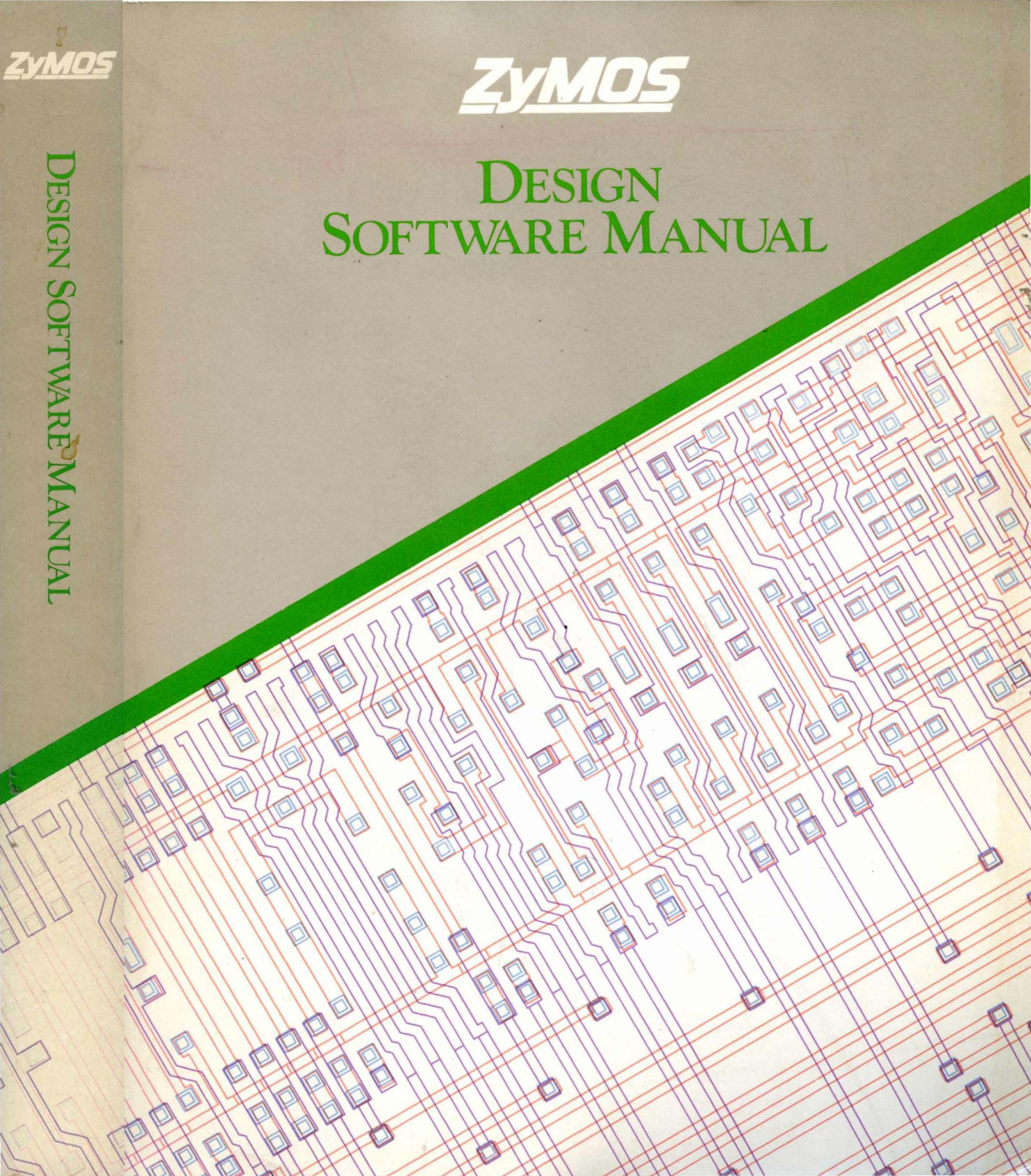
ZyMOS

DESIGN SOFTWARE MANUAL

ZyMOS

DESIGN SOFTWARE MANUAL



DESIGN SOFTWARE MANUAL

1985 EDITION

PUBLISHED BY
ZyMOS CORPORATION
477 N. MATHILDA AVENUE
SUNNYVALE, CA 94086

ZyP DESIGN SYSTEM INTRODUCTION

Doc: 20-010-001 Rev: G | Issued: August 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800 TWX 910-339-9530 ZYMOS SUVL

List of effective pages

PAGE	REVISION
Title	Rev: G
Table of Contents	Rev: G
Pages 1-1 through 1-2	Rev: G
Pages 2-1 through 2-3	Rev: G
Page 3-1	Rev: G
Pages 4-1 through 4-2	Rev: G
Pages 5-1 through 5-3	Rev: G
Page 6-1	Rev: G
Page 7-1	Rev: G
Page 8-1	Rev: G

TABLE OF CONTENTS

1 OVERVIEW

Welcome to ZyP. ZyP is a state-of-the-art, standard cell, computer aided design system specifically developed by ZyMOS for the design of custom circuits. ZyP is a user friendly software system which includes application rograms and libraries, linked through an integrated data base, to facilitate custom circuit development. ZyP has been operational since 1981 and includes many advanced features and technologies such as memory compilers, core microprocessors, analog MOS, and CHMOS. All ZyP applications software has been developed by ZyMOS specifically for standard cell design. This software is readily available to users via time-share facilities and is also available under license.

Central to the ZyP system is a library of standard cells. Electrical and topological cell data is manipulated by ZyP to verify network performance and to generate network artwork used in the custom circuit manufacturing cycle.

Standard cells offer advantages by being pre-defined. One important advantage is the ability to electrically characterize cell performance at the silicon level. When a set of cells is used in a custom circuit, the electrical data is used to verify cell operation in the network environment and accurately simulate silicon level network performance.

opment time and cost. But in addition, it offers
the flexibility required to maintain a competitive edge. No restrictions on number or types of cells incorporated in a specific circuit is imposed. No new structured design methodology or actual silicon layout design will be required or implied. Rather, the user is supported, at his own facility, at low cost, without the need for special equipment, in the design of systems. ZyP translates systems to silicon.

Cells range from simple to complex, standard to custom, analog to digital. Each has a detailed data sheet and each is incorporated in a number of different applications libraries. Cells are provided in the basic ZyMOS CMOS production technologies,

- Zy40000 Silicon Gate CMOS
- Zy50000 Silicon Gate CHMOS

Extensive ZyP documentation has been developed to assist users in understanding the ZyP system, learning how to apply ZyP to custom circuit development, and as reference documents. ZyP documentation is routinely updated and enhanced. We welcome your suggestions and comments.

A functional ZyP overview is presented in Figure 1. At the top of the figure, a graphical presentation is made of ZyPNET. ZyPNET is an international data network used to access the ZyP design system. The lower portion of Figure 1 shows the software programs and application libraries which comprise the ZyP system. Circuit simulation (ZySPICE), logic simulation (ZyPSIM), artwork generation (ZyPART), and test program generation (ZyTEST) software packages are linked via an integrated data base to cell libraries and device models.

Initially, users will spend most of their time studying cell data sheets and describing networks to the logic simulator. ZyMOS will transfer the network file, after verification, over ZyPNET to complete the composite generation and test program.

The overall objective of ZyP is to provide the tools, techniques, and design technology data base required to successfully design custom CMOS integrated circuits. There are few restrictions, but there are rules and the need for good design practices. The designer must invent, innovate, and interpret - ZyP will analyze, verify, and check.

The remaining sections of the ZyP System Introduction summarize the ZyP system documentation, ZyPNET, applications software, how to get started, and how to complete a design.

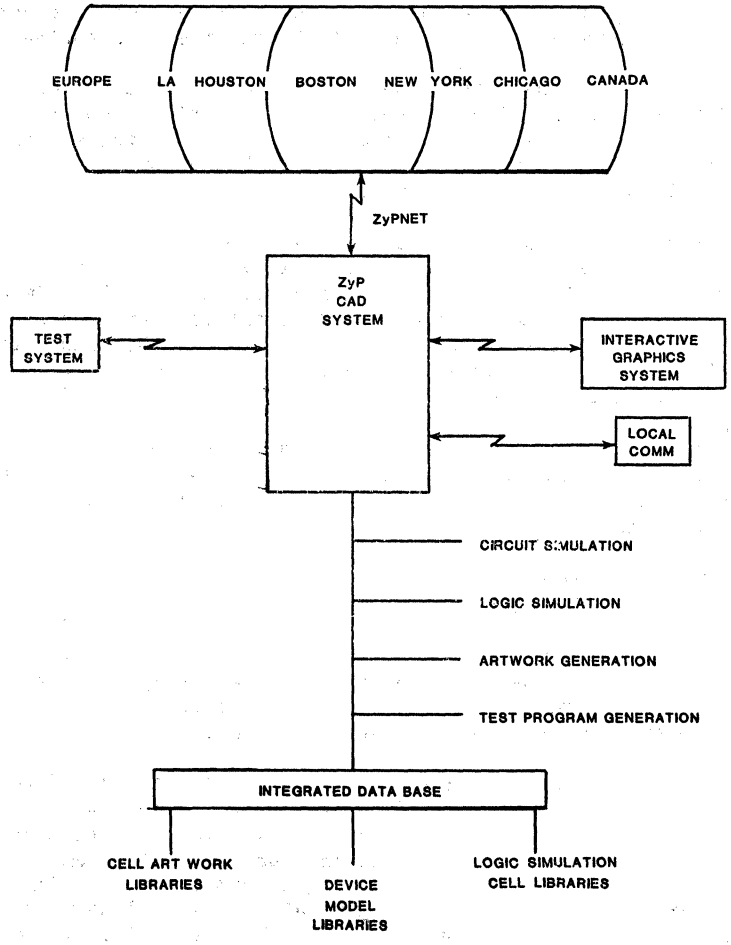


FIGURE 1. ZyP SYSTEM OVERVIEW

2 ZyP SYSTEM DOCUMENTATION

ZyP system documentation is organized into various documents, some of which have several volumes. The documents are numbered 20-010-1XX and each focuses on a specific area of ZyP. The beginning user should read this Introduction and the ZyP Preparation Guide in 20-010-101, study ZyPNET VAX or Prime for accessing ZyP, study ZyPSIM in 20-010-109, and review the desired cell library.

The following is a summary description of the ZyP documents.

2.1 ZyP DESIGN SYSTEM INTRODUCTION, DOC: 20-010-101

The ZyP Design System Introduction provides an overview of the ZyP system with summaries of the system software. It also includes a ZyP Preparation Guide with guidelines for starting your ZyP design.

2.2 ZyPNET PRIME USER'S MANUAL, DOC: 20-010-103

The ZyPNET Prime User's Manual includes an introduction on accessing ZyP on the ZyMOS Prime computer. Also included are the Prime User's Guide, PRIMOS operating system reference guide, and the PRIMOS editor reference guide.

2.3 Zy40000 SERIES CMOS CELL LIBRARY, DOC: 20-010-107

The Zy40000 Series CMOS Cell Library contains silicon gate CMOS design data, examples, and cell library data sheets.

2.4 ZySPICE USER'S MANUAL, DOC: 20-010-108

The ZySPICE User's Manual includes an overview on using ZySPICE, a description of the ZyMOS process models, and the ZySPICE reference manual.

2.5 ZyPSIM USER'S MANUAL, DOC: 20-010-109

The ZyPSIM User's Manual includes an overview of the ZyMOS generated ZyPSIM MOS logic simulator and the ZyPSIM reference manual. A document on the ZyPSIM Test Language is also included.

2.6 ZyP SPECIFICATION AND TEST USER'S MANUAL, DOC: 20-010-110

The ZyP Specification and Test User's Manual provides ZyP device specification forms, a user's guide on design criteria and requirements for test, and the ZyTEST User's Manual. Also provided is the ZySPEC User's Guide.

2.7 Zy40000 SERIES ANALOG CMOS CELL LIBRARY, DOC: 20-010-111

The Zy40000 Series Analog Cell Library contains design data, examples, and cell data sheets for the analog cell library. Analog design is one of the advanced ZyP features and enables integration of mixed analog and digital subsystems.

2.8 ZyCOMP-4 USER'S MANUAL, DOC: 20-010-112

The ZyCOMP-4 User's Manual contains a detailed description of the ZyCOMP-4 4 bit core processor cell, the ZYCOMP4 cross assembler, and the ZyCOMP-4 Emulator Board User's Guide. ZyCOMP-4 is a general purpose 4 bit microprocessor implemented in the Zy40000 library. It has advanced architectural features including eight memory reference addressing modes, two index registers, and vectored interrupts. When integrated with RAM, ROM, and I/O cells, it becomes a powerful customer specific single chip computer.

2.9 ZyPLIB - MEMORIES, DOC: 20-010-114

The ZyPLIB - Memory manual contains design data, examples, and cell data sheets for generating RAM, ROM, and PLA component macro cells. The ZyMEM User's Guide describes the procedure for memory element compilation. These memory 'super cells' can be combined with random logic, ZyCOMP-4, and other 'super cells' to integrate complex subsystems.

2.10 ZyPART USER'S MANUAL DOC: 20-010-115

The ZyPART User's Manual contains user guides for the collection of composite generation software referred to as ZyPART. This document is only available with installations.

2.11 ZyP UTILITIES, DOC: 20-010-116

The ZyP Utilities Manual contains user guides for general ZyP utility programs. These include ZyPEE (ZyP Error Eliminator), MAILBOX, ZL, and various operating utilities.

2.12 Zy50000 SERIES CHMOS CELL LIBRARY, DOC: 20-010-117

The Zy50000 Series CHMOS Cell Library contains design data, examples, and cell library data sheets for the Zy50000 CHMOS process.

2.13 ZyPNET VAX USER'S MANUAL, DOC: 20-010-118

The ZyPNET VAX User's Manual includes an introduction on accessing ZyP on the ZyMOS VAX computer and the VAX/VMS Primer.

2.14 Zy80C49 USER'S MANUAL, DOC: 20-010-119

The Zy80C49 User's Manual contains a detailed description of the 80C49 core microprocessor cell. The Zy80C49 is a true mask compatible cell version of the standard 80C49 microcomputer. It is implemented in Zy50000 CHMOS technology and is fully compatible with the Zy50000 Series CHMOS Cell library as well as the 8749 EPROM device.

2.15 ZyP MACRO'S MANUAL, DOC: 20-010-121

The ZyP MACRO'S manual contains design information, examples, and datasheets for the ZyP Predefined Macro's library. This library contains logic equivalent functions for the 7400 series TTL and 4000B series CMOS standard product IC's in both Zy50000 CHMOS and Zy40000 CMOS technologies.

3 ZyPNET

The term ZyPNET refers to the data network and computer resource used to access and execute ZyP designs. ZyPNET references the network developed by ZyMOS but it can also reference a network established for a specific company. The terms ZyPNET VAX and ZyPNET Prime refer specifically to the VAX and Prime versions of ZyPNET and both are currently available at ZyMOS.

3.1 ZyPNET DATA NETWORK

The ZyPNET data network is implemented via Tymnet at 300 and 1200 baud. Accessing the system requires only a user terminal, modem, and printer.

3.2 ZyPNET VAX

The ZyMOS VAX computer system is available to ZyP users for executing ZyP designs in a time share environment. Users have access to the various ZyP applications programs, libraries, and utilities. Upon completion of a design, ZyMOS will transfer the user's design files to an in-house directory for artwork (ZyPART) generation and test program completion.

Document 20-010-118 provides detailed information on ZyPNET VAX.

3.3 ZyPNET PRIME

The ZyMOS Prime computer system is also available to ZyP users for executing ZyP designs in a time share environment. Selection between VAX and Prime is at the user's discretion. Users have access to the various ZyP applications programs, libraries, and utilities. Please contact ZyMOS as to the availability of specific software modules or releases.

Document 20-010-103 provides detailed information on ZyPNET Prime.

4 APPLICATION SOFTWARE SUMMARY

ZyP is a software system which links, via an integrated data base, various software modules and cell oriented applications libraries for the purpose of verifying designs and automating the artwork and test program generation process. The following is a summary of the basic software modules. Many additional software modules are available such as ZyPEE, ZyMEM and ZySPEC, which are described elsewhere in the ZyP documentation.

4.1 ZyPSIM

ZyPSIM is a ZyMOS proprietary general purpose MOS logic simulator specifically developed for the ZyP system. It is supported with standard cell model libraries enabling networks of standard cells to be simulated based on actual standard cell performance.

The logic simulator is the primary interface between the user and the ZyP system. Simulation of a network is the main design verification tool for ZyP based designs. Accurate simulation of a network of standard cells guarantees IC functionality and establishes IC dynamic performance capability. In addition to logic simulation, the ZyPSIM network file is used as the source data file for artwork generation. The simulation output file is used for test program development. ZyPSIM supports numerous sophisticated design features including hierarchical design, artwork capacitance estimation, and guardbanding over different operating conditions.

Document 20-010-109 provides detailed information on ZyPSIM.

4.2 ZySPICE

ZySPICE is a ZyP variant of the UC Berkeley generated SPICE circuit simulator. It includes model modifications and model files for the ZyMOS production processes. ZySPICE supports DC and TRANSIENT analysis of transistor level circuits.

Document 20-010-108 provides detailed information on ZySPICE.

4.3 ZyPART

ZyPART is a set of ZyMOS proprietary programs which convert a logic simulator network file to circuit artwork used in the tooling generation cycle.

Document 20-010-115 provides detailed information on ZyPART.

