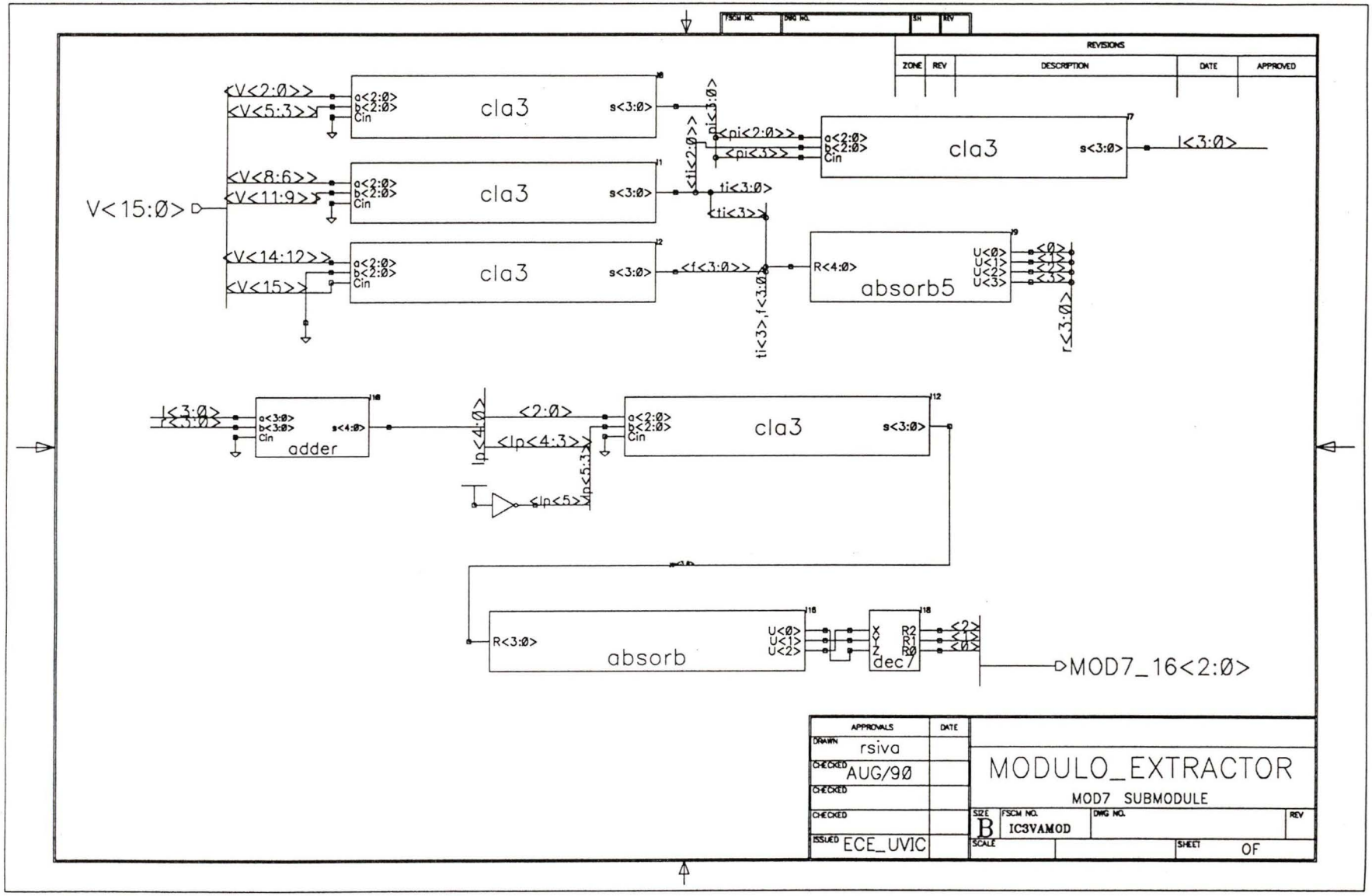
The schematic diagrams of the 16-bit Modulo Extractor along with its sub-blocks implemented in 3 μ CMOS3DLM technology are illustrated in pages 186-192.

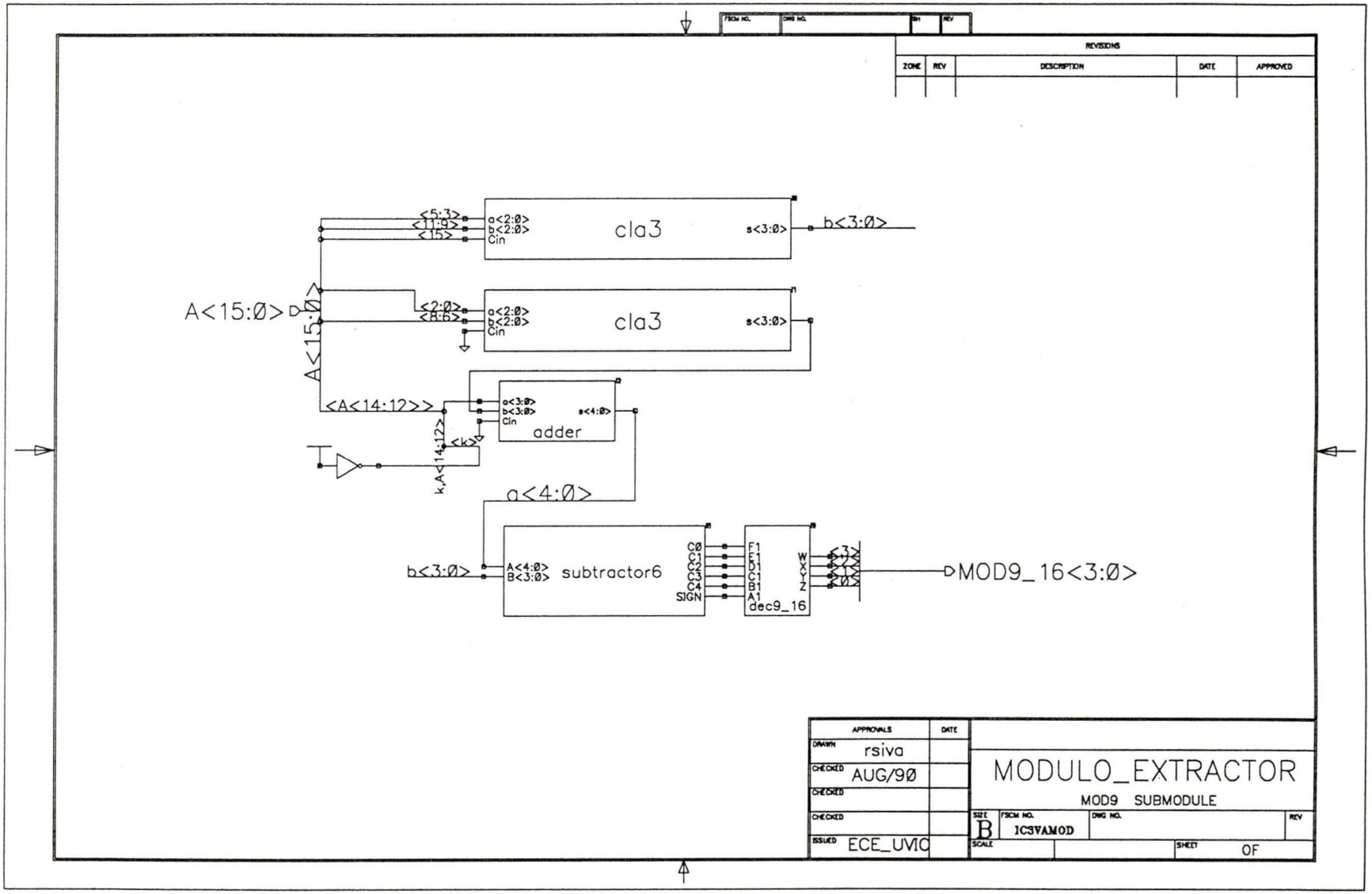

FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
ppx
vdd!
vddcore
outpadx
inpadx
(pad)
OUT
MOD_EN
A<15:0>
MOD_EXT
MOD3<1:0>
MOD6<2:0>
MOD5<2:0>
MOD10<3:0>
MOD7<2:0>
MOD9<3:0>
IN
gpx
gnd!
gndcore
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO_EXTRACTOR
(TOP_LEVEL)
SIZE
B
FSCM NO.
IC3VARNG
DWG NO.
REV
SCALE
SHEET
OF