4.4 ZyTEST

ZyTEST is a set of ZyMOS proprietary software modules which process the ZyPSIM output file, translate the data to Fairchild Sentry compatible code, and compact the file to minimize test vectors. ZyTEST also allows users to compare simulation output files to verify design guardbanding for voltage and temperature extremes.

Document 20-010-110 provides detailed information on ZyTEST.

4.5 ZyP SOFTWARE UTILITIES

ZyPEE: ZyPEE is the ZyP Error Eliminator, it is the ZyMOS expert system software. ZyPEE can analyze a circuit's network description and identify complex design errors with respect to IC design technique, IC testing, mask generation, and production.

ZyMEM: ZyMEM is a powerful memory compiler program enabling the user to automatically generate RAM, ROM, and PLA circuits. From a high level organizational description (ex; # of words, # of bits/word, etc.) ZyMEM will create the logic coding, mask information, programming files, and datasheets, for the user's specific memory.

ZL: ZL provides Prime user's the capability to display the ZyPSIM Logic Simulation Models for all standard cells.

ZySPEC: ZySPEC will automatically generate a circuit specification for a ZyP standard cell IC. This specification fully defines a ZyP circuit for production and test.

BUGME: BUGME is a convenient tool for ZyPNET Prime users for reporting and documenting the occasional ZyP system software bug. It automatically notifies the ZyP Design Support staff of your need for assistance.

4.6 MICROPROCESSOR DEVELOPMENT AIDS

ZyMOS offers numerous development tools in support of our microprocessor core cells. Included are assemblers such as the ZyPCOMP4 (ZyCOMP4 uP), and the ASM49 (80C49), data transfer and formatters such as ZyPROM and ZyCE, and hardware emulator boards for use with commercially available development systems.

5 GETTING STARTED

Having gained some familiarity with what ZyP is about, the next step is to start applying the system. It is recommended that beginning users follow the procedure outlined below.

5.1 COMPUTER SYSTEM

ZyP is available on ZyPNET VAX or Prime or an equivalent in house computer system. Having defined your host system, review the following information in the appropriate document.

5.1.1 OPERATING SYSTEM

The computer operating system is a general purpose utility that assigns resource (eg storage) and applications software to the user. For VAX, the operating system is called VMS and for Prime, it is called PRIMOS.

The introductory document for VMS is the VAX/VMS Primer located in document 20-010-118. The introductory document for PRIMOS is The Prime User's Guide, located in document 20-010-103. These introductory guides should be studied to familiarize the user with the appropriate operating system.

5.1.2 FILE STRUCTURE

Files in general are lines (or strings) of text which have some collective meaning and must be preserved. For example, a network description is a file. VMS and PRIMOS assume a certain file structure which should be understood and is discussed in the introductory guides referenced above.

5.1.3 EDITOR

Editors are used for creating and modifying files. Once the operating system and file structure are understood, the editor is invoked to create a new user file. The VMS editor EDT is described in the VAX/VMS Primer. This is a screen and line editor. To use the screen editor, you must have a VT100 terminal or equivalent. The line editor is compatible with most standard terminals. The PRIMOS editor is described in The New User's Guide to Editor and Runoff in document 20-010-103. This is a line editor only. New users should familiarize themselves with the creation, editing, and saving of files as described in the reference documents.

5.1.4 LOGGING IN

Once the items discussed above have been studied the user can access the target system by following the log-in procedure described in the Introduction section of the appropriate ZyPNET document. To access the system, you must have a password assigned by ZyMOS. It is recommended that you try logging in to understand the log-in procedure and verify your password. It is also suggested that you create, save, and reaccess a file using the editor.

ZyPNET VAX and Prime are available to users over Tymnet. The user should have a 24 line, 80 character per line terminal and a 300 or 1200 baud modem. The terminal should be configured for 8 bit word length, 1 stop bit, no parity, and full duplex. The modem should be Bell 103 compatible at 300 baud or Bell 212A at 1200 baud. A printer connected to the terminal printer port is also a necessity.

Contact ZyMOS ZyP Applications for Tymnet and ZyPNET access codes as well as assistance with any login problems.

5.2 CELL LIBRARIES

Having gained an understanding of operating systems and accessing ZyP, the next step is to review the cell libraries and generic data for the desired technology. The standard ZyP technologies are as follows.

5.2.1 Zy40000 SERIES CMOS CELL LIBRARY

The Zy40000 silicon gate CMOS technology is a high performance isoplanar bulk CMOS process using 5u design rules. It provides approximately twice the performance of metal gate CMOS with a 40% increase in density. This process operates from 1.1 to 6 volts, -55 to 125 degree C. Document 20-010-107 provides cell data sheets and process generic data.

5.2.2 Zy50000 SERIES CHMOS CELL LIBRARY

The Zy50000 silicon gate CHMOS technology is a high performance, N-well, CMOS process. It provides up to a 40% increase in performance over the 5u CMOS process with a substantial increase in density. The CHMOS process operates from 2 to 6 volts, -55 to 125 degree C. Document 20-010-117 provides cell data sheets and process generic data.

5.3 SIMULATION

The ZyP system provides simulation software design aids for the circuit designer. The simulators use predefined process models (ZySPICE) and cell models (ZyPSIM) to simplify the designer's task and permit him to concentrate on design while the computer automates analysis. The ZyP simulators are the primary user interfaces to the ZyP system. Two ZyP manuals are provided as tutorial and reference documents on using the simulators. The user should study the appropriate documents and become familiar with the capabilities, commands, files, and operation of the simulators for application to the design task.

5.4 ADDITIONAL DETAILS

A ZyP Preparation Guide is available which provides background information along with a simple, detailed example of the procedure for beginning a ZyP IC design. The ZyP Preparation Guide is located in the ZyP Design System Introduction, Doc: 20-010-101.

6 DESIGN COMPLETION

One of the primary advantages of ZyP design is being able to routinely integrate full custom designs successfully, quickly, and economically. ZyMOS has established a number of formalized procedures and forms to support design completion. The forms (with the exception of the ZyP device specification) are located in the Appendix and should be copied as needed. Each is summarized below.

6.1 SPECIAL CELL DEFINITION

The user should make every effort to utilize existing standard cells to complete the design. However, specific designs may require cells which are not available. ZyMOS designates these as special cells, where a special cell is either a standard cell not currently in the library or a custom cell. The Special Cell Definition form is provided to allow users to define special cells.

6.2 ZyP DEVICE SPECIFICATION

ZyP device specifications are generated for the user by the application program ZySPEC. ZySPEC should be run at the conclusion of the design to generate a final IC specification document.

6.3 ZyP FILE TRANSFER CHECKLIST

The ZyP File Transfer Checklist is the formal transfer form. Acceptance of this form by the user and ZyMOS initiates the integration process and guarantees proper completion of all necessary tasks.

6.4 ZyP BUG REPORT

ZyMOS has extensively tested all applications software and libraries. However, we recognize that with a design system of the complexity of ZyP, bugs may occur. The ZyP online utility, BUGME, is provided to assist the reporting of bugs as well as their correction. The ZyP Bug Report form in the appendices is an overlay of the report generated by BUGME.

7 SUMMARY

ZyP opens a new dimension in custom circuit development. It assists design, tooling, and production. Production is guaranteed. Pricing is determined up front. Alternate technologies are available. Access is worldwide. Welcome to the system which is setting the standard in custom VLSI.

ZyMOS is committed to ZyP. We are developing new libraries, new technologies, and new software tools. ZyP is your long term VLSI solution.

8 APPENDICES

Forms described in section 6 follow and can be copied as needed.

ZyMOS CORPORATION
477 N. Mathilda Ave.
Sunnyvale, CA 94086

ZyP™ DESIGN SYSTEM

SPECIAL CELL DEFINITION

(Send c/o ZyP™ Marketing Mgr.)

COMPANY NAME: _____

ADDRESS: _____

PURCHASING CONTACT: _____ Tel# _____

TECH. CONTACT: _____ Tel# _____

PROCESS _____ OPERATING TEMP _____

OPERATING VOLTAGE _____ MAX. FREQUENCY _____

SPECIAL SPECS/REQUIREMENTS (Attach ZySPICE simulation for model generation)

_____

_____

CIRCUIT DIAGRAM (Show port diagram, logic diagram, and transistor level circuit diagram with device sizes. Use reverse side if necessary)

TO BE COMPLETED BY ZyMOS

MKT RECEIVED DATE: _____ BY: _____

ENGR IN DATE: _____ BY: _____

ENGR OUT DATE: _____ BY: _____

CELL TYPE: STANDARD ☐ CUSTOM ☐

CELL NAME: _____ MODEL AVAIL: _____ ARTWORK AVAIL: _____

ZyP™ ENGR APPROVAL: _____ ZyP™ SUPP. APPROVAL: _____

COMMENTS: _____

_____

_____

_____

CIRCUIT DIAGRAM:

ZyP FILE SUBMITTAL CHECKLIST

ZyMOS CORPORATION
477 N. Mathilda Ave.
Sunnyvale, CA 94086

ZyP DESIGN SYSTEM
(408) 730-8800

(Please send c/o ZyP Marketing Program Mgr.)

CUSTOMER NAME: _____ CUSTOMER CHIP NAME: _____

ADDRESS: _____

PURCHASING CONTACT: _____ TELEPHONE: _____

TECHNICAL CONTACT: _____ TELEPHONE: _____

STANDARD ZyP SPECIAL ZyP

Files Transferred: ___chip.SPC ___chip.COM

NET File Requirements

- ___No ZyPSIM primitives (except CAP)
- ___All telescoping cells defined as macros
- ___No node names greater than 23 characters
- ___.NORUN and .NOSPIKE included
- ___.INPUT included for capacitance estimation
- ___.PADS included for pinout
- ___.TPRINT statement - ordered pin list
- ___.GUARDBAND included
- ___ZyPEE verification complete
- ___ZySPEC run

COMMENT File Requirements

- ___Critical nets (capacitance, resistance, timing)
- ___Customer data
- ___Pin out options

Documentation

- ___ZyP format circuit schematic included (required)
- ___Customer functional circuit description (optional)

Test Requirements

- ___Simulation run at guardband extremes
- ___Total test time less than 5 seconds
- ___Total test vectors less than 4K

Special Requirements

- ___Special cell required
- ___Memory (RAM, ROM, PLA), analog cells, or core processor
- ___Non standard packaging
- ___Non standard testing

If any of the above items are checked, contact the ZyP Marketing Project Manager prior to net submission for schedule/cost information.

FILE ACCEPTANCE APPROVALS

Approval (ZyMOS to complete)

- ___ All output files regenerated from chip.NET and verified ___ ZyPEE verified
- ___ Guardbanding verified via ZyTEST over spec range
- ___ ZyP device specification approved by all departments

Signatures

CUSTOMER: _____ FILE SUBMIT DATE: _____

ZyMOS Engineer: _____ FILE ACCEPTANCE DATE: _____

ZyMOS Mkt.: _____ # Grids: _____ ZyMOS Device # _____

Rev: E 9/84

ZyP BUG REPORT

GENERAL INFORMATION

COMPANY: _____ DATE: _____

NAME: _____ PHONE NO. _____

SOFTWARE INFORMATION

Please check program(s) affected.

___ ZYPSIM ___ ZYPSIM MODELS ___ ZYSPICE ___ ZYSPICE MODELS
___ ZYTEST ___ ZYPART ___ ZYPEE ___ ZYP DOCUMENTATION
___ PRIMOS (Please specify REV # _____) ___ OTHER

INPUT FILES:
(.NET, etc.) _____

_____

OUTPUT FILES:
(.PRINT, etc.) _____

_____

EXPLANATION / COMMENTS

_____

_____

_____

_____

_____

_____

_____

_____

_____

URGENCY OF SOLUTION (Please check one)

___ Major bug : Has halted project progress, immediate fix.
___ Minor bug : Will prevent project completion, short term.
___ Miscellaneous bug: Error that must be corrected, non-critical.

FOR ZyMOS INTERNAL USE ONLY

Bug I.D. _____

To be fixed by: _____ Completion date: _____

Disposition of software affected: _____

_____

_____

RETURN THIS FORM TO: ZyMOS CORPORATION (408)730-8800
477 N. MATHILDA AVE.
SUNNYVALE, CA. 94086
ATTN: ZyP DESIGN SUPPORT

Rev. A 8/83

ZyPNET PRIME USER'S GUIDE

Doc: 20-010-003 Rev: E	Issued: January 1984	ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA. P.O. BOX 62379, SUNNYVALE, CA 94088 TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

PRIMOS is a trademark of Prime Computer, Inc.

TYMNET is a trademark of Tymnet Inc.

ZyP is a trademark of ZyMOS Corporation.

Table of Contents

2

1 INTRODUCTION

The ZyPNET Prime User's Guide introduces the Prime version of ZyPNET. The term ZyPNET refers to the data network and computer resource used to access the ZyP system and execute ZyP designs. ZyPNET references the network developed by ZyMOS and it can also reference a network established for a specific company. The term ZyPNET-Prime refers specifically to the Prime version of ZyPNET (ZyPNET-VAX is an alternate version) and is available at ZyMOS.

The ZyP system can be accessed over voice-grade phone lines using a standard terminal. A typical configuration consists of a CRT terminal, modem, and printer. A suggested terminal is the Televideo 950 with printer port.

1.1 NOTATION USED IN THIS MANUAL

Upper Case: The text must be typed exactly as specified with only the underlined portion required. For example, LOGOUT may be typed as either LO or LOGOUT.

Lower Case: The word in lower case is to be specified by the user. It may refer to a filename or any of several optional parameters. For example, to invoke the editor, the command would be:

ED fname

If the file was A.NET, the command would be typed as:

ED A.NET

1.2 UNIVERSAL EDITING CHARACTERS

The Prime will recognize two keys as special editing keys at all times. They are used to correct spelling errors in the current line. The backspace key will erase the most recent character typed. Multiple backspaces are allowed, until the beginning of the line is reached. The DEL key will delete everything on the current line. Thus,

EX NET(DEL)ED A.NET would be interpreted as: ED A.NET

ZyMOS has changed the standard Prime character and line delete keys. Prime documentation will reference (') as character delete and (?) as line delete.

2 LOGIN AND LOGOUT PROCEDURE

The Prime can be accessed over normal voice-grade phone lines at 300 or 1200 baud. Since the ZyMOS computer is connected to TYMNET, it may be accessed from most major cities, worldwide, with a local phone call. Accessing the Prime over TYMNET is a two step procedure, TYMNET login and ZyMOS Prime login.

TYMNET login:

1. Dial the number provided by ZyMOS. When you hear the high-pitched tone, place the phone in the acoustic coupler, or push the 'data' button on your modem or phone set.

2. Type a carriage return if you are operating at 300 baud, or the character a if you are operating at 1200 baud with a video terminal.

3. TYMNET will ask you to log in. ZyMOS will assign you a TYMNET ID which is used to respond to the TYMNET log in request.

   please log in: ZyMOS assigned TYMNET ID

4. Next, TYMNET will ask for a password. The TYMNET password is also assigned by ZyMOS. Enter the password. It will not echo to your terminal. Type a Carriage Return. When the plus sign appears, type another carriage return. You are now connected to the ZyMOS Prime computer and will receive a message similar to the following:

   PRIMENET xx.x ZSVx

Prime login:

1. Type: LOGIN name

   where name is the Prime user ID assigned to you by ZyMOS.

2. You will then be asked for your Prime password, initially assigned by ZyMOS. Again, it will not echo to your terminal:

   Password? Prime password

3. The computer will respond with:

```
name (xx) logged in Thursday, 12 Jan, 84  20:04:12
Welcome to PRIMOS version xx.x
Last login Wednesday, 11 Jan 84 12:00:45

Welcome to ZyMOS

OK,
```

When you are first assigned a Prime user ID, you will also be assigned a Prime password. It is a good practice to change your password from time to time to prevent unauthorized access to your files. Once you have logged in, you can change your password with the following command:

CPW old password.
(You will be prompted for a new password.)

When you are finished using the system, type:

LOGOUT

2.1 PROBLEMS IN LOGGING IN

There are several error messages or problems you may encounter when logging in. The following are examples of the most common problems.

1. Invalid command
   Login please
   ER!

Response to keyboard activity before login.

2. Invalid user id or password; please try again
   Login please
   ER!

Response to wrong user name or password.

3. llooggiinn

Terminal set to half duplex. Switch to full duplex.

4. No response.

Your terminal is off-line, not connected to your modem, or you are configured for the wrong baud rate.

3 FILE STRUCTURE

Under the Prime Operating System (PRIMOS), related files can be grouped together into directories. A directory may contain not only files, but other directories as well. Thus, you could have a directory for each chip you are working on. Within each chip's directory could be sub-directories containing logic simulation files, documentation, circuit simulation files for special cells, etc. In this manner, files can be organized in an orderly fashion, permitting easy location. New users often ignore this feature and find it necessary to go back and reorganize their directories. The directory structure of PRIMOS is of fundamental importance in understanding file management. Refer to chapters 2 and 4 of The New User's Guide to Editor and Runoff and chapter 2 of the Prime User's Guide for more information.

3.1 LISTING DIRECTORIES

When you have first logged in, you will be 'attached' to your main directory, also called UFD for User File Directory. To list all the files within it, type:

LD

In the example directory on the next page, if you had logged in with user name JOE, and had typed LD, the names CHIP1, CHIP2, and DOC would be listed.

3.2 CREATING NEW DIRECTORIES

To create a sub-directory (also called sub-UFD), type:

CREATE sub-directory-name

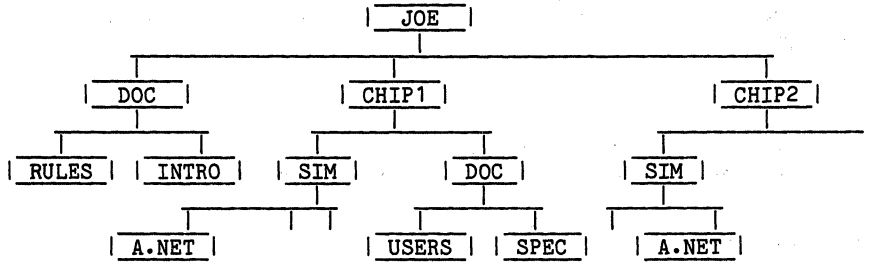
Using the directory on the next page, if you wished to create a directory to hold all files for a third chip, you would type: CR CHIP3. (Note: main directories, UFD's, sub-UFD's, and sub directories will all be referred to as directories. The context of the directory will define its position within the structured hierarchy.)

3.3 ATTACHING TO A DIRECTORY

The easiest way to work with files in a directory is to 'attach' to the directory. This is done by typing:

ATTACH directory-name

Once attached to a directory, you may access the files within it by their simple name. For example, in the directory below, if you had logged in as JOE, and typed: A JOE>DOC, you could access the file JOE>DOC>RULES by its simple name: RULES. If you then wished to access JOE>CHIP1>SIM>A.NET, you would have to type the entire name, or first attach to JOE>CHIP1>SIM, i.e. type: A JOE>CHIP1>SIM



The argument in the above attach command, JOE>CHIP1>SIM, is called a path name or tree name. It starts at the top of the UFD and specifies the path thru the directory to get to the desired sub-UFD. The greater-than symbol (>) is used as the delimiter in path names and is called a path or tree separator.

3.4 DELETING FILES

To delete a file, type:

DELETE fname

e.g. ATTACH JOE>CHIP1>DOC
DELETE SPEC
DELETE JOE>DOC>RULES

3.5 COPYING FILES

To copy a file, type:

COPY fname1 fname2

where fname1 is moved to fname2. For example,

1. To create a copy of a file within the current directory, type:

   A JOE>DOC>CHIP1
   COPY SPEC OLDSPEC

2. To copy a file from one directory to another:

   COPY JOE>CHIP1>DOC>SPEC JOE>CHIP2>DOC>SPEC

3.6 BASIC DIRECTORY AND FILE COMMAND SUMMARY

1. LD - lists all files in a directory and the current directory pathname.

2. CREATE (sub-UFD-name) - creates new sub-UFD.

3. ATTACH (pathname) - attaches to the new directory specified by pathname.

4. DELETE (fname or sub-UFD-name) - deletes file or sub-UFD.

5. COPY (pathname1 pathname2) - copies files or sub-UFD's.

There are additional file commands defined in the Prime User's Guide.

4 EDITOR

The PRIMOS Editor is a line editor and is used to create and edit files containing your logic network, test cases and documentation. It creates a copy of the file you wish to edit, and makes all changes and additions to the copy. It will not update the original file unless specifically told to do so.

This section is intended to present only the basic commands needed to use the EDITOR. For complete information, refer to the New User's Guide to Editor and Runoff.

To edit an already existing file, type: ED fname

To create a new file, Type: ED

4.1 EDIT VS INPUT MODE

Once you have started the editor, there are two modes of operation: EDIT mode and INPUT mode. You can toggle back and forth between these modes by typing a carriage return at the beginning of any line. The editor will respond with 'INPUT' if you are entering INPUT mode or 'EDIT' if you are entering EDIT mode.

4.2 SPECIAL CHARACTERS

INPUT mode is used to enter text. What you type is entered immediately following the current line at which you are positioned. Other than the backspace and DEL key (see introduction), the only other common special character is the back-slash character: \. Whenever it is typed, the current column is set to the next tab stop. The default tab stops are at columns 6, 12, and 30. For example, if you had typed:

```
OUT\GAA1\IN1 IN2 OUT
```

It would be entered into your file as:

```
OUT   GAA1   IN1 IN2 OUT
```

4.3 SUMMARY OF EDIT COMMANDS

Edit mode is used to reposition the current line to a different location in the file, to print several lines, to change a phrase, to delete lines, etc. The most commonly used edit commands are:

C /XXX/YY/	Change the frirst occurence of XXX to YY in the current line.
C .XXX.YY.	Same as above, with a period used as the delimiter to separate the phrases.
C /XXX/YY/G	Change XXX to YY everywhere in the current line.
C /XXX//100	delete the firrst occurence of XXX in the next 100 lines. If a line does not contain an XXX, it will not be changed.
C /X/*/G100	Change all X to * in next 100 lines.
L XYZ	Locate the next line which contains XYZ.
F ABCDE	Find the next line which starts with ABCDE in column 1.
P12	Print the next 12 lines. The last line printed becomes the new current line.
D	Delete the current line.
D12	Delete the current line and the following 11 lines.
T	Go to the top of the file.
B	Go to the bottom of the file.
N	Go to the next line.
N-4	Backup 4 lines.
N10	Advance 10 lines.
UNLOAD fname 10	Copy the current line and the following 9 lines to a file named fname. The previous contents of fname will be lost.
LOAD fname	Insert the contents of file, fname, following the current line. The LOAD and UNLOAD commands are used to move blocks of text.

SAV fname	Save the working copy in file 'fname' without exiting.
FIL fname	File the working copy in file 'fname', and exit the editor.
QUIT	Exit the editor without updating the file being edited. If changes have been made, the editor will ask if you wish to save the changes. Respond with a YES or NO.

4.4 SAMPLE EDIT SESSION

In the following dialog, the user's input is underlined.

First, is the login sequence for a user named DAVID with a password of GOLIATH:

```
OK, LOGIN DAVID
Password? ______

DAVID (user2) logged in Thursday, 12 Jan, 84   07:30:22
Welcome to PRIMOS version 19.2
Last login Wednesday, 11 Jan 84 12:14:39

OK,
```

Next, the old network file is saved in a file called OLD.NET, and a new network file is created.

```
OK, COPY CHIP1.NET OLD.NET
OK, DELETE CHIP1.NET
OK, ED
            (Note: Carriage return typed here)
INPUT
.PADS A:1, B:2, C:2, D:4, OUT:5, VSS:6, VDD:8
GATE1 IAA1 A D
GATE2 GAA1 D B E
GATE3 GAA1 E C OUT
.PRINT A B C / D E / OUT
.MAXTIME=100
.CLK0 A 100 R 0
.CLK0 B 200 R 0
.CLK0 C 400 R 0
.GO
.END
            (Note: Carriage return typed here)
EDIT
FILE CHIP1.NET

OK,
```

The user then notices some changes to make, re-edits the file, and logs out.

```
OK, ED CHIP1.NET
EDIT
N
.PADS A:1, B:2, C:2, D:4, OUT:5, VSS:6, VDD:8
C /C:2/C:3/
.PADS A:1, B:2, C:3, D:4, OUT:5, VSS:6, VDD:8
N5
.MAXTIME=100
C/100/1000/
.MAXTIME=1000
FIL

OK, LOGOUT
DAVID (user2) logged out Thursday 12 Jan, 84  07:35:26.
TIME USED = 00h 05m connect, 00h 04s CPU, 00h 05s I/O.
```

5 PRIMOS COMMANDS

PRIMOS is the Prime multi-user operating system. Major releases are identified with two digits such as 19, minor releases as a decimal such as .2. For example, PRIMOS 19.2 is major release 19, minor release 2. Operating systems are updated once or twice per year and may or may not affect the users operating environment. If a new release has a direct effect on the user, ZyMOS will notify you well in advance of installation of the new release with instructions on how to handle the pending update.

There are many PRIMOS commands, you have already learned the commands for logging in and handling directories. The Prime User's Guide defines all the commands and you should refer to this manual to find out what is available and how the commands are used.

The general format for the PRIMOS command line is:

```
COMMAND [name] [-OPTION argument]...[-OPTION argument]
```

where, name is an optional file or pathname and options are specified with the - delimiter. It is important to learn this format so that you can utilize PRIMOS commands most effectively.

PRIMOS commands can be organized into the following categories:

1. Accessing the system

   LOGIN, LOGOUT, CPW

2. Directory handling

   ATTACH, CREATE, DELETE, CNAME, LD

3. File handling

   SLIST, SPOOL, COPY, CLOSE, CNAME, DELETE

4. Editing

   ED

5. Job processing

   JOB, PHANTOM, COMINPUT, COMOUTPUT

6. Defining your environment

   ABBREV, RDY

7. Setting terminal characteristics

   TERM

8. System status

   STATUS, AVAIL

Refer to the Prime User's Guide for details on these commands as well as many others less frequently encountered in executing a ZyP design.

PRIMOS also has an extensive on-line help utility. The format is

HELP name

where name is the PRIMOS command or topic on which you wish information. If name is omitted, a list of available commands and topics is printed with a descriptive phrase for each.

6 APPLICATIONS PROGRAMS

Applications programs are programs provided by ZyMOS as part of the ZyP system. Typically, they use customer supplied files or source files, and process these files with ZyMOS provided data files or libraries.

ZyMOS has developed ZyP to operate via remote time-share access, on turnkey systems, and with workstations. The needs and resources required or available depend on the particular configuration being used. As a result, not all applications programs run on all configurations. For example, a circuit plot will not be compatible with a remote access alphanumeric terminal. However, all software for successfully completing a ZyP design is available to users, with numerous options depending on configuration.

ZyP applications programs fall into five categories as follows:

1. ZySPICE - circuit simulator.
2. ZyPSIM - logic simulator.
3. ZyTEST - test program generator.
4. ZyPART - artwork generator.
5. Utilities - ZyP specific utilities.

Each application program has a users manual describing its use.

ZyPSIM is the primary user interface to ZyP and its operation is described in detail in document 20-010-109. To invoke ZyPSIM type:

ZYPSIM filename library

where, filename is a user generated file of the form filename.net and library defines the library technology desired by the user. ZyPSIM will simulate the specified file and generate new files with network information and simulation results. All ZyP applications programs conform to this or a similar convention.

Application programs are enhanced and updated several times per year. Releases are defined with the format X.Y.Z, where X is the major release, Y is the minor release, and Z is the revision number. For example, ZyPSIM 5.0.2 is major release 5, minor release 0, revision 2.

ZyMOS less frequently completely revises or significantly upgrades one of the major programs. Such new releases are designated by a numeric descriptor in the name. For example, ZyPSIM II is a major upgrade of ZyPSIM and is upwardly compatible.

Please refer to the ZyP documentation for user and reference data on ZyP applications programs.

7 JOB CONTROL

Creation of the necessary source files and job submission can be achieved by typing the required source data and commands whenever a simulation run is made. However, this can be tedious and inflexible. The Prime system provides several powerful utilities to assist file generation, collectively referred to as Job Control. This section discusses some of the job control techniques available and references ZyPSIM as an example. This section parallels a similar section found in the ZyPSIM User's Guide.

7.1 BUILDING A ZyPSIM .NET FILE

The three components of a .NET file,

1. Network Description
2. Pattern Definition
3. Run Commands

can be individually created and subsequently combined to build a single .NET file required by ZyPSIM. This technique supports construction of option files, for example pattern or run file variations, which can be combined with fixed files, for example a network file, to simulate various operating conditions.

A Prime utility called CONCAT is used to merge files. Refer to the Prime User's Guide for details on the CONCAT command.

As an example of applying CONCAT to ZyPSIM, assume circuit DEMO.NET is to be constructed from three files,

```
DEMO.NETWORK1
DEMO.PATTERN1
DEMO.RUN1
```

The command sequence used to construct the .NET file would be,

```
CONCAT DEMO.NET -COMMAND -OVER -NHE
INS DEMO.NETWORK1
INS DEMO.PATTERN1
INS DEMO.RUN1
QUIT
```

The concatenate program is invoked by the line containing CONCAT. Following the CONCAT keyword is the resulting filename, DEMO.NET. Several options are then specified,

-COMMAND indicates that the command mode will be used with the file data to follow.

-OVER will overwrite (delete) any existing file named DEMO.NET.

-NHE suppresses headers which would otherwise be inserted between the files being combined.

The filenames are inserted using the INS command, one per line. This will allow easy changes when CONCAT is used in a Job Control file. The order of insertion is important and should follow ZyPSIM conventions and requirements. For example, the title should be the first line of the first file concatenated and .END should be the last line in the last file.

The concept of separation and merging of files can be extended to facilitate many processing tasks and takes on added significance when used in conjunction with command files.

7.2 COMMAND FILES

There are two types of command files available on PRIMOS, command input and command output. The Primos User's Guide fully describes these utilities.

7.2.1 COMMAND INPUT

A command input file is a file containing PRIMOS commands, sub program commands, and program data. When the file is executed, the commands are executed as though they were typed at a terminal.

This is a convenient means of creating job control files, where the command sequence is fairly constant with usually only filename changes. The file may be edited for different job submissions or a separate file may be used for each type of job.

The file is created with the desired sequence of commands and execution is then simply a matter of typing,

```
CO filename
```

where filename is the name of the command file. The progress of the execution of commands will be displayed at the terminal.

7.2.2 COMMAND OUTPUT

Command input files can be executed in such a way that the progress of the job is not displayed at the terminal. This occurs when a job is run in batch mode, leaving the terminal free for other work. However, it is desirable to have a record of the progress of a job. The command output file serves this function. Typing,

```
COMO filename
```

will open the named file and direct all terminal and program dialog to it. If the job is interactive, everything that appears at the terminal will also be written to the command output file. If the job is a batch job, all output that would normally appear at the terminal will be written to the command output file.

The command COMO -END will close the file and allow spooling, listing, etc. This command may be given in a command input file for batch jobs or typed at the terminal for interactive jobs.

7.3 JOB CONTROL FILE

The job control file is a command input file which uses a command output file to log job progress. A typical job control file for running DEMO could be constructed as follows:

```
COMO DEMO.RUN_LOG
CONCAT DEMO.NET -COMMAND -OVER -NHE
INS DEMO.NETWORK1
INS DEMO.PATTERN1
INS DEMO.RUN1
QUIT
ZYPSIM DEMO S
DELETE DEMO.NET
SPOOL DEMO.PRINT
COMO -END
CO -END
```

The first line opens the command output file DEMO.RUN_LOG. Next, CONCAT is used to build the .NET file as previously described. When this is completed, the next line invokes ZyPSIM for DEMO using the 5u CMOS library. After the simulation is completed, the DEMO.NET file is deleted to conserve disk space and the simulation results are spooled to the line printer. The command output file is terminated with the COMO -END command and the command input file is terminated by the CO -END command, restoring control to the terminal.

Storing the above file as RUN_DEMO allows the user to execute a DEMO simulation by typing,

```
CO RUN_DEMO
```

A network change could be made as a result of analyzing the simulation results and a new simulation could be executed by typing CO RUN_DEMO again.

Command files can systematize the development process and support orderly verification of networks. The simple example presented here is intended to introduce the concept and illustrate the applications of command files to ZyPSIM. It is important to keep a constant file prefix (in this case DEMO) with applicable suffixes or extensions to allow for easy modification or duplication.

7.4 JOB EXECUTION

Jobs which are defined in command files can be executed in three ways.

1. Interactive - direct submission by typing CO filename. Interactive jobs have the fastest turn-around time.

2. Phantom - submission as a background job whenever a phantom is available. To submit, type PH filename. The terminal is free while the phantom is running.

3. Batch - submission to a batch queue by typing JOB filename. The terminal is free while the batch job is running.

Refer to the Job Control section of the ZySPICE User's Guide for additional information on phantom and batch jobs. Also refer to the various Prime reference manuals for information on job control utilities.

8 SUMMARY

ZyPNET Prime is a powerful computer utility used as a resource for executing ZyP designs under the PRIMOS operating system. The system has many user oriented features which have been introduced in this guide and which are described further in the Prime User's Guide. Beginning users are urged to read the Prime User's Guide to become familiar with the system and its capabilities.

ZyPNET - VAX USER'S GUIDE

Doc: 20-010-018 Rev: B | Issued: September 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: B
Table of Contents	Rev: B
Pages 1-1 through 1-2	Rev: B
Pages 2-1 through 2-2	Rev: B
Pages 3-1 through 3-2	Rev: B
Pages 4-1 through 4-2	Rev: B
Pages 5-1 through 5-2	Rev: B
Pages 6-1 through 6-3	Rev: B
Page 7-1	Rev: B
Page 8-1	Rev: B
Page 9-1	Rev: B
Index	Rev: B

TABLE OF CONTENTS

1 INTRODUCTION

The ZyPNET VAX User's Guide introduces the VAX version of ZyPNET. The term ZyPNET refers to the data network and computer resource used to access the ZyP System and execute ZyP Designs. ZyPNET references the network developed by ZyMOS and it can also reference a network established for a particular company. The term ZyPNET VAX refers specifically to the VAX version of ZyPNET (ZyPNET - Prime is an alternate version) and is available at ZyMOS.

VAX systems tend to be terminal dependent and many users may prefer the DEC VT100, VT125, or equivalent. These terminals have special control keys for using the EDT screen editor. However, ZyPNET VAX is terminal independent for most application programs and can be accessed over voice-grade phone lines. A typical configuration consists of a CRT terminal, modem, and printer. Suggested terminals include the DEC VT100 and Televideo 950. This manual will assume a universal terminal using the line editing commands of EDT.