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
MOD_EN
A<15:Ø>
<Ø>
<1>
<2>
<3>
<v<3:Ø>>
v<3:Ø>
<v<1>,v<Ø>>
v<2:Ø>
A<15:Ø>
mod3_16
MOD3_16<1:Ø>
MOD6_16<2:Ø>
X<15:Ø>
mod5_16
MOD5_16<2:Ø>
MOD1Ø_16<3:Ø>
mod7_16
MOD7_16<2:Ø>
A<15:Ø>
mod9_16
MOD9_16<3:Ø>
MOD3<1:Ø>
MOD6<2:Ø>
<2:Ø>
MOD5<2:Ø>
<3:Ø>
MOD1Ø<3:Ø>
<2:Ø>
MOD7<2:Ø>
<3:Ø>
MOD9<3:Ø>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/9Ø
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO_EXTRACTOR
SIZE
B
FSCM NO.
IC3VAMOD
DWG NO.
REV
SCALE
SHEET
OF


REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
A<15:0>
<A<7:0>>
<A<15:8>>
A<7:0>
mod3
MOD3<1>,MOD3<0>
MOD3<1:0>
MOD3<1:0>
X0
X1
X2
X3
R0
R1
cod
MOD3_16<1:0>
<1>
<0>
<0>
T
R0
R1
U1
U2
mod6
MOD6_16<2:0>
<0>
<0>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO_EXTRACTOR
MOD3 AND MOD6 SUBMODULES
SIZE
B
FSCM NO.
IC3VAMOD
DWG NO.
REV
SCALE
SHEET
OF


REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
<X<7:0>>
A<7:0>
mod5
MOD5<2:0>
M<2:0>
<X<15:8>>
A<7:0>
mod5
MOD5<2:0>
X<15:0>
X<15:0>
N<2:0>
<0>
<1>
<2>
<0>
<1>
<2>
E
D
C
B
A
MOD5_16<2:0>
dec5_16
MOD5_16<2:0>
MOD5_16<2:0>
<0>
D
C
B
A
mod10
MOD10<3:0>
MOD10<3:0>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO_EXTRACTOR
MOD5 AND MOD10 SUBMODULES
SIZE
B
FSCM NO.
IC3VAMOD
DWG NO.
REV
SCALE
SHEET
OF


REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
V<15:0>
cla3
cla3
cla3
cla3
absorb5
adder
cla3
absorb
dec7
MOD7_16<2:0>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO_EXTRACTOR
MOD7 SUBMODULE
SIZE
B
FSCM NO.
IC3VAMOD
DWG NO.
REV
SCALE
SHEET
OF


FSCM NO.
DWG NO.
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
A<15:Ø>
<5:3>
<11:9>
<15>
a<2:Ø>
b<2:Ø>
Cin
cla3
s<3:Ø>
b<3:Ø>
<2:Ø>
<8:6>
a<2:Ø>
b<2:Ø>
Cin
cla3
s<3:Ø>
A<15:Ø>
<A<14:12>>
a<3:Ø>
b<3:Ø>
Cin
s<4:Ø>
adder
<k>
k,A<14:12>
a<4:Ø>
b<3:Ø>
A<4:Ø>
B<3:Ø>
subtractor6
C0
C1
C2
C3
C4
SIGN
F1
E1
D1
C1
B1
A1
dec9_16
W
X
Y
Z
MOD9_16<3:Ø>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO_EXTRACTOR
MOD9 SUBMODULE
SIZE
B
FSCM NO.
IC3VAMOD
DWG NO.
REV
SCALE
SHEET
OF

CMC MULTIPROJECT BONDING DIAGRAM

RETICLE CODE U4302

BRAND ID LABEL IC3 / VAMOD

OTHER IDENTIFICATION FEATURES

WAFER NUMBERS

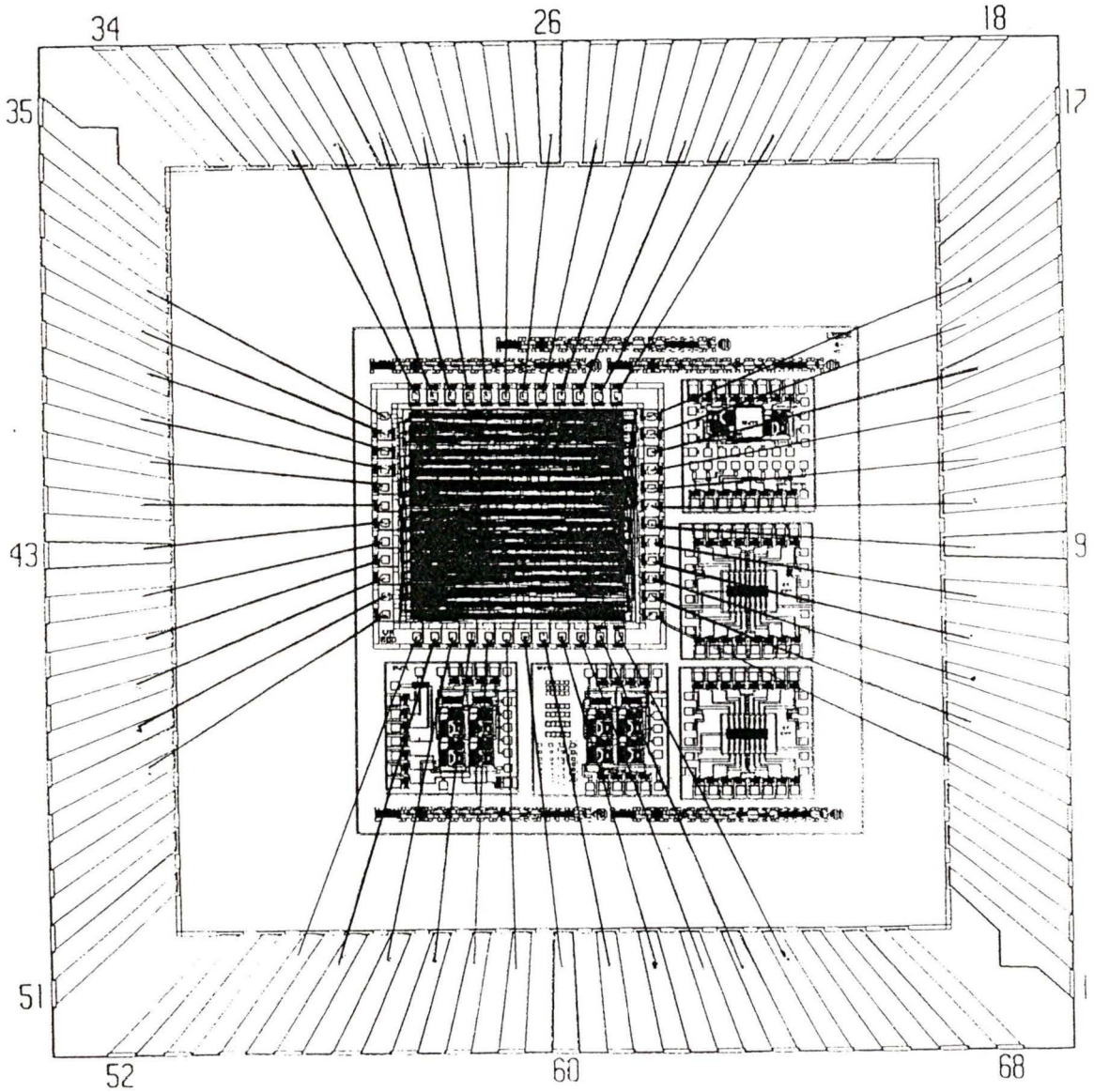
DESIGN FILE REFERENCE

PACKAGE	KD84840A			LID	C-804		
WIRE ALLOY	99% Al/1% Si	DIA.	.001"	ELONG.	1.5 - 4%	T.S.	14-16 gms
	99% Al/1% Si		.00125"		1.5 - 4%		18-22 gms
D/A PREFORM	ALLOY 98% Au/2% Si	RECOMMENDED SIZE				W/B METHOD	U.S.