1.1 NOTATION USED IN THIS MANUAL

1. Underlined - shows user input. For example,

   PRINT CHIP.NET

2. Not Underlined - shows computer response. For example, the response to the above command might be,

   Job 210 entered on queue LPB0:

3. Exclamation mark ! - all entries following an exclamation mark are explanatory notes.

4. Directory name - user defined directory or sub-directory.

1.2 SPECIAL CHARACTERS

Special characters are used to speed communication, improve readability, and to signify an environment or condition.

1.2.1 SYSTEM PROMPT

The VAX system prompt is $. It signifies that you are at the operating system (called VMS) command level and the system is ready for a command.

1.2.2 CHARACTER DELETE

The delete key, marked DEL on most terminals, backspaces over the leftmost character and deletes it.

1.2.3 LINE DELETE

The current line is deleted by typing CTRL/U. To use the control functions, first press and hold the CTRL key, then press a second key, in this case U. Control, CTRL, and ^ will be used synonymously in this document.

1.2.4 INTERRUPT/CONTINUE

To interrupt a command type CTRL/C or CTRL/Y. To resume execution of the interrupted command, type CONTINUE.

1.3 REFERENCE DOCUMENTS

The ZyPNET VAX documentation includes three DEC reference manuals.

1. VAX/VMS Primer - introductory manual to the VAX/VMS environment.
2. EDT Editor Manual - comprehensive user's guide to the DEC VAX EDT editor.
3. VAX/VMS Command Language User's Guide - comprehensive user's guide to the DEC Command Language called DCL.

2 LOGIN AND LOGOUT PROCEDURES

The VAX can be accessed over voice grade phone lines at 300 or 1200 baud. Since the ZyMOS computer is connected to TYMNET, it may be accessed from most major cities, worldwide, with a local phone call. Accessing the VAX over TYMNET is a two step procedure, TYMNET login and ZyMOS VAX login.

2.1 TYMNET LOGIN

a. Dial the number provided by ZyMOS. When you hear the high pitched tone, place the phone in the acoustic coupler or push the 'data' button on your modem or phone set.

b. Type a carriage return if you are operating at 300 baud or the letter "A" if you are operating at 1200 baud with a video terminal.

c. TYMNET will ask you to log in. ZyMOS will assign you a TYMNET ID which is used to respond to the TYMNET log in request.

please log in: ZyMOS assigned TYMNET ID

d. Next, TYMNET will ask for a password. Your TYMNET password is also assigned by ZyMOS. Enter the password. It will not echo to your terminal. Type a carriage return. When the plus sign appears, type another carriage return. You are now connected to the ZyMOS VAX system and will receive a message similar to the following:

ZyMOS call connected

2.2 VAX LOGIN

a. Go through the VAX login sequence as follows:

```
Carriage Return      !Hit RETURN key
User name: user ID   !Enter assigned user ID, RETURN
Password:  password  !Enter password, not echoed, RETURN
```

b. The system will respond with

Welcome to VAX/VMS Version X.X

$ (system prompt)

When you are first assigned a VAX user ID, you will also be assigned a password. It is good practice to change your password from time to time to prevent unauthorized access to your files. Once you have logged in, you can change your password with the following command:

```
SET PASSWORD

old: ________     !Enter old password
new: ________     !Enter new password
verify: ________  !Repeat new password
```

When you are finished using the system, type:

```
LOGOUT
```

3 FILE STRUCTURE

Under the VAX Operating System (VMS), related files can be grouped together into directories. A directory can contain files and other directories, also referred to as sub-directories. You could, for example, have a directory for each circuit you are developing. Within each device directory could be sub-directories containing logic simulation files, documentation, circuit simulation files for special cells, etc. Using directories, files can be organized in an orderly way, permitting easy location. New users often ignore this feature and find it necessary to go back and reorganize their directories. The directory structure of VMS is of fundamental importance to understanding file management. Refer to the VAX/VMS Primer and Command Language User's Guide for more information.

3.1 FILE SPECIFICATION

File specifications provide the system with all the information it needs to uniquely identify a file. File specifications have the following format:

```
node::device:[directory]filename.type;version
```

The punctuation marks and brackets are required to separate the fields. The fields are:

node	Network node name
device	Device name
directory	Directory name or list
filename	File name
type	File type
version	File version number

The maximum size of a file specification (also called filespec), including delimiters, is 128 characters.

3.2 DEFAULTS FOR FILESPECS

The filespec, when defined in its entirety, is long and cumbersome to type every time you want to reference a file. However, it is possible to let the system supply values for certain fields, and these values are called defaults. Normally, you will not need to specify node or device, these are established for you by the system administrator. In addition, directory name can be omitted and will default to your current (default) directory. To display your default device and directory, type:

SHOW DEFAULT

The system will respond something like,

SYS$USER: (name)

where name is your user ID.

4 DIRECTORY COMMANDS

The filespec, as defined earlier, contains node and device names (usually defaulted by the user), a directory name or list, and file information. We will normally only reference directories and files.

A directory name is a 1-9 alphanumeric character string, must be enclosed in square brackets ([]), and has a .DIR extention. Sub-directories can be created in directories to create a directory hierarchy, each level of the hierarchy is separated by a period (.). We will refer to directories and sub-directories synonymously in this document. The directory [JOE], illustrated below, has subdirectories [JOE.DOC], [JOE.CHIP1], and [JOE.CHIP2]. Similarly, sub-directories can have sub-directories, with up to eight levels in any one hierarchy.

The following sections define the most common VMS directory commands. Files are discussed later.

4.1 CREATING NEW DIRECTORIES

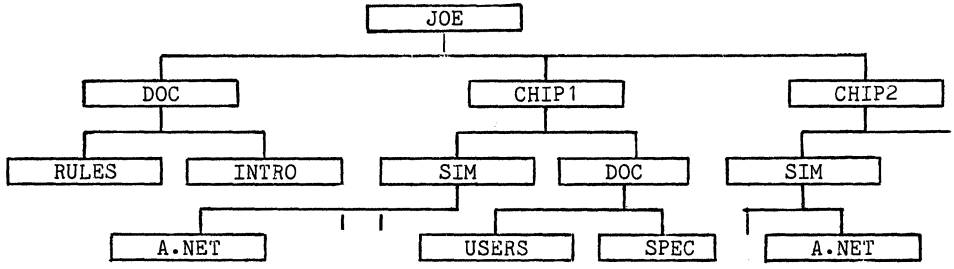
Sub-directories can be created within an existing directory using the CREATE command as follows:

CREATE/DIRECTORY [sub-directory-name]

Where sub-directory-name is a user assigned directory name. For example, in the directory illustrated below, if you want to create a new directory JOE.CHIP3, you would type:

CREATE/DIRECTORY [JOE.CHIP3]

This command creates entry to sub-directory CHIP3.DIR in directory [JOE]. You can now use JOE.CHIP3 in a filespec.



4.2 LISTING DIRECTORIES

To list your (default) directory, type:

DIR

4.3 ATTACHING TO A DIRECTORY

To attach to a directory, type:

SET DEFAULT [directory-name]

For example,

SET DEFAULT [JOE.CHIP1]

Will attach the user to directory JOE.CHIP1.

4.4 DELETING DIRECTORIES

To delete a directory, you must first delete all files in the directory. You must also specify the directory version number when deleting it. The general format is:

DELETE [directory-name;version number]

Where ";" is the version number delimiter. For example, to delete [JOE.CHIP1], version 1, type:

DELETE [CHIP1.DIR;1]

4.5 SUMMARY OF BASIC DIRECTORY COMMANDS

a. SHOW DEFAULT
Shows default device and directory of filespec.

b. SET DEFAULT [directory-name]
Changes or sets default to directory name.

c. CREATE/DIRECTORY [sub-directory-name]
Creates sub-directory.

d. DIR
Lists default directory.

e. DELETE [directory-name]
Delete directory.

5 FILE COMMANDS

File names, file types, and version numbers uniquely identify files within directories. A file name is a 1-9 character string name assigned to a file. A file type is an optional 1-3 character string usually used to identify a files contents and preceded by a (.). Version numbers are decimal numbers from 1 to 32767 that differentiate versions of a file. When you update or modify a file and do not specify a version number for the output file, the system saves the original version for back-up and increments the version number by 1 for the output file. Version numbers are preceded by a semicolon (;).

Files are created both by the editor and ZyP applications programs. For example you can create a ZyPSIM network file using the editor and saving it as chip.NET. Running ZyPSIM on chip.NET creates several new files including chip.OUT and chip.PRT which contain results of the simulation.

The following commands are useful in managing files.

5.1 DELETING FILES

To delete a file, type:

DELETE filename

For example, if you want to delete file CHIP.NET;1, type:

DELETE CHIP.NET;1

You must type the file name exactly as defined and include the version number.

5.2 PURGING FILES

The PURGE command deletes all but the highest version number of a specified file. For example,

PURGE CHIP.NET

would delete all but the most recent version of CHIP.NET.

5.3 COPYING FILES

To copy a file, type:

COPY input-filespec output-filespec

Where input-filespec specifies the name of one or more input files to be copied and output-filespec specifies the output file into which the input files are to be copied.

For example, if you are in directory [JOE.CHIP1.DOC] as previously illustrated and wish to make a copy of SPEC called OLDSPEC, type

COPY SPEC OLDSPEC

The most recent version of SPEC will be copied to OLDSPEC.

To copy SPEC to [CHIP2.DOC]SPEC, type

COPY SPEC [CHIP2.DOC]SPEC

6 EDITOR

The standard VAX/VMS editor is called EDT. EDT is a sophisticated text editor which can function as a line editor or screen editor. Line editing is recommended for low baud rates and is discussed in this section. A comprehensive EDT Editor Manual is provided with the ZyPNET VAX documentation and should be referenced for more information.

6.1 STARTING UP

To start an editing session, type:

EDIT filename

Where filename is of the form "name.type;version". If version is omitted, the most recent version will be used. The system will respond as follows:

a. New file

```
Input file does not exist
[EOB]
*
```

b. Existing file

```
1   (line 1)
*
```

For the case of a new file, the first response indicates that the file specified does not exist (as expected), [EOB] denotes that your file pointer is at the end of the buffer (file), and * is the EDT prompt. If the file exists, it is loaded into a working buffer, the first line is printed, and the * prompt is displayed.

6.2 HELP

EDT has an extensive on line help utility. Typing HELP after the asterisk prompt will display all line editing commands available. Typing HELP command will display information about "command".

6.3 CREATING A FILE

To create a file, start an edit session as defined previously, issue the INSERT command, enter your data, and terminate the insertion mode with CTRL/Z. The DEL and CTRL/U functions are active when using EDT. An example session follows.

```
$EDIT CHIP.NET
Input file does not exist.
[EOB]
*INSERT

   *   NETWORK DESCRIPTION
   *
   *   PADS
   *
   CLOCK1   PAA2  CLK1   N15
   ENABLE1  PDA1  EN1    N9
   .
   .                                  CNTL/Z

[EOB]
*
```

NOTE: The *'s after the INSERT command are entered in the file to specify ZyPSIM comments and are not EDT prompts.

6.4 RANGE SPECIFICATIONS

EDT initially assigns line numbers in increments of one to each line in the buffer (file). When using EDT edit commands you can specify the line or range of lines you want the command to act upon. EDT has the ability to recognize words in combination with line numbers to develop a range specification. The most common are as follows:

1. Period (.) - current line
2. n THRU m - line numbers n thru m.
3. n:m - same as THRU
4. BEGIN - top of buffer (file).
5. END - end of buffer (file).
6. BEFORE - all lines before current line.
7. REST - all lines including and after current line
8. WHOLE - all lines of buffer (file).

6.5 EDITING COMMANDS

The following is a summary of the most common EDT editing commands.

1. DELETE - Deletes range of lines.
2. FIND - Locates specified line.
3. MOVE - Deletes text in one location and inserts it in another.
4. RESEQUENCE - Assigns new line numbers in increments of one.
5. SUBSTITUTE - Replaces a character string with another.
6. TYPE - Displays a specified range of lines.
7. "text" - Locates first occurrence of text.

6.6 ENDING AN EDIT SESSION

An edit session is normally ended in one of two ways.

EXIT - Writes file to start up file name and increments the version number, exits to VMS command level.

QUIT - Abandons file and exits to VMS command level.

6.7 JOURNAL FILE

EDT keeps a journal file of all edits made during an edit session. If a system failure or operator error (such as hitting CTRL/Y) inadvertently ends your edit session, you can recover lost edits using the RECOVER qualifier with EDT. For example, if you were editing CHIP.NET and a system failure occurred, you would type:

```
EDIT/RECOVER CHIP.NET
```

EDT would then reenact the edit session, reading the commands from the journal file and executing them on the screen.

If you invoke EDT without the RECOVER option on an interrupted file, the journal file will be automatically deleted.

When exiting EDT normally, the journal file is automatically deleted.

7 VMS DCL

VMS is the Digital Equipment VAX multi-user operating system. Major releases are identified with a digit such as 3, minor releases as a decimal such as .2. For example, VMS 3.0 is major release 3, minor release 0. Operating systems are updated once or twice per year and may or may not affect the users operating environment. If a new release has a direct effect on the user, ZyMOS will notify you well in advance of installation of the new release with instructions on how to handle the pending update.

DEC calls the command language used to communicate with VAX/VMS Digital Command Language or DCL. There are many DCL commands, you have already learned the commands for logging in and handling directories and files. The VAX/VMS Command Language User's Guide defines all the commands and you should refer to this manual to find out what is available and how the commands are used.

VMS also has an extensive DCL on line help utility. By typing HELP at command level, you can obtain a list of DCL commands. To get help about a particular command, type HELP command.

The general format for the DCL command line is

COMMAND/{qualifier}.../{qualifier} {name}

where COMMAND is a DCL command, qualifier is an optional command qualifier separated by the / delimiter, and name is an optional filespec.

When entering commands which require parameters, you need not enter the entire command at once. The command interpreter will prompt for all required parameters not specified on the command line.

DCL commands are available for accessing the system, directory and file handling, editing, job processing, defining your environment, and system status. Refer to the VAX/VMS Primer and the VAX/VMS Command Language User's Guide for details on using DCL.

8 APPLICATIONS PROGRAMS

Applications programs are programs provided by ZyMOS as part of the ZyP system. Typically, they use customer supplied files or source files, and process these files with ZyMOS provided data files or libraries.

ZyMOS has developed ZyP to operate via remote time-share access, on turnkey systems, and with workstations. The needs and resources required or available depend on the particular configuration being used. As a result, not all applications programs run on all configurations. For example, a circuit plot will not be compatible with a remote access alphanumeric terminal. However, all software for successfully completing a ZyP design is available to users, with numerous options depending on configuration.

ZyP applications programs fall into five categories as follows:

1. ZySPICE - circuit simulator.
2. ZyPSIM - logic simulator.
3. ZyTEST - test program generator.
4. ZyPART - artwork generator.
5. Utilities - ZyP specific utilities.

Each application program has a users manual describing its use.

ZyPSIM is the primary user interface to ZyP and its operation is described in detail in document 20-010-109. To invoke ZyPSIM type:

ZYPSIM filename library

where filename is a user generated file of the form filename.net and library defines the library technology desired by the user. ZyPSIM will simulate the specified file and generate new files with network information and simulation results. All ZyP applications programs conform to this or a similar convention.

Application programs are enhanced and updated several times per year. Releases are defined with the format X.Y.Z, where X is the major release, Y is the minor release, and Z is the revision number. For example, ZyPSIM 5.0.2 is major release 5, minor release 0, revision 2.

ZyMOS less frequently completely revises or significantly upgrades one of the major programs. Such new releases are designated by a numeric descriptor in the name. For example, ZyPSIM II is a major upgrade of ZyPSIM and is upwardly compatible.

Please refer to the ZyP documentation for user and reference data on ZyP applications programs.

9 SUMMARY

ZyPNET VAX is a powerful computer utility used as a resource for executing ZyP designs under the VMS operating system. The system has many user oriented features which have been introduced in this guide and which are described further in the DEC support documentation. Beginning users are urged to read the VAX/VMS Primer to become familiar with the system and its capabilities.

V

Z

4

ZyPSIM USER'S GUIDE

Doc: 20-010-009 Rev: E	Issued: July 1984	ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA. P.O. BOX 62379, SUNNYVALE, CA 94088 TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

4

Table of Contents

List of Illustrations

1 INTRODUCTION

1.1 OVERVIEW

ZyPSIM is a ZyMOS proprietary general purpose MOS logic simulator specifically developed for the ZyP System. It is supported with standard cell model libraries enabling networks of standard cells to be simulated based on actual cell performance.

The logic simulator is the primary interface between the user and the ZyP system. Simulation of a network is the main design verification tool for ZyP based designs. Accurate simulation of a network of standard cells virtually guarantees IC functionality and establishes IC dynamic performance capability. In addition to logic simulation, the ZyPSIM network file is used as the source data file for artwork generation (ZyPART) and the simulation results are used for test program development (ZyTEST). The ZyP Error Eliminator (ZyPEE) program is also available to assist in identification and correction of network errors.

The ZyPSIM User's Guide is intended as an overview for using ZyPSIM and provides a methodology for its application with examples.

The ZyPSIM Reference Manual should be reviewed in detail to become familiar with ZyPSIM syntax, commands, and techniques.

The ZyPSIM User's Guide divides the general application of ZyPSIM into the following steps:

a. Basics - description of some basic logic simulation concepts.

b. Network Description - how to describe a ZyP network to ZyPSIM.

c. Pattern Definition - how to define patterns for exercising the ZyP network.

d. Running ZyPSIM - how to run an actual simulation.

Each of these four steps will be described as it applies to a ZyP circuit design.

1.2 DOCUMENT CONVENTIONS

In the examples illustrating ZyPSIM usage, upper case text indicates commands, lower case text indicates parameters (e.g., ZyPSIM fname library).

1.3 RELATED DOCUMENTS

Supplemental information regarding ZyPSIM operation can be obtained from the following documents:

Document	Document Title
20-010-209	ZyPSIM Reference Manual
20-010-309	ZyPSIM Test Language
20-010-016	ZyPEE User's Guide
20-010-010	ZyP Specification and Test User's Guide
20-010-210	ZyTEST User's Guide
20-010-310	ZySPEC User's Guide
20-010-107	Zy40000 Series CMOS Cell Library
20-010-117	Zy50000 Series CHMOS Cell Library
20-010-111	Zy40000 Series Analog Cell Library

2 SOME BASICS

Before studying an actual simulation, some basic concepts of logic simulation in general and ZyPSIM in particular will be reviewed.

2.1 FILES

ZyPSIM operates with two classes of files, primary and secondary. Files are either user generated or ZyPSIM generated and are described below.

2.1.1 PRIMARY FILES

Primary files are always required or generated when running a simulation. There are six primary files as follows:

a. filename.NET

A user generated file with name "filename" and extension .NET. It contains all user input including network description, input patterns, and commands.

b. filename.OUT

A ZyPSIM generated file with name "filename" and extension .OUT. It contains compilation results including error messages.

c. filename.PRT

A ZyPSIM generated file with name "filename" and extension .PRT. It contains simulation results in the form of output pattern listings.

d. filename.TPF

A ZyPSIM generated file with name "filename" and extension .TPF. It contains simulation results for the fast operating condition.

e. filename.TPS

A ZyPSIM generated file with name "filename" and extension .TPF. It contains simulation results for the slow operating condition.

f. filename.TPN

A ZyPSIM generated file with name "filename" and extension .TPN. It contains simulation results for the nominal operating condition.

2.1.2 SECONDARY FILES

Secondary files are files generated to aid network analysis. Examples are as follows:

a. filename.SPI

A ZyPSIM generated file which lists spike events by time and node name. If no spikes are generated, this file is null.

b. filename.XRE

A ZyPSIM generated cross reference file resulting from the .XREF command. This file lists fan-in, fan-out, capacitance, absolute (calculated) delay, and node sensitivity to loading for every node. The .XRE file is useful in debugging and analyzing networks.

c. xxxx.PRT

A ZyPSIM generated print file resulting from a multiple print command of the format .PRINT, xxxx nodelist. The nodelist output for the simulation will be written to file xxxx.PRT and use of multiple print files allows the user to trace multiple events with manageable listings.

d. filename.TOG

A ZyPSIM generated file which lists all the circuit nodes and indicates those which have been toggled.

2.2 PRIMITIVES

Primitives are the fundamental components manipulated by ZyPSIM. They include gates, flip flops, capacitors, etc. ZyMOS has utilized these primitives to construct performance accurate models of standard cells in the form of macros.

The user will never use primitives to describe a ZyP network. However, knowledge of what primitives exist will be useful in understanding how models are constructed. They can also be used to create boolean expressions for print conditions. An example of the latter is 'print on A or B'. Applying signals A and B to an OR gate and printing on the gate output will accomplish the desired print function. This concept can be applied to print file generation, network analysis, and test.

2.3 CELL LIBRARIES

ZyPSIM is supported by a series of cell model libraries. The desired library is specified when entering the ZyPSIM environment as described in Section 5. Each of the ZyP technologies has cell library models using worst case process parameters for delay and loading attributes to simulate silicon level performance.

Libraries are specified as follows:

Zy4 - Zy40000 Series CMOS Cells.
Zy5 - Zy50000 Series CHMOS Cells.

In addition to the standard ZyMOS libraries, the user can create his own libraries by using the .LIBRARY command. This is a convenient way of making complex elements available to multiple projects or to map the standard cell library to a user specific index.

Cell models are a key feature of ZyPSIM. The cell library enables ZyPSIM to be a high accuracy silicon level circuit analysis tool. Cell models represent:

- functionality
- both intrinsic (no load) and load dependent delays
- input capacitance loading.

For certain cells additional parameters define

- set-up time
- hold time
- minimum pulse width
- maximum slope time.

Models are created as macros. Macros are defined prior to the network file and specified as calls (or uses) in the network file.

The model for a simple inverter may be defined as follows:

```
.MACRO IAA1 IN OUT
IAA1 INV 3.2,1.2 IN=IN OUT=OUT
.EOM
```

MACRO is the keyword for defining a macro, IAA1 is the macro name, and IN OUT is the I/O pin list. For the inverter, there is one input IN and one output OUT. IAA1 is also a standard cell name and is described in the appropriate cell data sheet.

Following the .MACRO keyword is the actual logic/performance definition of the device. IAA1 on the second line is the gate name of the inverter, INV invokes the ZyPSIM inverter model, the 3.2,1.2 field defines zero to one and one to zero propagation delays, IN=IN defines the input, and OUT=OUT defines the output. The IN=IN field can include input loading data. The actual cell IAA1 has intrinsic delay, load dependent propagation and rise/fall time delays, and input loading data included in the model derived from ZySPICE simulation.

The macro is concluded with the .EOM statement.

The input node name IN and output node name OUT which define the I/O pin list in the first line of the macro are assigned for convenience but do not have to be used when applying the model.

The general format for calling a specific cell macro is macro name input list output list element name

For example, the macro for cell IAA1 can be called in a network file as follows,

```
N2  IAA1  IN=N1  OUT=N2     (keyword method)
or
N2  IAA1     N1      N2     (fixed format method)
```

For the case of the inverter, N2 is the inverter name, IAA1 is the macro name, N1 is the input, and N2 is the output. Note that the gate name and the output name are the same. This is only for convenience. N1 and N2 are an **ordered** list of nodenames assigned to the input(s) and output(s). For IAA1, N1 is assigned to model input IN and N2 is assigned to model output OUT. If the keyword method is used, the signals may be listed in any order; if the fixed format method is used, the signals **must** be listed sequentially as specified by the .MACRO definition. The fixed format method minimizes typing but requires prior knowledge of the signal order.

ZyPSIM cell models and the required INPUT OUTPUT format are defined in the appropriate cell data sheet.

When the network which contains the IAA1 macro is simulated, actual delays and loads will be used in the simulation to accurately predict silicon level performance. These model parameters correspond to the data sheet numbers documented in the ZyPLIB manuals.

Defining (.MACRO) cell models has been done by ZyMOS for the ZyP user's convenience. Applying the models with macro calls is required in the user's network.

It is extremely important to note that every element in a network **must** be defined as a standard cell macro. The macro call is the user link to standard cell, both logic/performance simulation and artwork generation. Only standard cell macro calls will appear in silicon.

2.4 TIMING

Logic simulation produces a state versus time output listing for a network. The basic unit of time is nano-second (i.e., 1E-9 second).

All cell models have their timing parameters derived from detailed ZySPICE simulations of the actual logic/circuit design which has been committed to artwork (layout) and, therefore, will appear in silicon. The resulting parameters are based upon worst case process (i.e., slow) and nominal environmental operating conditions of:

TEMP = +27 deg C
VSUPPLY = +5.0 Volts

Guardbanded simulations (e.g., "worst case") of one's network is required by ZyMOS and easily accomodated by using the .GUARDBAND command, which applies a ZyPSIM generated multiplication constant to all timing parameters in those cell models appearing in the network description after the .GUARDBAND command. The user specifies the operating conditions which adjust the network timing for voltage, temperature, and processing effects.

2.5 DEMO CIRCUIT

Throughout the subsequent discussion, the circuit shown in Figure 2-1 will be used as an example. Section 8 contains the ZyPSIM example files. The filename for this circuit is DEMO and is run using the Zy50000 library. DEMO is also used as an example circuit in most of the ZyP Documentation.

DEMO is identical to the standard CMOS 4520. The 4520 is a dual, four bit, synchronous, binary, up counter. Figure 2-1 shows one of the dual counters and, as indicated in the note, adding 17 to the node numbers defines the second counter. The two counters are completely independent. Both will be described in the network file but only one will be monitored during simulation.

2.6 COMPUTER RESOURCE

ZyPSIM has been designed to conserve computer resource usage while providing high accuracy. Nevertheless, ZyPSIM may require many CPU seconds to simulate large networks. Use batch processing where possible and conserve nanoseconds. Verify both logic and timing of subnetworks first and combine into larger networks after subnetwork debug is complete. Most importantly, plan your simulation session.

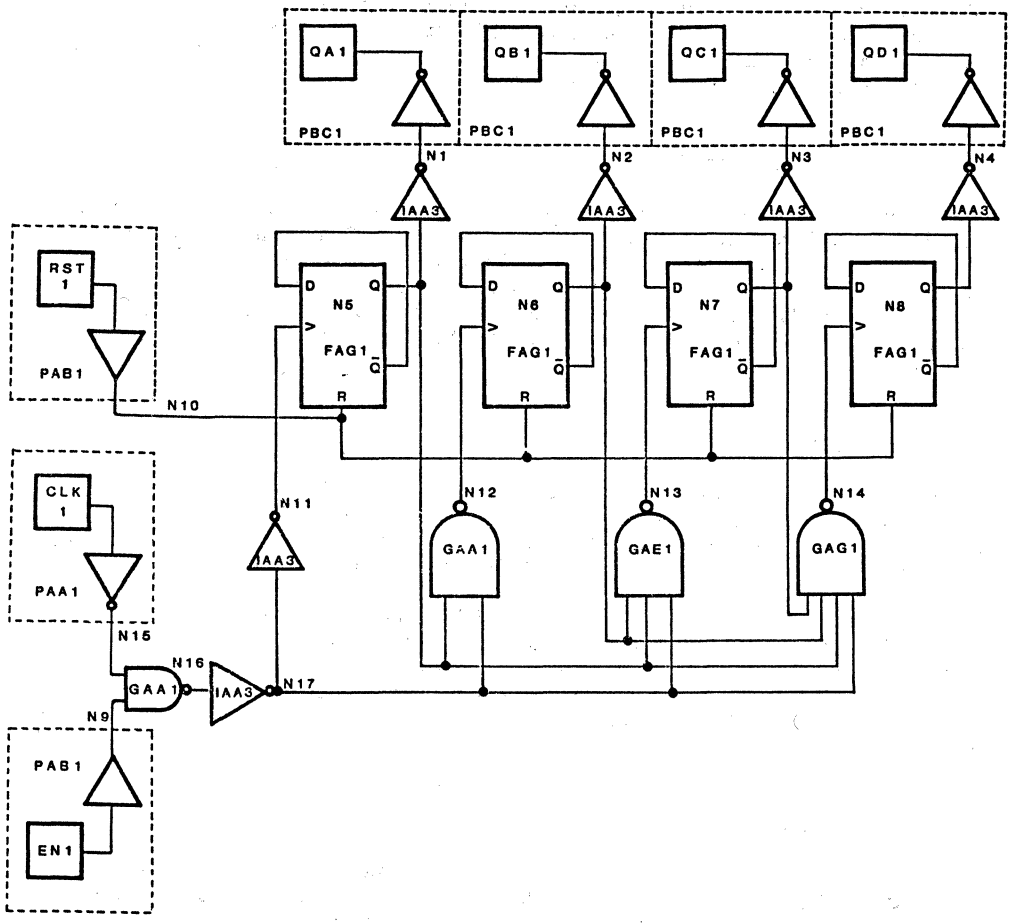


FIGURE 2-1. DEMO CIRCUIT DIAGRAM

3 NETWORK DESCRIPTION

The network description is the initial part of the user data file

filename.NET

Via the editor, the user creates a network description and files it as filename.NET. For DEMO, the network description is stored in file

DEMO.NET

3.1 NODES AND CELLS

Before creating a network file, all network nodes must be uniquely defined and all components assigned to standard cells. Figure 2-1 is annotated with node names N1, N2, N3, etc. and all components appear as standard cells. For nodes, any convenient name can be used; the first character must be a letter and the remaining characters can be numbers, letters, *, _, or period. It is suggested that node names start with the letter N followed by sequential numbers as in figure 2-1. Inverted (negative) nodes can be prefixed with a second N as in NN1, NN2, etc. ZyP documentation generally follows this convention.

3.2 NETWORK FILE STRUCTURE

As mentioned previously, the network description is the initial part of the user data file filename.NET. The network portion of this file consists of two required sections:

1. "title" must be the first line in the file.
2. The network description using standard cell macro calls.

The DEMO network file listing begins as follows:

```
ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)
*
*                    NETWORK DESCRIPTION
*
*          PADS
*
CLOCK1       PAA1 CLK1 N15
```

The title is ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION) as defined in the first line of the .NET file.

Following the title in the DEMO network listing is the actual network description. The first cell in the network listing is a bond pad cell PAA1. Bond pad cells contain protection networks, input and/or output buffers, and bonding pads for interfacing the IC to external pins.

After the pad cell section of DEMO.NET is the section labeled "INVERTERS". The first inverter cell is called with the statement

```
N1   IAA3  N5   N1
```

Refer to the IAA3 data sheet for technical details on IAA3. Ten incidences of use of IAA3 are defined according to the format defined in paragraph 2.3. The network is built up with macro cell name calls until it has been completely defined. The user should study Section 8.1 with figure 2-1 to understand the details of defining a network. The model formats are defined on the cell data sheets.

The network file can also contain user defined macros of ZyP cells. These definitions, using the .MACRO keyword, precede their use and are convenient for defining circuit subnetworks. See section 6 for a further discussion of macros.

It is important to reiterate that all network elements must be specified with standard cell macro calls. Cells not specified as standard cell macro calls will not appear in the silicon circuit.

4 PATTERN DEFINITION

Pattern definition refers to creation of an input pattern used to exercise a network and verify operation. The pattern definition is generally the second part of the user file

filename.NET

4.1 PATTERN FILE STRUCTURE

The pattern section of the .NET file is constructed using the ZyPSIM commands:

a. .CLK - used to generate time variant input signals.

b. .TABLE - used to define inputs in tabular form.

c. .TTABLE and .TGEN - used to generate test system compatible patterns. They support powerful subroutine capabilities.

These commands are defined in detail in the ZyPSIM Reference Manual (20-010-209) and ZyPSIM Test Language manual (20-010-309).

The pattern definition for DEMO consists of the following statements:

```
*
*                    PATTERN DEFINITION
*
.CLK1 CLK1 100 200 R 100
.CLK1 CLK2
.CLK1 RST1 50 3450
.CLK1 RST2 50 3450
.CLK1 EN1 450 650
.CLK0 EN2
```

The first command applies to input signal CLK1. It starts at 1 and remains there until time 100. At time 100 it goes to 0, returns to 1 at interval 200, and toggles every 100 nanoseconds thereafter. This corresponds to a frequency of 5 MHz.

The second command applies to the clock input CLK2. It remains 1 for the entire simulation.

The remaining four commands apply to the remaining four inputs of DEMO and are similar to the CLK1 and CLK2 inputs.

As can be seen, defining input patterns for ZyPSIM is flexible and straight forward. The ZyP user should also study application of the .TABLE and .TTABLE commands. Powerful subroutine capabilities are available with the .TTABLE command.

4.2 GENERAL PATTERN COMMENTS

The worst case network simulation responds as an actual MOS circuit. Node timing is a function of driving device impedance and load. Every node and intermediate state can be monitored and occasionally the data can be overwhelming. It is recommended that a circuit first be simulated while monitoring a minimum number of signals. If the response is not as expected, additional nodes can be monitored to identify problem areas.

At time zero, the input signals are allowed to propagate through the entire network to initialize internal nodes. As a result, all nodes start in a stable state : 1,0,X,Z,etc.

No internal node initialization is directly accomodated with ZyPSIM because IC testing requires initialization from the device's external pins. Plan for initialization and testing of the final circuit at the beginning of the design. The simplest approach to network initialization is to apply SET/RESET/PRESET signals to all network memory elements at time zero.

5 RUNNING A SIMULATION

Having created the network description and pattern definition sections of the .NET file, the next step is to run a simulation.

5.1 RUN COMMANDS

Run commands are the third and final part of the .NET file.

5.1.1 PRINT

This command specifies which nodes to print during a simulation. The format is:

1. .PRINT nodelist

   The simulation output is directed to both the terminal monitor (CRT) and the file filename.PRT

2. .PRINT,filename nodelist

   The simulation output is directed to the file filename.PRT

3. .PRINT,filename.ext nodelist

   The simulation output is directed to the file filename.ext

ZyPSIM supports multiple .PRINT commands; therefore, a single simulation may contain up to seven (7) .PRINT commands.

For DEMO.NET, the print command is:

```
.PRINT RST1/N10/CLK1/N15/N16/ etc.
```

where .PRINT is the command keyword, RST1, N10, CLK1, N15, N16, etc., are node names to be printed, and / signifies a blank column.

5.1.2 TIMING

ZyPSIM simulates in nanoseconds. An optional .RESOLUTION command will increase internal simulation resolution to one tenth of a nanosecond. There are three run commands which define print times and the maximum number of nanoseconds to be simulated. These times should be specified in integral nanoseconds.

a. .POC nodelist - defines print as those times when nodelist changes.

b. .DELTAT=x - specifies time increment in nanoseconds (x) between print times.

c. .MAXTIME=y - specifies maximum number of nanoseconds (y) for the simulation.