BONDING DIAGRAM

NOTES: 1. DIE ATTACH PAD SIZE: .400 X .400

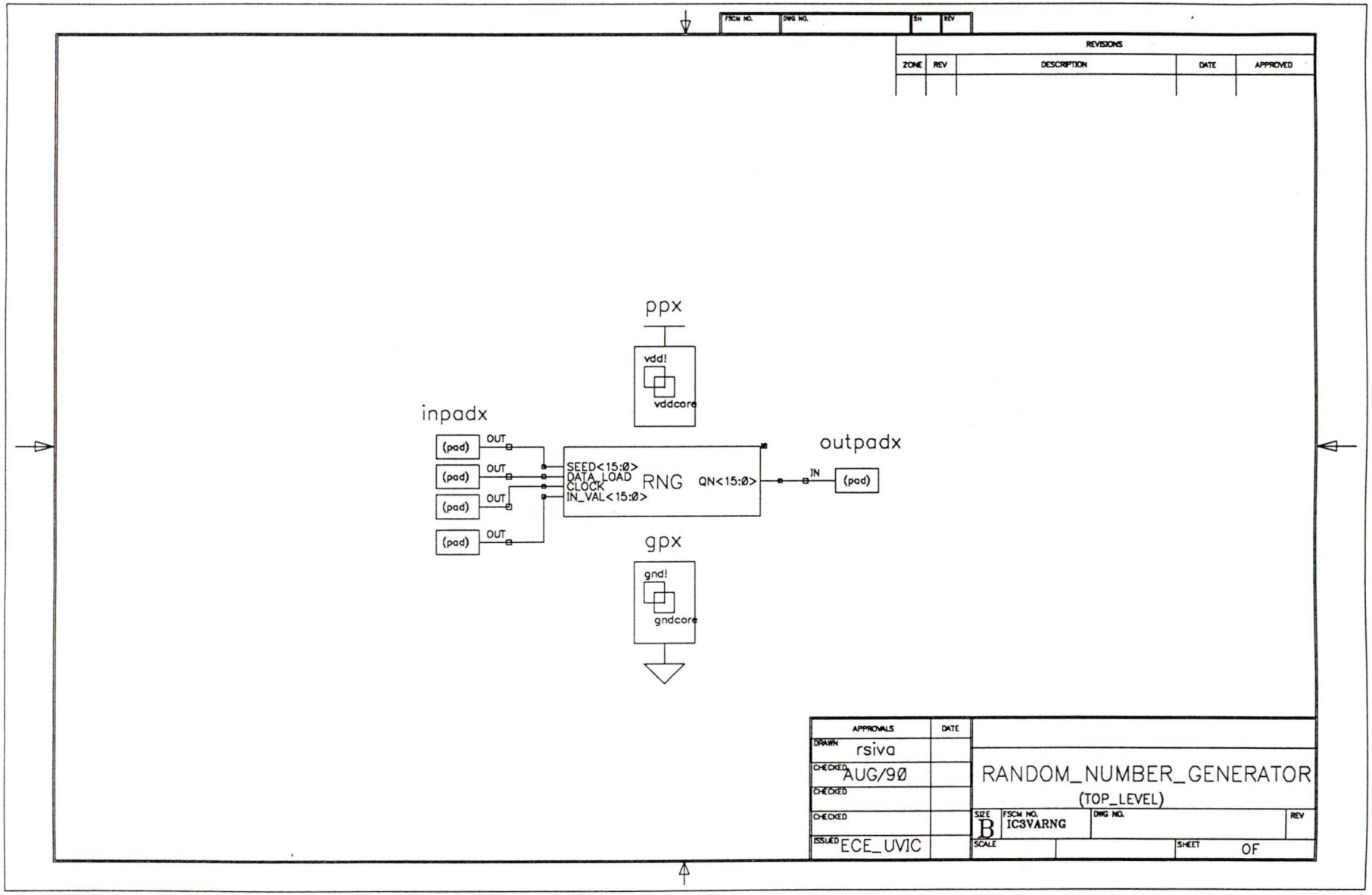
2. ZERO GROUND

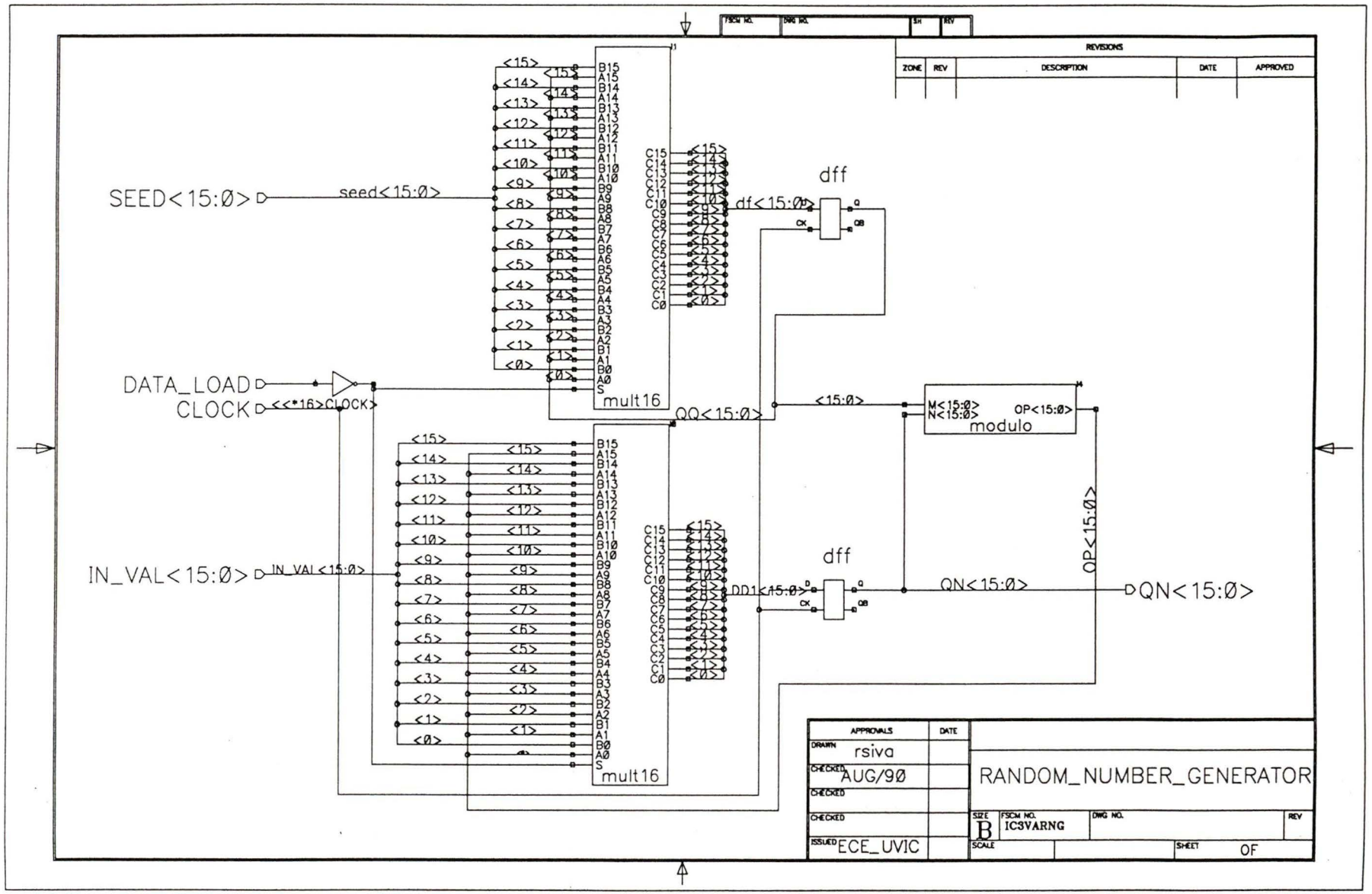


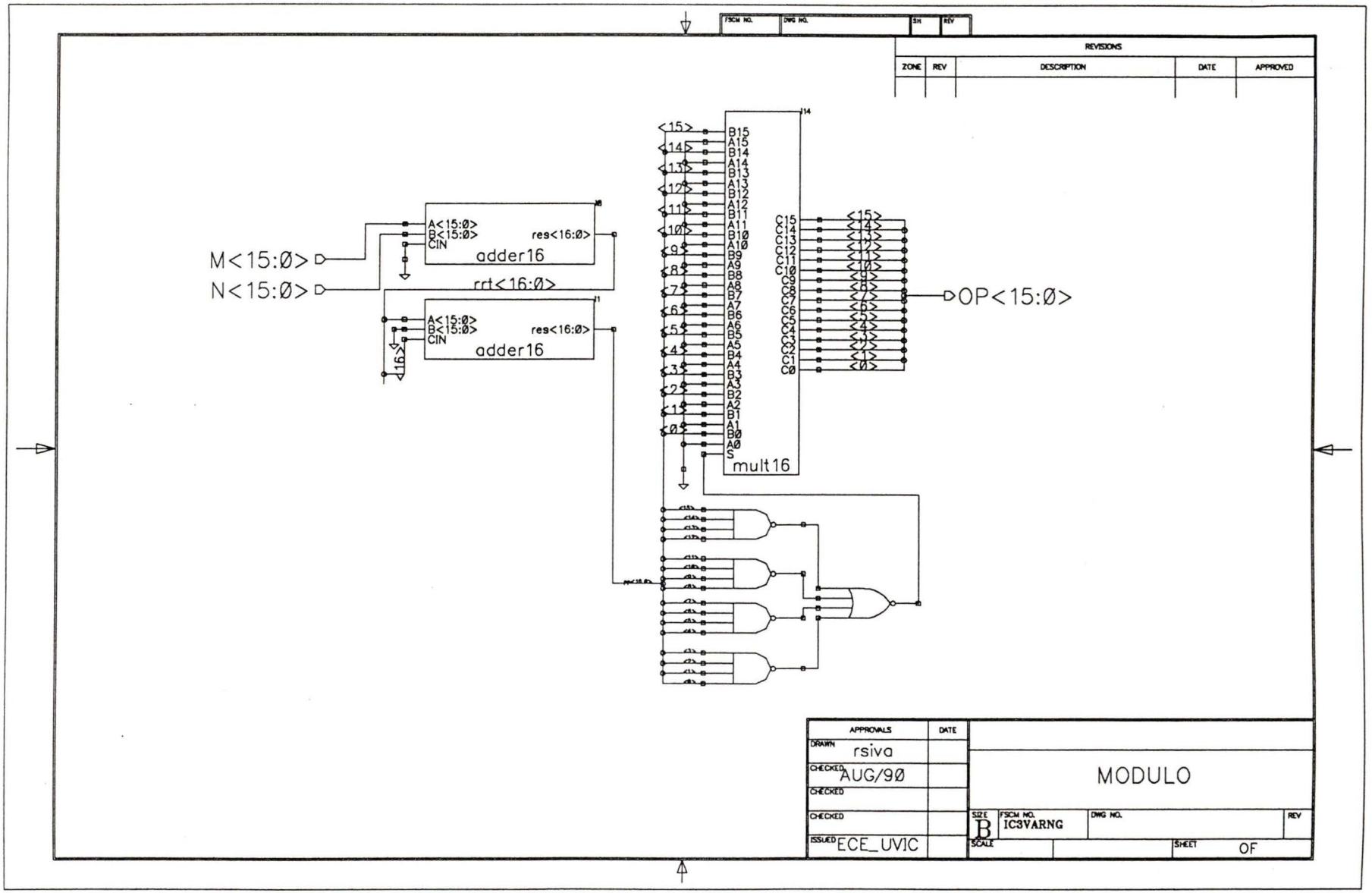
Appendix C

Schematic Diagrams of Random Number Generator

The schematic diagrams of 16-bit mixed congruential Random Number generator implemented in 3 μ CMOS3DLM technology are illustrated in pages 194-197.


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
ppx
vdd!
vddcore
inpadx
(pad)
OUT
(pad)
OUT
(pad)
OUT
(pad)
OUT
SEED<15:0>
DATA_LOAD
CLOCK
IN_VAL<15:0>
RNG
QN<15:0>
outpadx
IN
(pad)
gpx
gnd!
gndcore
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
RANDOM_NUMBER_GENERATOR
(TOP_LEVEL)
SIZE
B
FSCM NO.
IC3VARNG
DWG NO.
REV
SCALE
SHEET
OF


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