The DEMO network file specifies .POC with no nodelist. This is a shorthand way of specifying print when any node in the print list changes.

5.1.3 INITIATING A RUN

Following definition of print nodes and simulation timing, the simulation is initiated with the command

```
.GO
```

The .GO command executes immediately; it should be the next to last command in the .NET file.

When this command is executed, ZyPSIM will initiate a run for the number of nanoseconds specified in the .MAXTIME= command. For DEMO, the maxtime is 3600 nanoseconds.

ZyPSIM enters a simulation run in an uninitialized state and executes the number of nanoseconds specified.

5.1.4 TERMINATION

A simulation run is terminated at the end of maxtime.

The .NET file is terminated with the command

```
.END
```

This must be the last line of the .NET file.

5.2 RUN EXECUTION

Once the three sections of the .NET file are complete, the file is ready to be executed by typing

```
ZYPSIM filename {Zy4, Zy5}
```

where Zy4 or Zy5 selects the desired library.

5.3 OUTPUT

Two primary files store ZyPSIM output:

filename.OUT - compilation results.

filename.PRT - simulation results.

For DEMO the output files are:

DEMO.OUT

DEMO.PRT

The DEMO.PRT file created by running DEMO.NET is listed in Section 8.2. An example .OUT file is listed in Section 8.4.

If multiple runs are made using the same .NET file but with modified run commands, and a prior output is to be saved, it is necessary to rename either the filename.PRT file or the filename.NET file. Alternatively, the user can direct the output to a user defined filename. See the ZyPSIM Reference Manual for definition of multiple .PRINT commands.

6 ADVANCED FEATURES

The DEMO circuit demonstrates the basic requirements and operation of ZyPSIM. However, there are a number of advanced ZyPSIM features which specifically address standard cell IC design with the ZyP system.

6.1 COMPILATION RESULTS

After ZyPSIM has compiled a network, it summarizes the results with the following data:

```
CIRCUIT HAS BEEN COMPILED

  NUMBER OF SIGNALS           = w
  NUMBER OF MACRO DEFINITIONS = x
  NUMBER OF GRIDS             = y
  NUMBER OF TRANSISTORS       = z
```

This data is preserved in the .OUT file.

Number of signals, macros, and transistors are self explanatory but number of grids is a unique feature of ZyPSIM. Based on the network, cell list, and number of cells, ZyPSIM is able to predict final die size. A grid is a unit of length and varies as a function of technology. By knowing the number of grids, annual volume, and package, ZyMOS is able to quote production prices.

Section 8 shows the summary output stored in the .OUT file for DEMO_TEST. Note that it also includes a detailed summary of cell occurrences by name and grid count.

The syntax of the macro statement is COMMAND (see below) followed by the name of the macro followed by an ordered list of signal names. The following table is presented in order to more adequately define the matrix of user defined macros.

6.2 USER MACROS

Macros combine elements or cells into groups which can be manipulated as single components. Standard cells are macros of ZyPSIM elements, developed by ZyMOS, to model cell functionality and performance. User macros can group standard cells into complex components using one of four methods described in table 6-1. The syntax of the macro statement is COMMAND (see below), followed by the name of the macro, followed by an ordered list of signal names. User defined macro names must not begin with the letter A, P, or Z.

TABLE 6-1. MATRIX OF USER DEFINED MACROS

COMMAND	CATEGORY	PURPOSE/DESCRIPTION
.MACRO	REGULAR	Hierarchical design, typing convenience. No artwork positions are preserved.
.MACRO	TELESCOPING	Preserves electrical characteristics and saves space. Contains only telescoping digital cells.
.MACRO	ANALOG	Required for analog functions. Contains only analog cells.
.CMACRO	CLUSTER	Preserves electrical characteristics for critical portions of a circuit.

6.2.1 REGULAR (.MACRO)

A regular macro is a grouping of standard cells into a desired function. Refer to the ZyPSIM Reference Manual for detailed rules on macro construction. The general procedure is to define the macro ID, define inputs and outputs, and describe the internal configuration of cells and their interconnection.

Consider, for example, a one of four decoder. The following macro could be constructed to provide this function.

```
*                STANDARD USER MACRO
*                  1 OF 4 DECODER
*
*         OUT1=00  OUT2=10  OUT3=01  OUT4=11
*
.MACRO DECODER IN1 IN2 OUT1 OUT2 OUT3 OUT4
NIN1  IAA1  IN1   NIN1
NIN2  IAA1  IN2   NIN2
OUT1  GAB1  IN1   IN2  OUT1
OUT2  GAB1  NIN1  IN2  OUT2
OUT3  GAB1  IN1   NIN2 OUT3
OUT4  GAB1  NIN1  NIN2 OUT4
.EOM
*
```

The decoder macro can be called in a .NET file as follows:

```
*               CALL DECODER MACRO
*
DEC1  DECODER  INA INB OUTA OUTB OUTC OUTD
*
```

DEC1 is the macro ID for this specific usage of the macro, DECODER is the macro name type, and INA ... OUTD is an ordered pin list of user assigned names.

Standard macros are subject to certain rules:

1. All components in a macro must be standard non-telescoping ZyP cells.

2. The macro must be defined prior to its use.

3. Nesting is limited by the "expanded" element ID.

The nesting/naming capability requires further explanation. As described in the ZyPSIM Reference Manual, each time a macro is nested, the macro ID is prefixed to all element ID's in the called macro. The "extended" element ID cannot exceed 16 characters. For example, assume the decoder macro is called by a second macro as follows:

```
*         NESTED MACRO NODE NAME EXAMPLE
*
.MACRO EX N1 N2 N3
   .
   .
   .
DEC3  DECODER N1 N2 OUT5 OUT6 OUT7 OUT8
   .
   .
   .
.EOM
*
```

The element ID of the first inverter in macro DECODER is expanded as the original element ID (NIN1), plus macro ID (DEC3), separated by periods. For this example, the name becomes:

```
DEC3.NIN1
```

The ZyPSIM naming convention permits unique identification of all elements in a network, including those in macros. As can be seen, the element ID has grown from 4 characters to 9 characters due to the nesting. Element ID's expanded by nesting cannot exceed 16 characters. If single character names are used, this permits nesting to 8 levels. In practice, 4-6 levels are typical. Hierarchical design to four levels is a powerful tool when utilizing macros to their fullest advantage.

6.2.2 TELESCOPING

Telescoping macros are a unique feature of the ZyP system and deserve special attention. These macros support the construction of variable length registers, counters, memories, and certain analog functions. However, unlike standard macros, a telescoping macro physically places cells adjacent to each other to achieve a variable size function. As a result, telescoping macros must conform to certain rules.

Consider, for example, a seven stage shift register. This register could be defined as a standard user macro using seven FAC1 flip flops, or as a telescoping macro using telescoping cells. Telescoping cells consist of control and body cells which, when laid side by side, pass control and data lines from stage to stage without using the general routing channels. This results in enhanced performance, flexibility, and optimum utilization of silicon. Telescoping macros should be used whenever possible to minimize area and maximize performance.

As previously mentioned, telescoping macros consist of two generic standard cell types, control and body. A control cell provides the control interface to the rest of the external network and, in some instances, may include a body cell and output ports. For example, the control cell for a telescoping shift register, SAQ1, includes clock and reset inputs and outputs which only go to body cells. A body cell is the repeating function and is designed to be located physically and logically next to a control cell or compatible body cell. Cell SAE1 is an example of a body cell which is designed to be used with control cell SAQ1. Refer to the SAQ1 and SAE1 data sheets for additional cell and application information. By providing the control/body cell capability with telescoping macros, the user is able to 'build' variable length functions.

The seven stage shift register telescoping macro could be constructed as follows:

```
*           TELESCOPING USER MACRO
*              7 STAGE SHIFT REG
*
*          INPUTS ARE DATA, CLOCK, RESET
*          OUTPUTS ARE Q4 AND Q7
*
.MACRO  SR7 DATA CLOCK RESET Q4 Q7
CST1   SAQ1 DATA CLOCK RESET QS1 CLO NCLO NR Q1 NQ1
ST2    SAE1 QS1  CLO   NCLO  NR  QS2 Q2   NQ2
ST3    SAE1 QS2  CLO   NCLO  NR  QS3 Q3   NQ3
ST4    SAE1 QS3  CLO   NCLO  NR  QS4 Q4   NQ4
ST5    SAE1 QS4  CLO   NCLO  NR  QS5 Q5   NQ5
ST6    SAE1 QS5  CLO   NCLO  NR  QS6 Q6   NQ6
ST7    SAE1 QS6  CLO   NCLO  NR  QS7 Q7   NQ7
*
.EOM
*
```

The shift register macro can be called in a .NET file as follows:

```
*           CALL SHIFT REGISTER MACRO
*
SREG1  SR7 D CL R T1 T2
*
```

SREG1 is the name of this occurrence of SR7, SR7 is the telescoping macro name, and D ... T2 is an ordered pin list of user assigned names.

Telescoping macros are subject to the following rules:

1. All components in the macro must be compatible standard telescoping ZyP cells.
2. The macro must be defined prior to its use.
3. Nesting in the telescoping macro is not allowed, i.e., telescoping macros cannot call other macros.
4. Cells **must** be listed in order of signal flow.
5. There is a physical size constraint as defined by the specific cell type used.

6.2.3 ANALOG (.MACRO)

Analog macros are required in order to use analog cells. This allows (forces) the analog cells to be treated separately from the digital logic during layout and is necessary so that the special constraints of mixed analog and digital circuits (separation of wells, provision for separate analog power and ground lines, etc.) can be handled. For a further discussion of analog cells see the ZyP Analog Design Guide in 20-101-111 Vol. 1.

Analog cells, and only analog cells, must be used within an analog macro. An analog macro may not contain digital cells, digital telescoping cells, or a user defined macro call of any kind.

6.2.4 CLUSTER (.CMACRO)

The CMACRO or cluster MACRO feature of the ZyP Design System allows the system designer a means of affecting the physical layout of the IC design while interacting at the logical simulation and verification level, and as such is the means of indicating critical clock skew areas or timing paths which should be preserved during the artwork phase. It is used to group non-telescoping cells which are related only by logic functionality. The cells are placed in a row in the sequence in which they were defined in the CMACRO definition.

For the purpose of illustration, assume that a serial path exists in a particular design, and that this serial path is a classical example of register clock skew, (i.e., that the interstage clock and data path could be subject to an unequal delay such that the preceding (N-1)0 state data change precedes the synchronous clock to the Nth register stage). This condition is not always detectable prior to the physical layout of the circuit, however ZyMOS utility ZyPEE will detect this arrangement logically and alert the system designer that a potential problem exists. At this time the system designer can define the path in question as a CMACRO and group the stages adjacent to each other thereby eliminating any possible skewing problems.

One important rule for building CMACRO's is that all internal connections of the Macro must also appear as external connections of the CMACRO in order to be correctly connected. This means that if an internal node connects to internal I/O ports, then both of these ports must appear as separately named signals in the CMACRO I/O list. To see how this rule changes the syntax first consider the circuit in figure 6-1.

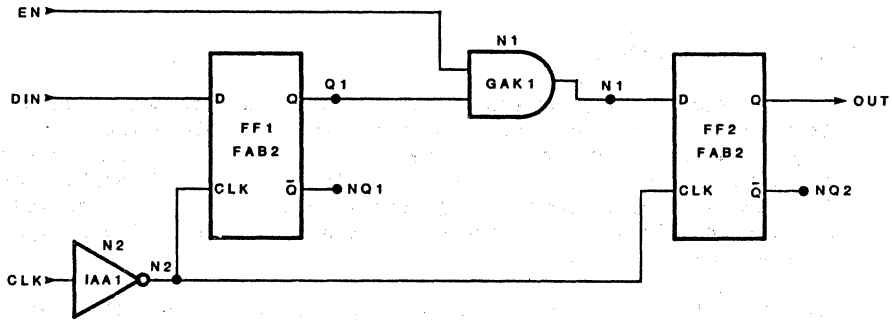


FIGURE 6-1. EXAMPLE CIRCUIT

A regular Macro named SHIFT could be specified for this circuit portion as:

```
.MACRO SHIFT  DIN  CLK   EN  OUT
FF1     FAB2  DIN   N2   Q1  NQ1
FF2     FAB2   N1   N2  OUT NQ2
N1      GAK1   EN   Q1   N1
N2      IAA1  CLK   N2
.EOM
```

And it could be called as:

```
M7     SHIFT  INP   CL  EN1  OUT
```

To make a CMACRO of this circuit requires additional nodes to be named and brought out to the macro I/O list. Figure 6-2 illustrates the network changes needed.

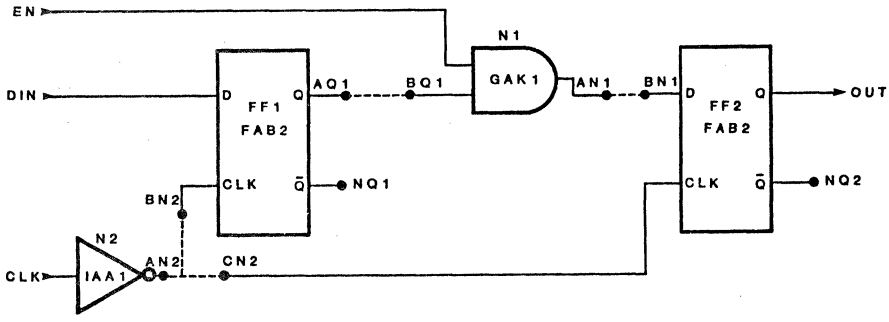


FIGURE 6-2. NETWORK CHANGES FOR CMACRO

A CMACRO named SHIFT could be specified for this circuit as:

```
.CMACRO SHIFT  DIN  CLK   EN  OUT
+ AQ1 BQ1  AN1 BN1  AN2 BN2 CN2
N2      IAA1  CLK  AN2
FF1     FAB2  DIN  BN2  AQ1   NQ1
N1      GAK1   EN  BQ1  AN1
FF2     FAB2  BN1  CN2  OUT   NQ2
.EOM
```

And it could be called as:

```
M7      SHIFT  INP   CL  EN1  OUT
+ Z10 Z10  Z11 Z11  Z12 Z12 Z12
```

The cell order in the CMACRO (N2, FF1, N1, and FF2) is the order from left to right in which the cells will be placed in a row of the actual chip. Notice that the only internal nodes which do not appear on the CMACRO I/O list are NQ1 and NQ2 because they do not connect to any other cells, either within the macro or outside of it. The dummy node names Z10, Z11, and Z12 must be unique within their nesting/naming level since they become signal names just like INP and CL on the level of the M7 macro.

There is just one further rule for the formation of CMACRO's. A CMACRO may not contain/call any other defined macro. A CMACRO may be called by a user defined macro but it may not call one itself. Thus, a CMACRO may not include either analog, telescoping digital, CMACRO digital , or "regular" digital macros. A CMACRO may contain only stand alone digital cells.

The user is cautioned against abusing the CMACRO feature since a system of total critical timing paths may totally inhibit the ability to do a computer routing and thus require a full custom hand layout.

6.2.5 MAXIMUM CONTIGUOUS LENGTH

When a telescoping, analog or cluster macro is processed by the ZyPART software, a composite cell is generated which is a contiguous block containing all the component cells. The "new" cell could be of significant length relative to the optimum length of the row. This makes placement of other cells in the row, or signal routing through the row somewhat difficult. To avoid this situation, it is necessary to restrict the length of a telescoping macro to 60% of row length. This may be calculated as follows:

W = P*SQR(2 * G * H) or P*(NP * 2 * H)/4 (pad limited case).

whichever is greater, where:

W = Maximum width of macro
P = .6 (60% of row)
G = Total number of grids in chip (given by ZyPSIM)
H = Standard height for library in grids
NP= Number of device pads

If a macro exceeds the maximum, it must be broken up and cascaded into several smaller macros. If necessary, the data sheets for the appropriate cells will describe any special cascading techniques.

6.2.6 MACRO COMBINATIONS AND NESTING

We have already discussed how regular macros may be nested (paragraph 6.2.1) but since other types have been introduced a summary of syntax rules is needed.

1. TELESCOPING macros may contain only telescoping digital cells.
2. ANALOG macros may contain only analog cells.
3. CLUSTER macros may contain only stand-alone digital cells.
4. REGULAR macros may contain stand-alone digital cells,
   or REGULAR macro CALLS
   or TELESCOPING macro CALLS
   or ANALOG macro CALLS
   or CLUSTER macro CALLS

Of course any regular, analog, etc. macros nested in a regular macro must be defined before they are called and only the call, not the definition may be nested within the macro.

The standard, telescoping, and mixed macro facilities of ZyPSIM and the ZyP system provide powerful hierarchical design capabilities for use in network files. The examples given in this section illustrate some of the techniques and rules for applying macros to user networks.

6.3 ROUTING CAPACITANCE

ZyPSIM, combined with ZyP cell models, provides simulation of silicon level performance. However, in the initial analysis of a network, the effects of routing capacitance are unknown because no actual composite has been generated. ZyPSIM can provide an estimate of routing capacitance in advance of the circuit mask design.

ZyMOS has algorithmically defined nodal capacitance, based on historical data, to allow users to automatically generate interconnect capacitance estimates. These estimates are added to the actual node capacitance values and can be inspected in the .XREF file described later.