SEED<15:0>
seed<15:0>
DATA_LOAD
CLOCK
<<*16>CLOCK>
IN_VAL<15:0>
IN_VAL<15:0>
mult16
mult16
dff
dff
df<15:0>
DD1<15:0>
QQ<15:0>
<15:0>
M<15:0>
N<15:0>
OP<15:0>
modulo
QN<15:0>
QP<15:0>
QN<15:0>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
RANDOM_NUMBER_GENERATOR
SIZE
B
FSCM NO.
IC3VARNG
DWG NO.
REV
SCALE
SHEET
OF


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
M<15:Ø>
N<15:Ø>
A<15:Ø>
B<15:Ø>
CIN
res<16:Ø>
adder16
rrt<16:Ø>
16
B15
A15
B14
A14
B13
A13
B12
A12
B11
A11
B1Ø
A1Ø
B9
A9
B8
A8
B7
A7
B6
A6
B5
A5
B4
A4
B3
A3
B2
A2
B1
A1
BØ
AØ
S
mult16
C15
C14
C13
C12
C11
C10
C9
C8
C7
C6
C5
C4
C3
C2
C1
CØ
OP<15:Ø>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
AUG/9Ø
CHECKED
CHECKED
ISSUED
ECE_UVIC
MODULO
SIZE
B
FSCM NO.
IC3VARNG
DWG NO.
REV
SCALE
SHEET
OF

CMC MULTIPROJECT BONDING DIAGRAM

RETICLE CODE U43C2

BRAND ID LABEL IC3
VARNG

OTHER IDENTIFICATION FEATURES

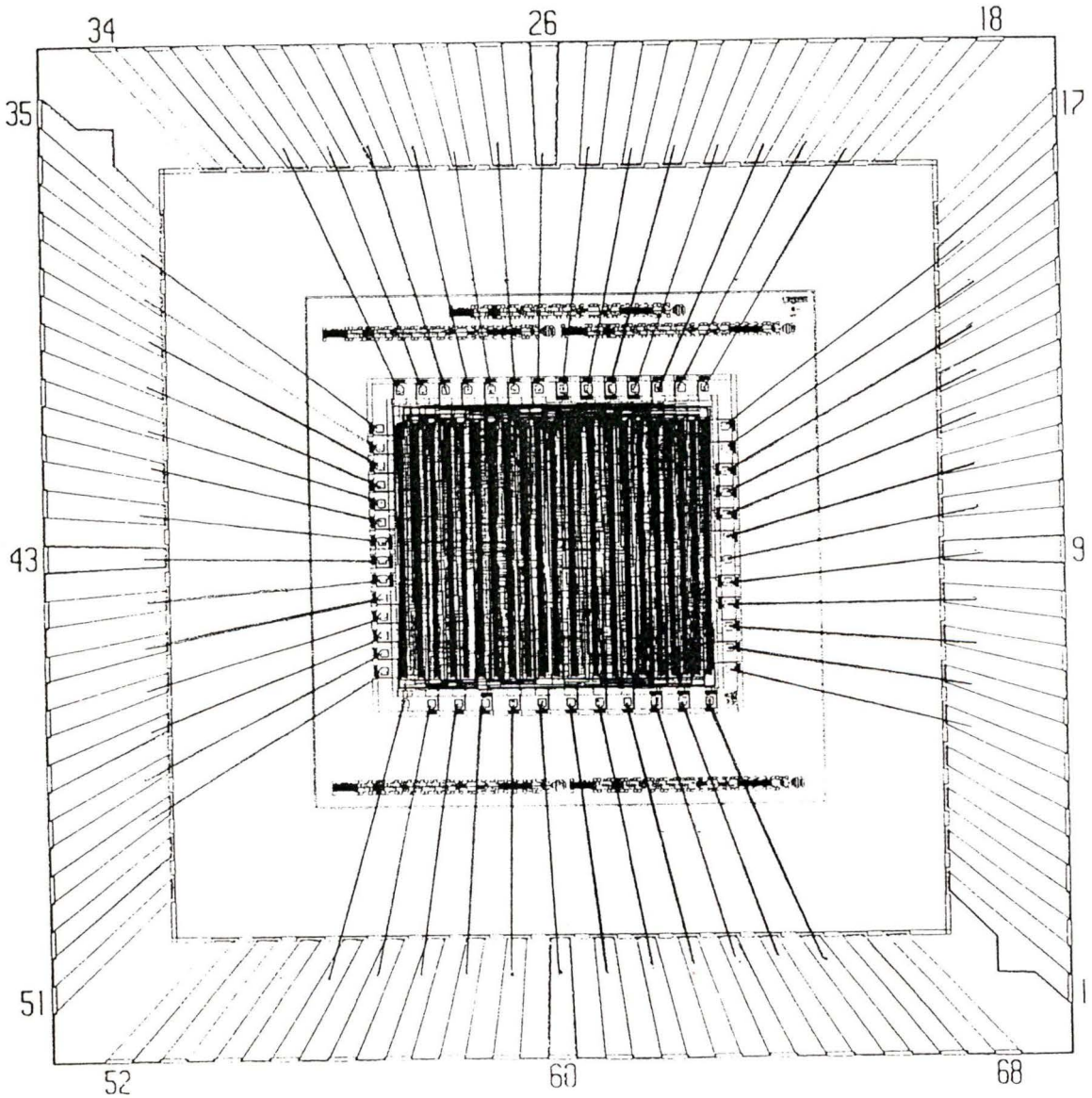
WAFER NUMBERS

DESIGN FILE REFERENCE

PACKAGE	KD84840A			LID	C-804		
WIRE ALLOY	99% Al/1% Si	DIA.	.001"	ELONG.	1.5 - 4%	T.S.	14-16 gms
	99% Al/1% Si		.00125"		1.5 - 4%		18-22 gms
D/A PREFORM	ALLOY 98% Au/2% Si	RECOMMENDED SIZE				W/B METHOD	U.S.

BONDING DIAGRAM

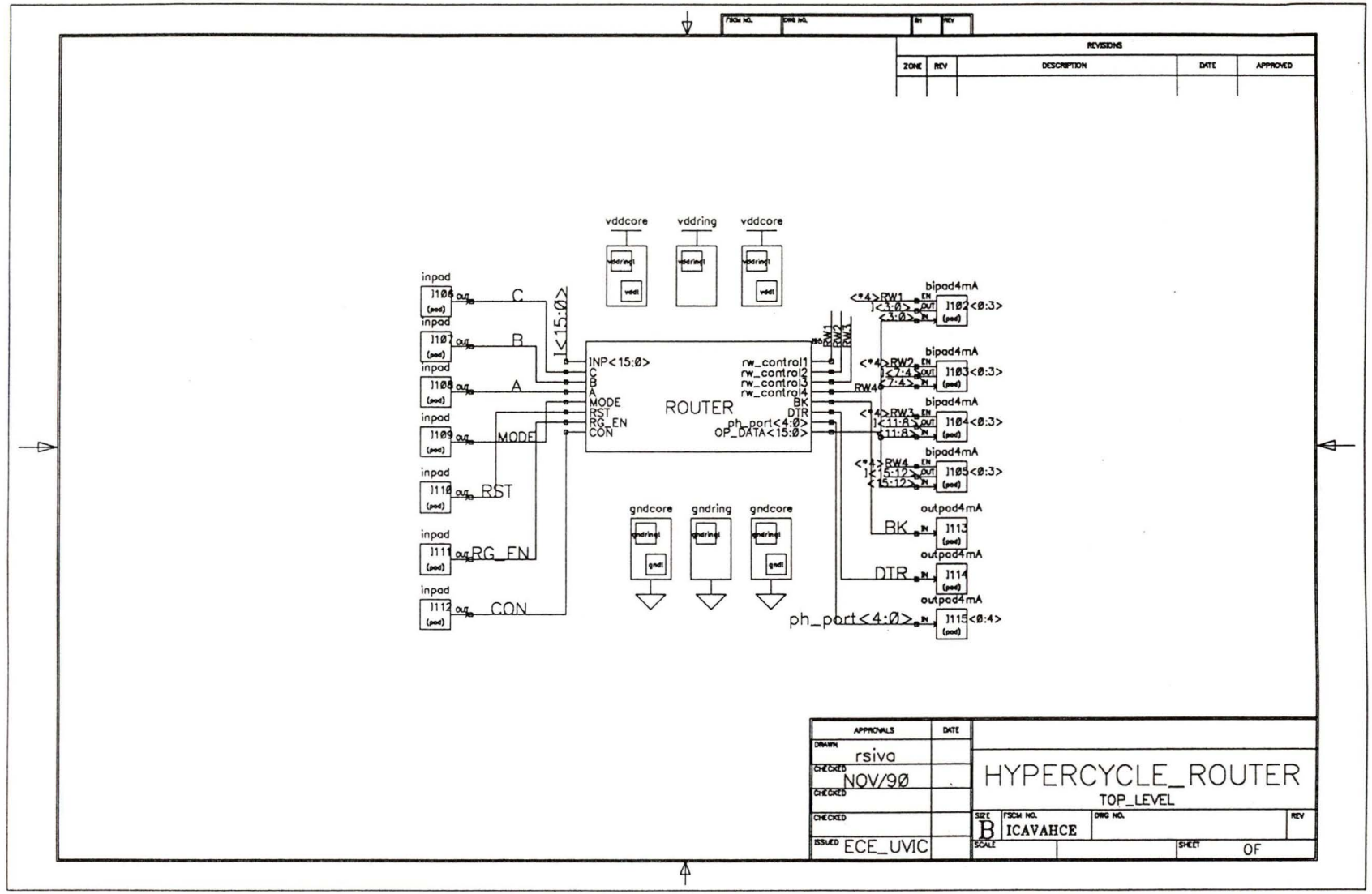
NOTES: 1. DIE ATTACH PAD SIZE: .400 X .400
2. ZERO GROUND

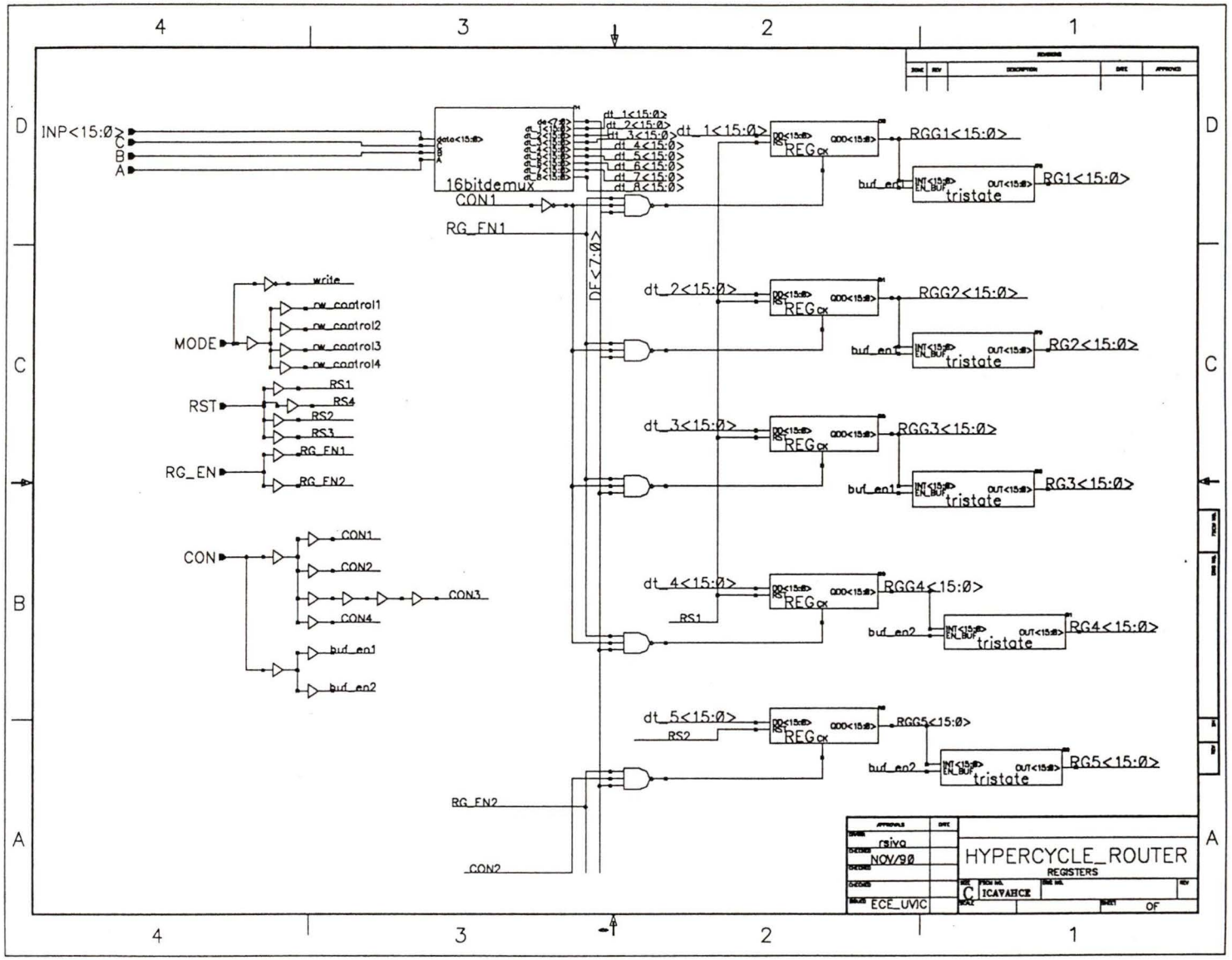


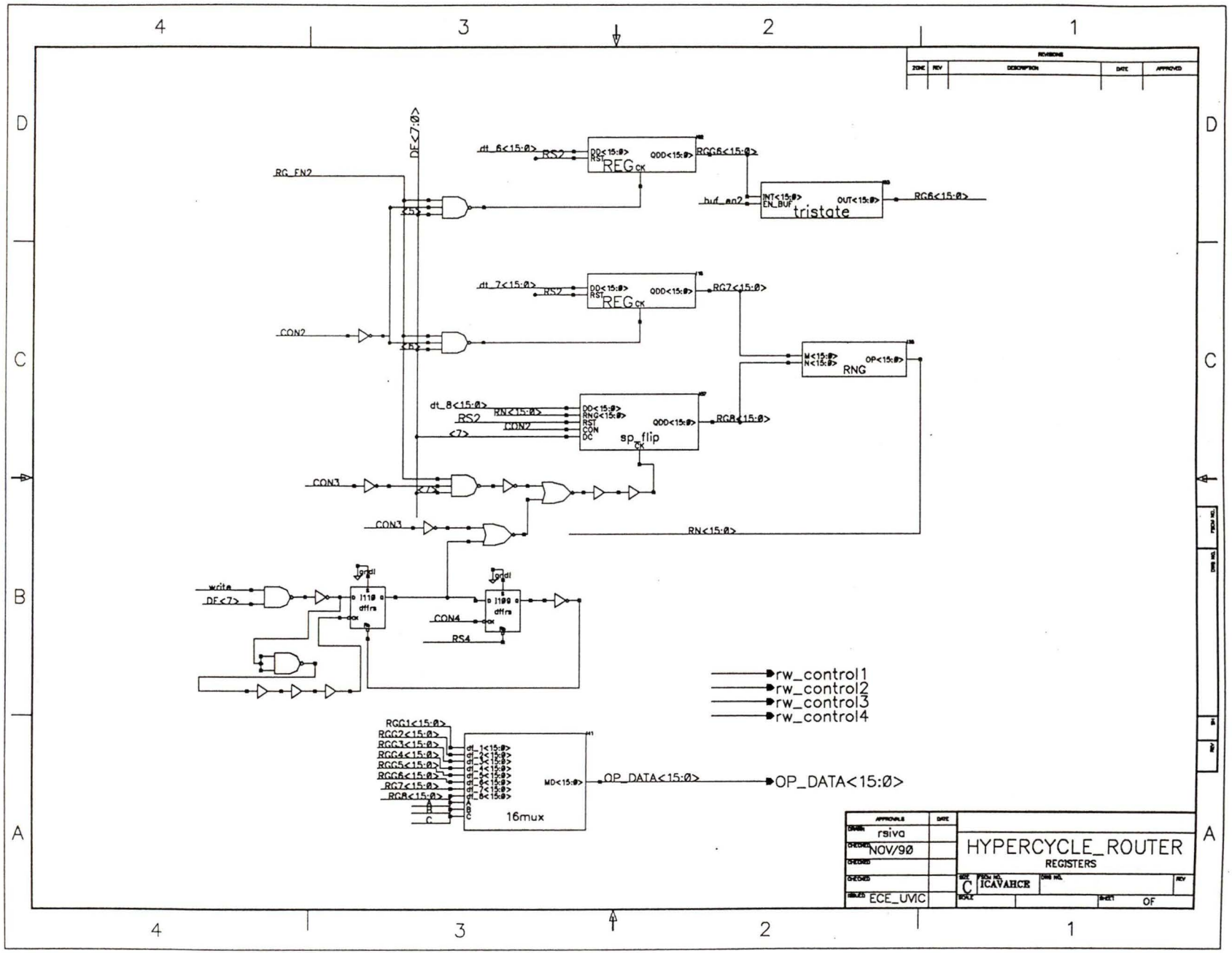
Appendix D

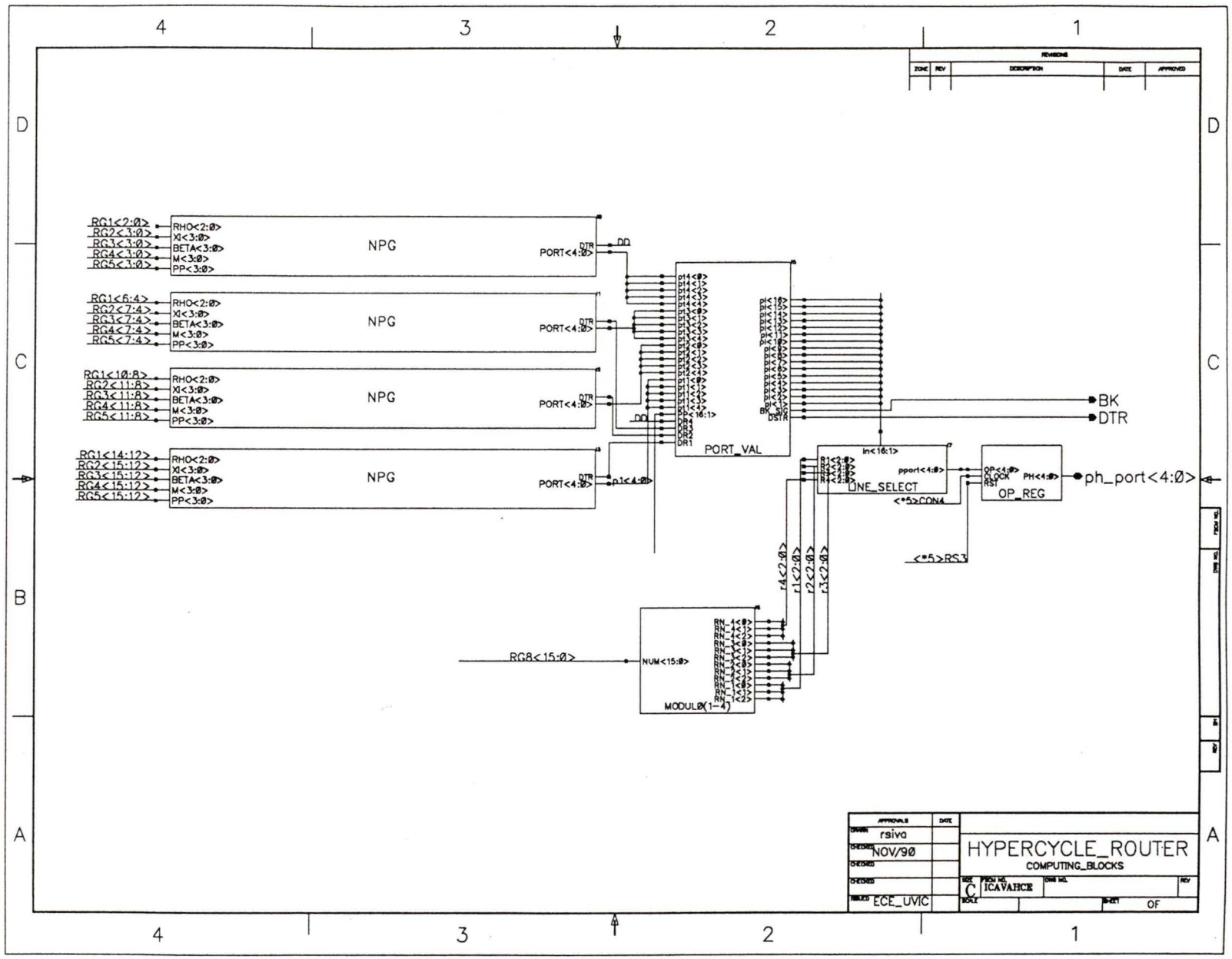
Schematic Diagrams of Hypercycle Router

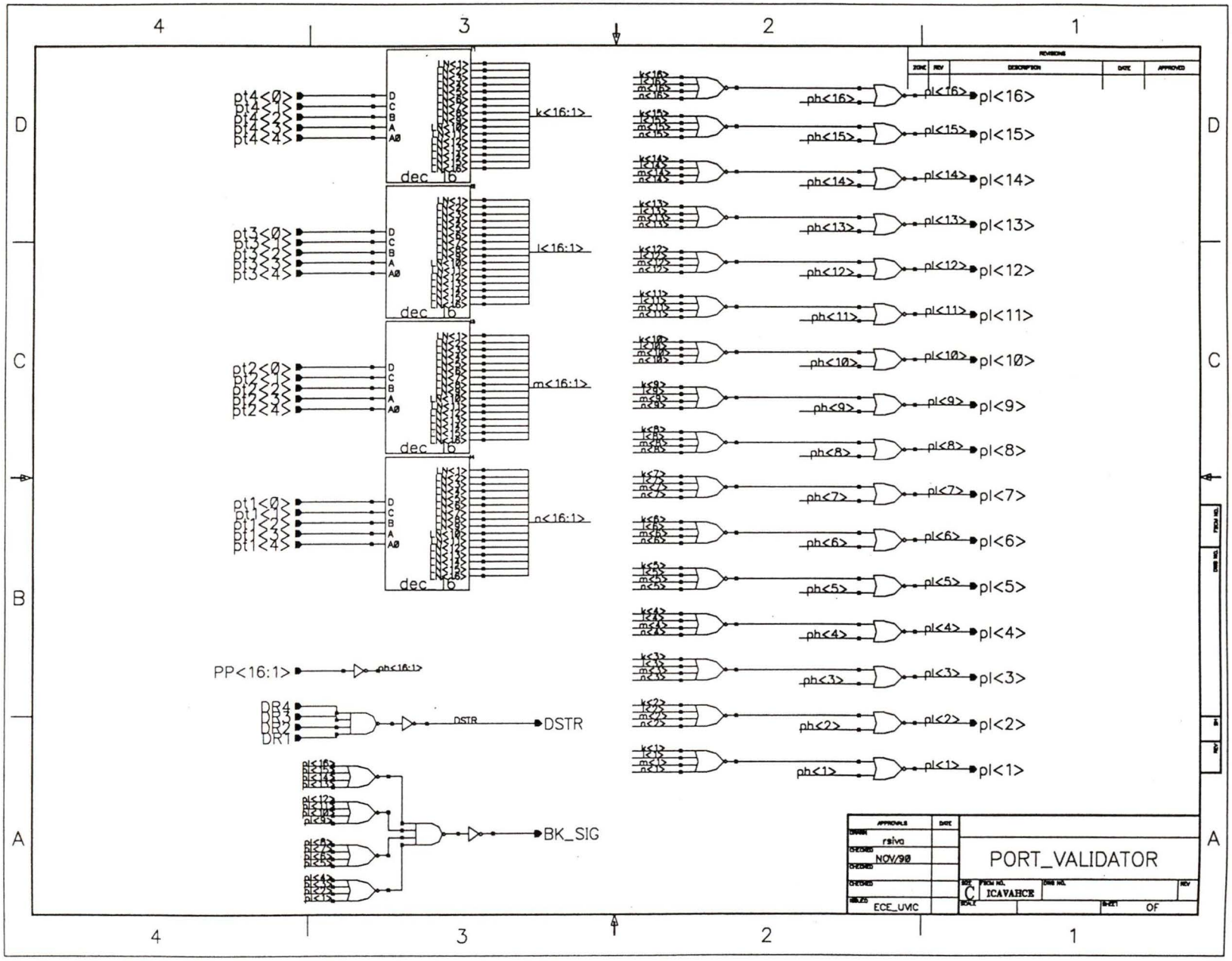
The schematic capture of the Hypercycle Router along with the bonding diagrams of the circuit implemented in 1.2 μ CMOS4S technology are shown in pages 199-204.


FSCM NO.
DWG NO.
SH
REV
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
vddcore
vddring
vddcore
inpad
J106
C
J107
B
J108
A
J109
MODE
J110
RST
J111
RG_EN
J112
CON
I<15:0>
INP<15:0>
C
B
A
MODE
RST
RG_EN
CON
ROUTER
rw_control1
rw_control2
rw_control3
rw_control4
BK
DTR
ph_port<4:0>
OP_DATA<15:0>
RW1
RW2
RW3
RW4
bipod4mA
J102<0:3>
J103<0:3>
J104<0:3>
J105<0:3>
outpad4mA
BK
J113
DTR
J114
ph_port<4:0>
J115<0:4>
gndcore
gndring
gndcore
APPROVALS
DATE
DRAWN
rsiva
CHECKED
NOV/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
HYPERCYCLE_ROUTER
TOP_LEVEL
SIZE
B
FSCM NO.
ICAVAHCE
DWG NO.
REV
SCALE
SHEET
OF


INP<15:0>
C
B
A
16bitdemux
CON1
RG_EN1
DF<7:0>
dt_1<15:0>
dt_2<15:0>
dt_3<15:0>
dt_4<15:0>
dt_5<15:0>
dt_6<15:0>
dt_7<15:0>
dt_8<15:0>
REG
RGG1<15:0>
RGG2<15:0>
RGG3<15:0>
RGG4<15:0>
RGG5<15:0>
buf_en1
buf_en2
tristate
RG1<15:0>
RG2<15:0>
RG3<15:0>
RG4<15:0>
RG5<15:0>
MODE
write
ow_control1
ow_control2
ow_control3
ow_control4
RST
RS1
RS2
RS3
RS4
RG_EN
RG_EN1
RG_EN2
CON
CON1
CON2
CON3
CON4
buf_en1
buf_en2
RG_EN2
CON2
rsivg
NOV/90
ECE_UVIC
HYPERCYCLE_ROUTER
REGISTERS
ICAVAHICE
SIZE
C
SHEET
OF


4
3
2
1
D
C
B
A
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
DE<7:0>
RG_EN2
dt_6<15:0>
RS2
REG
RGG6<15:0>
hiZ_en2
tristate
RG6<15:0>
dt_7<15:0>
RG7<15:0>
CON2
RNG
dt_8<15:0>
RN<15:0>
RS2
CON2
<7>
sp_flip
RG8<15:0>
CON3
CON3
RN<15:0>
write
DE<7>
dffrs
CON4
RS4
rw_control1
rw_control2
rw_control3
rw_control4
RGG1<15:0>
RGG2<15:0>
RGG3<15:0>
RGG4<15:0>
RGG5<15:0>
RGG6<15:0>
RG7<15:0>
RG8<15:0>
16mux
OP_DATA<15:0>
OP_DATA<15:0>
APPROVALS
DATE
DRAWN
rsiva
CHECKED
NOV/90
CHECKED
CHECKED
ISSUED
ECE_UVIC
HYPERCYCLE_ROUTER
REGISTERS
SIZE
C
FSCM NO.
ICAVAHCE
DWG NO.
REV
SCALE
SHEET
OF


RG1<2:0>
RG2<3:0>
RG3<3:0>
RG4<3:0>
RG5<3:0>
RHO<2:0>
XI<3:0>
BETA<3:0>
M<3:0>
PP<3:0>
NPG
PORT<4:0>
DTR
DQ
RG1<6:4>
RG2<7:4>
RG3<7:4>
RG4<7:4>
RG5<7:4>
RG1<10:8>
RG2<11:8>
RG3<11:8>
RG4<11:8>
RG5<11:8>
RG1<14:12>
RG2<15:12>
RG3<15:12>
RG4<15:12>
RG5<15:12>
p1<4:0>
PORT_VAL
BK
DTR
ONE_SELECT
OP_REG
ph_port<4:0>
<*5>CON4
<*5>RST
r4<2:0>
r1<2:0>
r2<2:0>
r3<2:0>
RG8<15:0>
NUM<15:0>
MODUL8(1-4)
rsiva
NOV/90
ECE_UVIC
HYPERCYCLE_ROUTER
COMPUTING_BLOCKS
ICAVAHCE


4
3
2
1
D
C
B
A
pt4<0>
pt4<1>
pt4<2>
pt4<3>
pt4<4>
D
C
B
A
A0
dec 16
k<16:1>
pt3<0>
pt3<1>
pt3<2>
pt3<3>
pt3<4>
l<16:1>
pt2<0>
pt2<1>
pt2<2>
pt2<3>
pt2<4>
m<16:1>
pt1<0>
pt1<1>
pt1<2>
pt1<3>
pt1<4>
n<16:1>
PP<16:1>
ph<16:1>
DR4
DR3
DR2
DR1
DSTR
BK_SIG
ph<16>
ph<15>
ph<14>
ph<13>
ph<12>
ph<11>
ph<10>
ph<9>
ph<8>
ph<7>
ph<6>
ph<5>
ph<4>
ph<3>
ph<2>
ph<1>
pl<16>
pl<15>
pl<14>
pl<13>
pl<12>
pl<11>
pl<10>
pl<9>
pl<8>
pl<7>
pl<6>
pl<5>
pl<4>
pl<3>
pl<2>
pl<1>
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
APPROVALS
DATE
DRAWN
rsilva
NOV/90
ECE_UVIC
PORT_VALIDATOR
SIZE
C
SCALE
PROJ NO.
ICAVAHCE
DWG NO.
REV
SHEET
OF

CMC MULTIPROJECT BONDING DIAGRAM

RETICLE CODE V44A1A

BRAND ID LABEL ICA
VAHCE

OTHER IDENTIFICATION FEATURES

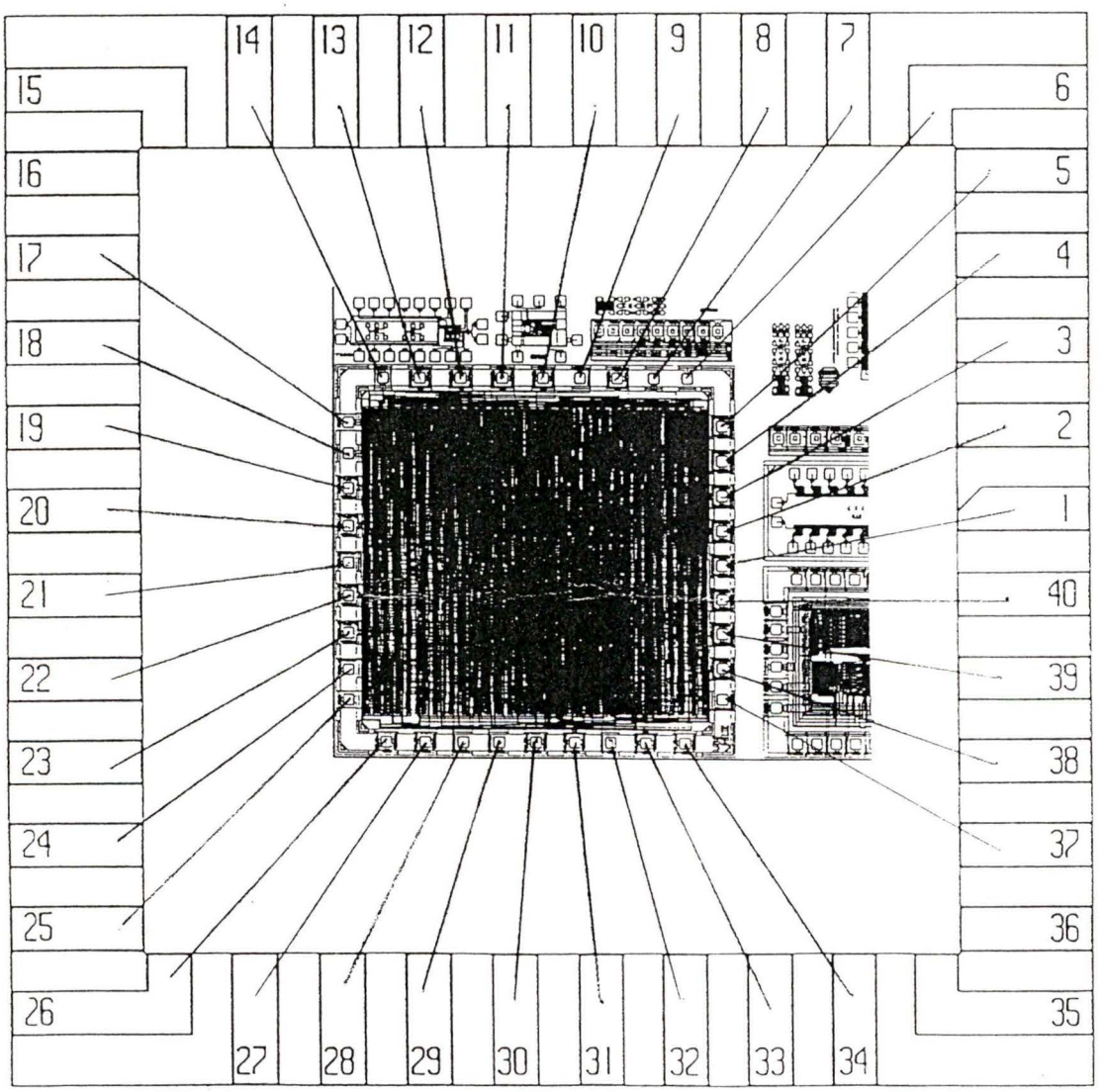
WAFER NUMBERS

DESIGN FILE REFERENCE

PACKAGE	IDK40F1-192G		LID	C-493-175-35M	
WIRE ALLOY	99% Al/1% Si	DIA.	.001"	ELONG.	1.5 - 4%
	99% Al/1% Si		.00125"		1.5 - 4%
				T.S.	14-16 gms
					18-22 gms
D/A PREFORM	ALLOY 98% Au/2% Si	RECOMMENDED SIZE		W/B METHOD	U.S.

BONDING DIAGRAM

NOTES: 1. DIE ATTACH PAD SIZE: .400 X .400
2. ZERO GROUND



VITA

Surname:	Radhakrishnan	Given Names:	Sivakumar
Place of Birth:	Nagercoil, India	Date of Birth:	5 June 1967

Educational Institutions Attended:

University of Victoria	1989 to 1991
University of Madras, India	1985 to 1989

Degrees Awarded:

B. Eng. University of Madras 1989

Publications:

1. N. J. Dimopoulos, D. Radvan, R. Sivakumar, "Implementation of the Backtrack-to-the Origin-and-Retry routing of the Hypercycle based Interconnection Networks", *Proceedings of the Canadian Conference on Electrical & Computer Eng.*, Montreal, Canada, Sept. 1990, pp. 67.2.1-67.2.4.

2. R. Sivakumar, N. J. Dimopoulos, K. F. Li "VLSI Design of a Modulo-Extractor", *Proceedings of the 1991 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Victoria, Canada, vol 1. May 1991, pp. 327-330.

3. N. J. Dimopoulos, R. Sivakumar, K. F. Li, V. Dimakopoulos, M. Choudhary and D. Radvan, "Hypercycles: A Status Report", *Proceedings of the 1991 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Victoria, Canada, vol. 1, May 1991, pp. 111-114.

4. N. J. Dimopoulos, R. Sivakumar, D. Radvan, "Routing and Processor Allocation on a Hypercycle based Multiprocessor", to be presented in *The 1991 International Conference on Supercomputing*, Cologne, Germany, June 17-21, 1991.

5. N. J. Dimopoulos, R. Sivakumar, D. Radvan, "Implementation of Routing Engine for Hypercycle based Interconnection Networks", submitted for presentation in *The 1991 Canadian Conference on VLSI*, Kingston, Canada, Aug 25-27, 1991.

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis: VLSI IMPLEMENTATION OF A ROUTER FOR THE BACKTRACK-TO-THE-ORIGIN-AND-RETRY-ROUTING SCHEME OF THE HYPERCYCLE BASED INTERCONNECTION NETWORKS

Author: [redacted]

(Signature)

SIVAKUMAR RADHAKRISHNAN

(Name in Block Letters)

June 14, 1991

(Date)