To invoke this feature, insert the .INPUT command shown in Appendix A into the .NET file. Refer to the ZyPSIM Reference Manual (Document No. 20-010-209) for additional information on the .INPUT command.

Including parasitic capacitance estimates as early in the design process as possible will reduce functional problems caused by parasitics in the latter stages of network verification. Simulation with actual routing capacitances is always required prior to releasing a network for integration. Refer to the design guide for the appropriate library for additional information on routing capacitance.

When the .INPUT command is invoked in the .NET file as shown above, the ZyPSIM .ACE command is automatically included into the .NET file.

6.4 CROSS REFERENCE

ZyPSIM will produce a network cross reference file called filename.XRE when the .XREF command is included in the .NET file. This file summarizes cell nodal capacitance, including the routing estimate described in paragraph 6.3, as well as the fanout for each node. Section 8.5 lists the .XRE file for DEMO_TEST.NET.

6.5 GUARDBANDING

The .GUARDBAND command allows the user to specify circuit operating conditions during simulation. ZyMOS has included timing variables in the cell models that adjust the network timing for voltage, temperature, and processing effects.

The .GUARDBAND command must appear in the network prior to any network description. The syntax is as follows:

```
.GUARDBAND  VDD=<Vmin,Vmax> TEMP=<Tmin,Tmax> [CONDITION= <c>]
```

Where:

1.5 $\leq$ Vmin,Vmax $\leq$ 6.0 for the Zy40000 Library.
2.0 $\leq$ Vmin,Vmax $\leq$ 6.0 for the Zy50000 Library.
-55 $\leq$ Tmin,Tmax $\leq$ 125.
c = FAST, SLOW or NOM

EX: .GUARDBAND VDD=4.5,5.5 TEMP=0,70 CONDITION=FAST

If no .GUARDBAND command is included, or if the CONDITION keyword is not used, the simulation will run under the worst case process nominal conditions of VDD=5.0, TEMP=27 C, and SLOW process.

When "CONDITION=SLOW" is selected the operating conditions of Vmin and Tmax are used for simulation. The "CONDITION=FAST" option uses the Vmax and Tmin parameters. When used with the Test Language a "CONDITION=SLOW" results in a .TPS simulation output file, "CONDITION=FAST" results in a .TPF file, and "CONDITION=NOM" results in a .TPN file.

Note that the slow process models are also referred to as worst case.

When no .GUARDBAND command is included the default parameters are:

```
PROCESS=SLOW
VDD=5 V
TEMP=27 C
```

These conditions result in no modification to the timing data since they represent the cell characterization conditions.

6.6 SPIKES

A spike is defined as the condition which exists when the input to a gate changes state before the prior state has had time to propagate to the output. When this condition occurs, an S is printed in the left hand margin of the simulation output and a special symbol is printed in the print list if the spiking node is included in the print list. These special symbols are:

ZERO	.
ONE	'
INDETERMINATE	~

In addition, the filename.SPI file, automatically generated by ZyPSIM, will list the spike event by time and node name. This file should be analyzed after a simulation to determine if spike conditions present specific hazards to the functionality of the network.

Because of the nature of spike generating conditions and the fact that often they represent no functional problem, ZyPSIM provides a .NOSPIKE command to suppress spikes on a selected number of nodes. Refer to the ZyPSIM Reference Manual for details on using the .NOSPIKE command.

6.7 TOGGLE ANALYSIS

ZyPSIM provides Toggle Analysis capability to ZyP users. Toggle analysis gives an indication of the completeness of a test pattern, by identifing those signal nodes which make transitions to various logic states. The ZyPSIM analysis will identify nodes which take a logic "one" level, a logic "zero" level, a high impedance state, and those which are driven (relevant for three state signals). The output is formatted in a 6 column listing and is saved to a file suffixed .TOG. See Section 8.7 for an example .TOG file listing.

7 .NET FILE GUIDELINES

7.1 NETWORK ORGANIZATION

There are few restrictions imposed upon network file generation. However, rather than randomly inputting the logic description, some planning should precede the creation of the .NET file. Organization of the file contents will support a hierarchical design approach, communicate the design goals more effectively, and allow transfer of the design to production expeditiously. The following guidelines/suggestions are rendered as preferred approaches for implementing the .NET file. This file must be given the name filename.NET where filename can be any user-supplied name.

1. The first line of filename.NET must be the title line. This will appear at the top of every printed sheet when a 'hard-copy' of the file is obtained. This line will be ignored for simulation purposes.

2. The .GUARDBAND command applies a constant multiplier to all timing parameters in the standard cell models to adjust for process, voltage, and temperature effects. Since the logic/timing simulator program (ZyPSIM) works on a computer system having a one-pass compiler, the .GUARDBAND command should precede the usage of any and all standard cells, and is usually the second line of the .NET file.

3. The command line for artwork capacitance estimation is typically the third line of a ZyPSIM .NET file.

4. Special cells, if any, must have their logic/timing models in a local library. The .LIBRARY command has to occur prior to any of the special cells being called. ZyMOS suggests that the .LIBRARY command be the fourth line in your file.

5. The .PADS command provides a list of all signals associated with the pads (including the power supply connections VDD and VSS). Even though the primary purpose of .PADS is for artwork purposes and not logic verification, this command must be included in the .NET file, preferably near the beginning. A predefined pin-out (if it exists) for a circuit is indicated by following each pad signal name with a colon (:) and the associated pin number.

6. Next, a list of all pad cell usages; power, ground, inputs, outputs, and/or I/O's can be specified early in the file for convenience in locating them.

7. Where warranted, the use of macros is recommended. Macros support both system logic partitioning and artwork generation (i.e., place and route). In many cases reduced network coding is realized. Macro utilization manifest itself as a two step procedure:

   a) definition and b) usage

Therefore, the macro definition is required prior to invoking the macro cell in the circuit. Including all macro definitions prior to the network description allows them to be easily located within a common area of the file.

8. The network description is the standard cell realization of the customer's logic design. It indicates both functionality and interconnect, and must conform to the syntax as specified on the data sheet for each standard cell which is used. Macro usages (invocations) are included in this part of the file.

9. A user-supplied test pattern provides the input stimulus to exercise the circuit and effects logic/timing verification of the design. The input vectors are implemented by:

   a. .CLK or .TABLE commands for simulating portions of the design (See ZyPSIM Reference Manual, Document No. 20-010-209).

   b. .TTABLE tester commands for simulating the entire circuit (see ZyPSIM Test Language manual, Document No. 20-010-309).

   The final test program is also derived from the input pattern. Therefore, appropriate effort, judgment, and time should be invested in test pattern generation.

10. To run a simulation, additional commands are needed to direct the ZyPSIM program to certain actions; these are the simulator control commands.

    a. .PRINT command(s) direct output of user specified nodes to given file(s)

    b. Timing commands determine when the output will be printed or simulation terminated (this does not affect simulation resolution)

    c. Additional run commands

       1. Enable/inhibit diagnostic error messages.

       2. Call in additional supporting files.

       3. Create subsidiary files.

11. The .GO command tells the program to initiate a simulation. Make sure that all requisite information is included in the file prior to .GO since this command executes immediately. Placing .GO as the next to last command in filename.NET is recommended.

12. The last line of filename.NET must be .END. This tells ZyPSIM that no additional information is to follow or is to be read.

7.2 NETWORK FORMAT

The file contents should be arranged in a visually appealing and easy-to-read manner. The small amount of extra effort spent in preparing a systematic format can only result in enhanced clarity and readability. Comments should be included as needed or desired in order to explain the different sections of the .NET file. Comments can be used to:

1. Delineate different control statements.
2. Describe the functionality contained within the macro definitions.
3. Annotate the network description.
4. Characterize individual sections of the test pattern.

7.3 NETWORK SIMULATION

The following are suggestions for efficient design and computer cost control for ZyP applications. We begin by assuming the user is either ready for network generation or has completed network generation. ZyMOS has found the procedures below to minimize CPU usage and connect time when simulating networks with ZyPSIM.

If your circuit is large, then partitioning for simulation purposes is recommended. Try to partition as logically as possible so that a simulation will exercise a particular function. Each of the blocks can be debugged separately until all errors are eliminated and the nodes toggle as expected. The following steps are suggested for efficient design.

1. After defining your circuit blocks you should plan to simulate and debug them separately. Later you may easily load the blocks together into the final .NET file with the ZyPSIM .INPUT command. If you simulate each section independently the estimated parasitic routing capacitance values will be incorrect. This is because the actual chip's full size is not known. You can make the simulator believe the entire chip is included in the simulation by adding a .INFO statement that reflects the missing part. To do this you will need to know the grid count of the entire chip. This can be easily extracted by doing a complete coding (inputting all the blocks) of the circuit and compiling it with ZyPSIM. You can then create a .INFO statement as shown below and insert it into your block section simulation.

```
.INFO 70000
** where 70000 = (# of total grids - # of grids this block)
```

2. Do not forget to include the .INPUT statement for an estimation of the layout capacitance due to parasitics and the particular process you are using. This is extremely important, and when combined with the macro above, this section of the overall chip will realize the most accurate loading and parasitic interconnect estimates available.

3. For simulation of an independent circuit block you should attempt to duplicate input signal timing and output loading as it exists in the complete chip. The node loading information may be taken from the .XRE file following ZyPSIM compilation of the complete circuit. These considerations are very important since they will affect your results when you do comprehensive simulations. You can take the output signals from successful runs as the inputs to new sections. Refer to the ZyPSIM Reference Manual, (Document No. 20-010-209) for information on this technique. Remember to simulate each block over full operating and process conditions using the ZyPSIM .GUARDBAND command.

4. After you have simulated and are satisfied with your results you should begin merging the sections together. The best procedure is to start with the particular block that has the main clock and control signals and include the most closely affected logical block next. Using this technique you can begin building toward the final simulation of the entire chip. As more logic is included in the final .NET file you must compensate in the .INFO statement. The number will become smaller and smaller until it is eliminated from the final simulation.

5. When merging functional blocks some errors should still be expected. Usually this is the result of the signal timing between blocks no longer being ideal as in the previous simulations. During initial design and partitioning of your circuit do not forget to plan methods which simplify the testing of the complete chip.

8 SIMULATION EXAMPLE

The example network will demonstrate two different ways of implementing the same test pattern definition:

1. .CLK commands
2. Test language commands

The corresponding .NET files will be

DEMO.NET
DEMO_TEST.NET

8.1 DEMO.NET LISTING

This is the .NET file for the example circuit of Figure 2-1. It describes two identical counters, N1 to N17 and N18 to N34. The net list is typical of a "design" net list in that it contains no provisions for artwork generation or test program generation. At this stage, only 50 pF of external loading is added to the network file to simulate the loading effects of the Sentry tester. For the purposes of this example simulation, only one half of the counter has been simulated. The clock, enable and reset inputs of the second half are held inactive.

```
*ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)
*
*               NETWORK DESCRIPTION
*
*          PADS
*
CLOCK1       PAA1 CLK1 N15
ENABLE1      PAB1 EN1  N9
QA1          PBC1 N1   QA1
QB1          PBC1 N2   QB1
QC1          PBC1 N3   QC1
QD1          PBC1 N4   QD1
RESET1       PAB1 RST1 N10
VSS          PDD1 VSS  GROUND
CLOCK2       PAA1 CLK2 N32
ENABLE2      PAB1 EN2  N26
QA2          PBC1 N18  QA2
QB2          PBC1 N19  QB2
QC2          PBC1 N20  QC2
QD2          PBC1 N21  QD2
RESET2       PAB1 RST2 N27
VDD          PDC1 VDD  POWER
*
*          INVERTERS
*
N1  IAA3 N5  N1
N2  IAA3 N6  N2
N3  IAA3 N7  N3
```

```
N4  IAA3 N8  N4
N18 IAA3 N22 N18
N19 IAA3 N23 N19
N20 IAA3 N24 N20
N21 IAA3 N25 N21
N11 IAA3 N17 N11
N17 IAA3 N16 N17
N28 IAA3 N34 N28
N34 IAA3 N33 N34
*
*          NAND GATES
*
N16 GAA1 N15 N9  N16
N33 GAA1 N32 N26 N33
N12 GAA1 N5  N17 N12
N29 GAA1 N22 N34 N29
*
N13 GAE1 N5  N6  N17 N13
N30 GAE1 N22 N23 N34 N30
*
N14 GAG1 N5  N6  N7  N17 N14
N31 GAG1 N22 N23 N24 N34 N31
*
*          FLIP FLOPS
*
N5  FAG1 N11 N10 N5  NN5
N6  FAG1 N12 N10 N6  NN6
N7  FAG1 N13 N10 N7  NN7
N8  FAG1 N14 N10 N8  NN8
N22 FAG1 N28 N27 N22 NN22
N23 FAG1 N29 N27 N23 NN23
N24 FAG1 N30 N27 N24 NN24
N25 FAG1 N31 N27 N25 NN25
*
* 50 PF LOAD CAPACITORS (SENTRY LOADING)
*
CL1 CAP IN=QA1:50
CL2 CAP IN=QB1:50
CL3 CAP IN=QC1:50
CL4 CAP IN=QD1:50
CL5 CAP IN=QD2:50
CL6 CAP IN=QB2:50
CL7 CAP IN=QC2:50
CL8 CAP IN=QD2:50
*
*                    PATTERN DEFINITION
*
.CLK1 CLK1 100 200 R 100
.CLK1 CLK2
.CLK1 RST1 50 3350
.CLK1 RST2 50 3350
.CLK1 EN1 450 650
.CLK0 EN2
*
```

```
*                       RUN COMMANDS
*
.PRINT  RST1/N10/CLK1/N15/N16/N17///N11/N12/N13/N14///
+       N5/N6/N7/N8///QA1/QB1/QC1/QD1
*
.POC
.MAXTIME=3600
.GO
.END
```

8.2 DEMO.PRT LISTING

The simulation output shown below corresponds to the preceeding netlist. The output is printed using .POC which is typically used in the design phase to provide the smallest time interval resolution (normally 1 nS) while still minimizing output.

*ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)

TIME	RST1	N0	C1K	N5	N6	N7	N1	N2	N3	N4	N5	N6	N7	N8	QA1	QB1	QC1	QD1
0	1	1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
50	0	1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
61	0	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
100	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
111	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0
112	0	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0
120	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
200	0	0	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
205	0	0	1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0
208	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0
212	0	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0
214	0	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
231	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0
286	0	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	0	0
300	0	0	0	0	1	0	1	1	1	1	1	0	0	0	1	0	0	0
311	0	0	0	1	1	0	1	1	1	1	1	0	0	0	1	0	0	0
312	0	0	0	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0
320	0	0	0	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0
400	0	0	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0
405	0	0	1	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0
408	0	0	1	0	1	1	0	0	1	1	1	0	0	0	1	0	0	0
412	0	0	1	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0
414	0	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0	0	0
415	0	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	0	0
430	0	0	1	0	1	0	1	1	1	1	0	1	0	0	1	0	0	0
459	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0
485	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0	1	0	0
500	0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	1	0	0
511	0	0	0	1	1	0	1	1	1	1	0	1	0	0	0	1	0	0
600	0	0	1	1	1	0	1	1	1	1	0	1	0	0	0	1	0	0
605	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0	1	0	0
700	0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	1	0	0
711	0	0	0	1	1	0	1	1	1	1	0	1	0	0	0	1	0	0
712	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0	0
720	0	0	0	1	0	1	0	1	1	1	0	1	0	0	0	1	0	0

*ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)

TIME	RST1	N0	CLK1	N5	N6	N7	N1	N2	N3	N4	N5	N6	N7	N8	QA1	QB1	QC1	QD1
800	0	0	1	1	0	1	0	1	1	1	0	1	0	0	0	1	0	0
805	0	0	1	0	0	1	0	1	1	1	0	1	0	0	0	1	0	0
808	0	0	1	0	1	1	0	1	1	1	0	1	0	0	0	1	0	0
812	0	0	1	0	1	0	0	1	1	1	0	1	0	0	0	1	0	0
814	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0	1	0	0
831	0	0	1	0	1	0	1	1	1	1	1	1	0	0	0	1	0	0
886	0	0	1	0	1	0	1	1	1	1	1	1	0	0	1	1	0	0
900	0	0	0	0	1	0	1	1	1	1	1	1	0	0	1	1	0	0
911	0	0	0	1	1	0	1	1	1	1	1	1	0	0	1	1	0	0
912	0	0	0	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0
920	0	0	0	1	0	1	0	0	1	1	1	1	0	0	1	1	0	0
921	0	0	0	1	0	1	0	0	0	1	1	1	0	0	1	1	0	0
1000	0	0	1	1	0	1	0	0	0	1	1	1	0	0	1	1	0	0
1005	0	0	1	0	0	1	0	0	0	1	1	1	0	0	1	1	0	0
1008	0	0	1	0	1	1	0	0	0	1	1	1	0	0	1	1	0	0
1012	0	0	1	0	1	0	0	0	0	1	1	1	0	0	1	1	0	0
1014	0	0	1	0	1	0	1	0	0	1	1	1	0	0	1	1	0	0
1015	0	0	1	0	1	0	1	1	0	1	1	1	0	0	1	1	0	0
1016	0	0	1	0	1	0	1	1	1	1	1	1	0	0	1	1	0	0
1029	0	0	1	0	1	0	1	1	1	1	1	1	1	0	1	1	0	0
1030	0	0	1	0	1	0	1	1	1	1	0	0	1	0	1	1	0	0
1059	0	0	1	0	1	0	1	1	1	1	0	0	1	0	0	0	0	0
1084	0	0	1	0	1	0	1	1	1	1	0	0	1	0	0	0	1	0
1100	0	0	0	0	1	0	1	1	1	1	0	0	1	0	0	0	1	0
1111	0	0	0	1	1	0	1	1	1	1	0	0	1	0	0	0	1	0
1112	0	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	1	0
1120	0	0	0	1	0	1	0	1	1	1	0	0	1	0	0	0	1	0
1200	0	0	1	1	0	1	0	1	1	1	0	0	1	0	0	0	1	0
1205	0	0	1	0	0	1	0	1	1	1	0	0	1	0	0	0	1	0
1208	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	0	1	0
1212	0	0	1	0	1	0	0	1	1	1	0	0	1	0	0	0	1	0
1214	0	0	1	0	1	0	1	1	1	1	0	0	1	0	0	0	1	0
1231	0	0	1	0	1	0	1	1	1	1	1	0	1	0	0	0	1	0
1286	0	0	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	0
1300	0	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	0
1311	0	0	0	1	1	0	1	1	1	1	1	0	1	0	1	0	1	0
1312	0	0	0	1	0	0	1	1	1	1	1	0	1	0	1	0	1	0
1320	0	0	0	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0
1400	0	0	1	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0
1405	0	0	1	0	0	1	0	0	1	1	1	0	1	0	1	0	1	0
1408	0	0	1	0	1	1	0	0	1	1	1	0	1	0	1	0	1	0
1412	0	0	1	0	1	0	0	0	1	1	1	0	1	0	1	0	1	0
1414	0	0	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0
1415	0	0	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	0
1430	0	0	1	0	1	0	1	1	1	1	0	1	1	0	1	0	1	0
1459	0	0	1	0	1	0	1	1	1	1	0	1	1	0	0	0	1	0
1485	0	0	1	0	1	0	1	1	1	1	0	1	1	0	0	1	1	0

*ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)

TIME	RST1	N10	CLK1	N156	N157	N	N1	N2	N3	N4	N5	N6	N7	N8	QA1	QB1	QC1	QD1
1500	0	0	0	0	1	0	1	1	1	1	0	1	1	0	0	1	1	0
1511	0	0	0	1	1	0	1	1	1	1	0	1	1	0	0	1	1	0
1512	0	0	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0
1520	0	0	0	1	0	1	0	1	1	1	0	1	1	0	0	1	1	0
1600	0	0	1	1	0	1	0	1	1	1	0	1	1	0	0	1	1	0
1605	0	0	1	0	0	1	0	1	1	1	0	1	1	0	0	1	1	0
1608	0	0	1	0	1	1	0	1	1	1	0	1	1	0	0	1	1	0
1612	0	0	1	0	1	0	0	1	1	1	0	1	1	0	0	1	1	0
1614	0	0	1	0	1	0	1	1	1	1	0	1	1	0	0	1	1	0
1631	0	0	1	0	1	0	1	1	1	1	1	1	1	0	0	1	1	0
1686	0	0	1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	0
1700	0	0	0	0	1	0	1	1	1	1	1	1	1	0	1	1	1	0
1711	0	0	0	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0
1712	0	0	0	1	0	0	1	1	1	1	1	1	1	0	1	1	1	0
1720	0	0	0	1	0	1	0	0	1	1	1	1	1	0	1	1	1	0
1721	0	0	0	1	0	1	0	0	0	1	1	1	1	0	1	1	1	0
1722	0	0	0	1	0	1	0	0	0	0	1	1	1	0	1	1	1	0
1800	0	0	1	1	0	1	0	0	0	0	1	1	1	0	1	1	1	0
1805	0	0	1	0	0	1	0	0	0	0	1	1	1	0	1	1	1	0
1808	0	0	1	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0
1812	0	0	1	0	1	0	0	0	0	0	1	1	1	0	1	1	1	0
1814	0	0	1	0	1	0	1	0	0	0	1	1	1	0	1	1	1	0
1815	0	0	1	0	1	0	1	1	0	0	1	1	1	0	1	1	1	0
1816	0	0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0
1817	0	0	1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	0
1828	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0
1829	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	0
1830	0	0	1	0	1	0	1	1	1	1	0	0	0	1	1	1	1	0
1858	0	0	1	0	1	0	1	1	1	1	0	0	0	1	1	1	0	0
1859	0	0	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0	0
1883	0	0	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0	1
1900	0	0	0	0	1	0	1	1	1	1	0	0	0	1	0	0	0	1
1911	0	0	0	1	1	0	1	1	1	1	0	0	0	1	0	0	0	1
1912	0	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1
1920	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0	0	0	1
2000	0	0	1	1	0	1	0	1	1	1	0	0	0	1	0	0	0	1
2005	0	0	1	0	0	1	0	1	1	1	0	0	0	1	0	0	0	1
2008	0	0	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0	1
2012	0	0	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0	1
2014	0	0	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0	1
2031	0	0	1	0	1	0	1	1	1	1	1	0	0	1	0	0	0	1
2086	0	0	1	0	1	0	1	1	1	1	1	0	0	1	1	0	0	1

*ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)

TIME	RST1	N10	CLK1	N15	N16	N17	N12	N13	N14	N56	N67	N78	QA11	QB11	QC11	QD11
2100	0	0	0	0	1	0	1	1	1	1	0	0	1	0	0	1
2111	0	0	0	1	1	0	1	1	1	1	0	0	1	0	0	1
2112	0	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1
2120	0	0	0	1	0	1	0	0	1	1	0	0	1	0	0	1
2200	0	0	1	1	0	1	0	0	1	1	0	0	1	0	0	1
2205	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0	1
2208	0	0	1	0	1	1	0	0	1	1	0	0	1	0	0	1
2212	0	0	1	0	1	0	0	0	1	1	0	0	1	0	0	1
2214	0	0	1	0	1	0	1	0	1	1	0	0	1	0	0	1
2215	0	0	1	0	1	0	1	1	1	1	0	0	1	0	0	1
2230	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0	0	1
2259	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0	0	0	1
2285	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	0	1
2300	0	0	0	0	1	0	1	1	1	1	0	1	0	1	0	1	0	1
2311	0	0	0	1	1	0	1	1	1	1	0	1	0	1	0	1	0	1
2312	0	0	0	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1
2320	0	0	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1
2400	0	0	1	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1
2405	0	0	1	0	0	1	0	1	1	1	0	1	0	1	0	1	0	1
2408	0	0	1	0	1	1	0	1	1	1	0	1	0	1	0	1	0	1
2412	0	0	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0	1
2414	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	0	1
2431	0	0	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0	1
2486	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	1	0	1
2500	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	1	0	1
2511	0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	0	1
2512	0	0	0	1	0	0	1	1	1	1	1	1	0	1	1	1	0	1
2520	0	0	0	1	0	1	0	0	1	1	1	1	0	1	1	1	0	1
2521	0	0	0	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1
2600	0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1
2605	0	0	1	0	0	1	0	0	0	1	1	1	0	1	1	1	0	1
2608	0	0	1	0	1	1	0	0	0	1	1	1	0	1	1	1	0	1
2612	0	0	1	0	1	0	0	0	0	1	1	1	0	1	1	1	0	1
2614	0	0	1	0	1	0	1	0	0	1	1	1	0	1	1	1	0	1
2615	0	0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1
2616	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	1	0	1
2629	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1
2630	0	0	1	0	1	0	1	1	1	1	0	0	1	1	1	0	1
2659	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1
2684	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1
2700	0	0	0	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1
2711	0	0	0	1	1	0	1	1	1	1	0	0	1	1	0	0	1	1
2712	0	0	0	1	0	0	1	1	1	1	0	0	1	1	0	0	1	1
2720	0	0	0	1	0	1	0	1	1	1	0	0	1	1	0	0	1	1

#ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)

TIME	RST1	N10	CLK1	N15	N16	N17	N11	N12	N13	N14	N15	N16	N17	N18	QA11	QB11	QC11	QD11
2800	0	0	1	1	0	1	0	1	1	1	0	0	1	1	0	0	1	1
2805	0	0	1	0	0	1	0	1	1	1	0	0	1	1	0	0	1	1
2808	0	0	1	0	1	1	0	1	1	1	0	0	1	1	0	0	1	1
2812	0	0	1	0	1	0	0	1	1	1	0	0	1	1	0	0	1	1
2814	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1
2831	0	0	1	0	1	0	1	1	1	1	1	0	1	1	0	0	1	1
2886	0	0	1	0	1	0	1	1	1	1	1	0	1	1	1	0	1	1
2900	0	0	0	0	1	0	1	1	1	1	1	0	1	1	1	0	1	1
2911	0	0	0	1	1	0	1	1	1	1	1	0	1	1	1	0	1	1
2912	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	1	1
2920	0	0	0	1	0	1	0	0	1	1	1	0	1	1	1	0	1	1
3000	0	0	1	1	0	1	0	0	1	1	1	0	1	1	1	0	1	1
3005	0	0	1	0	0	1	0	0	1	1	1	0	1	1	1	0	1	1
3008	0	0	1	0	1	1	0	0	1	1	1	0	1	1	1	0	1	1
3012	0	0	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1
3014	0	0	1	0	1	0	1	0	1	1	1	0	1	1	1	0	1	1
3015	0	0	1	0	1	0	1	1	1	1	1	0	1	1	1	0	1	1
3030	0	0	1	0	1	0	1	1	1	1	0	1	1	1	1	0	1	1
3059	0	0	1	0	1	0	1	1	1	1	0	1	1	1	0	0	1	1
3085	0	0	1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
3100	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
3111	0	0	0	1	1	0	1	1	1	1	0	1	1	1	0	1	1	1
3112	0	0	0	1	0	0	1	1	1	1	0	1	1	1	0	1	1	1
3120	0	0	0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1
3200	0	0	1	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1
3205	0	0	1	0	0	1	0	1	1	1	0	1	1	1	0	1	1	1
3208	0	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1	1
3212	0	0	1	0	1	0	0	1	1	1	0	1	1	1	0	1	1	1
3214	0	0	1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
3231	0	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1	1	1
3286	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

*ZYPSIM Zy50000 CHMOS DEMO (NORMAL .POC SIMULATION)

TIME	RST1	N0	CLK1	N5	N6	N7	N11	N12	N13	N14	N5	N6	N7	N8	QA1	QB1	QC1	QD1
3300	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
3311	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
3312	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
3320	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1
3321	0	0	0	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1
3322	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1	1	1	1
3350	1	0	0	1	0	1	0	0	0	0	1	1	1	1	1	1	1	1
3358	1	1	0	1	0	1	0	0	0	0	1	1	1	1	1	1	1	1
3364	1	1	0	1	0	1	0	0	0	0	1	1	1	0	1	1	1	1
3366	1	1	0	1	0	1	0	0	0	0	1	1	0	0	1	1	1	1
3368	1	1	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	1
3369	1	1	0	1	0	1	0	0	0	0	0	0	0	0	1	1	1	1
3371	1	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	1	1
3372	1	1	0	1	0	1	0	1	1	1	0	0	0	0	1	1	1	1
3393	1	1	0	1	0	1	0	1	1	1	0	0	0	0	1	1	1	0
3395	1	1	0	1	0	1	0	1	1	1	0	0	0	0	1	1	0	0
3397	1	1	0	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0
3398	1	1	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
3400	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
3405	1	1	1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0
3408	1	1	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0
3412	1	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0
3414	1	1	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
3500	1	1	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
3511	1	1	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0
3512	1	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0
3520	1	1	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
3600	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0

8.3 DEMO_TEST.NET LISTING

The following netlist describes a fully coded, transfer ready file of the same 4520 dual four bit counter from the DEMO example. Note that the ZyPSIM test language is used to guarantee testable patterns and that guardbanding is performed to consider operating condition variations. To account for artwork parasitics, capacitance estimation has been included (Prime specific). Note that a .TOGGLE command is also included and will produce a toggle analysis output file (.TOG).

```
*ZYPSIM Zy50000 CHMOS DEMO USING THE TEST LANGUAGE
*
* NOTE: THE SIMULATION USES WORST CASE COMMERCIAL ENVIRONMENTAL
* CONDITIONS AND INCLUDES ESTIMATED INTERCONNECT CAPACITANCE.
* A TOGGLE ANALYSIS OUTPUT REPORT IS ALSO REQUESTED.
*
.GUARDBAND VDD=4.5,5.5 TEMP=0,70 CONDITION=NOM
.INPUT ZYP>ZYPSIM>ZY5LIB>LAYOUTCAP
.TOGGLE
*
*        NETWORK DESCRIPTION
*
*          PIN ASSIGNMENT
*
.PADS CLK1:1 EN1:2  QA1:3  QB1:4  QC1:5  QD1:6  RST1:7  VSS:8
+     CLK2:9 EN2:10 QA2:11 QB2:12 QC2:13 QD2:14 RST2:15 VDD:16
*
*
*          PADS
*
CLOCK1       PAA1 CLK1 N15
ENABLE1      PAB1 EN1  N9
QA1          PBC1 N1   QA1
QB1          PBC1 N2   QB1
QC1          PBC1 N3   QC1
QD1          PBC1 N4   QD1
RESET1       PAB1 RST1 N10
VSS          PDD1 VSS  GROUND
CLOCK2       PAA1 CLK2 N32
ENABLE2      PAB1 EN2  N26
QA2          PBC1 N18  QA2
QB2          PBC1 N19  QB2
QC2          PBC1 N20  QC2
QD2          PBC1 N21  QD2
RESET2       PAB1 RST2 N27
VDD          PDC1 VDD  POWER
*
*          INVERTERS
*
N1  IAA3 N5  N1
N2  IAA3 N6  N2
N3  IAA3 N7  N3
N4  IAA3 N8  N4
N18 IAA3 N22 N18
```

```
N19 IAA3 N23 N19
N20 IAA3 N24 N20
N21 IAA3 N25 N21
N11 IAA3 N17 N11
N17 IAA3 N16 N17
N28 IAA3 N34 N28
N34 IAA3 N33 N34
*
*          NAND GATES
*
N16 GAA1 N15 N9  N16
N33 GAA1 N32 N26 N33
N12 GAA1 N5  N17 N12
N29 GAA1 N22 N34 N29
*
N13 GAE1 N5  N6  N17 N13
N30 GAE1 N22 N23 N34 N30
*
N14 GAG1 N5  N6  N7  N17 N14
N31 GAG1 N22 N23 N24 N34 N31
*
*          FLIP FLOPS
*
N5  FAG1 N11 N10 N5  NN5
N6  FAG1 N12 N10 N6  NN6
N7  FAG1 N13 N10 N7  NN7
N8  FAG1 N14 N10 N8  NN8
N22 FAG1 N28 N27 N22 NN22
N23 FAG1 N29 N27 N23 NN23
N24 FAG1 N30 N27 N24 NN24
N25 FAG1 N31 N27 N25 NN25
*
* 50 PF LOAD CAPACITORS (SENTRY LOADING)
*
CL1 CAP IN=QA1:50
CL2 CAP IN=QB1:50
CL3 CAP IN=QC1:50
CL4 CAP IN=QD1:50
CL5 CAP IN=QA2:50
CL6 CAP IN=QB2:50
CL7 CAP IN=QC2:50
CL8 CAP IN=QD2:50
*
*                    PATTERN DEFINITION
*
.TESTER SENTRY7                    % DEFINE TESTER
*
.TPERIOD 250                       % 4MHz DEVICE
*
* CLOCK TIMING AND DATA
*
.TGEN RZ 10 135 CLK1 CLK2
*
.TTABLE CLK1 CLK2
```

```
   18     1    1                  % CHECK COUNTER AND POSITIVE CLOCK
   15     1    1                  % PREPARE FOR INHIBIT/RESET TEST
    3     0    0
*
* RESET TIMING AND DATA
*
.TGEN NRZ 50 RST1 RST2
*
.TTABLE RST1,1 RST2,1
    1     1    1                  % START WITH A RESET
    1     0    0                  % REMOVE RESET IN CYCLE 2
   33     0    0
    1     1    1                  % END WITH A RESET
*
* ENABLE TIMING AND DATA
*
.TGEN NRZ 15  EN1  EN2
*
.TTABLE EN1,1  EN2,1
   31     1    1
    2     0    0                  % CHECK INHIBIT
    1     1    1
    2     0    0                  % CHECK NEGATIVE CLOCK
*
* MASK DATA
*
.TTABLE MASK
    1     0                       % MASK DURING INITIAL RESET
   17     1
   13     0                       % REPETITIVE DATA MASK
    5     1
*
*                    RUN COMMANDS
*
*
.TSTROBE 225   QA1,MASK  QB1,MASK  QC1,MASK  QD1,MASK
+              QA2,MASK  QB2,MASK  QC2,MASK  QD2,MASK
*
.TPRINT  CLK1  EN1  QA1  QB1  QC1  QD1  RST1  VSS
+        CLK2  EN2  QA2  QB2  QC2  QD2  RST2  VDD
*
*
.MAXTIME=9000                      % 36 VECTORS * TPERIOD
.XREF
.GO
.END
```

8.4 DEMO_TEST.OUT LISTING

This is the .OUT file generated by ZyPSIM as a result of running DEMO_TEST with the Zy50000 Series CHMOS library.

```
 1  *ZYPSIM Zy50000 CHMOS DEMO USING THE TEST LANGUAGE
 2  *
 3  * NOTE: THE SIMULATION USES WORST CASE COMMERCIAL ENVIRONMENTAL
 4  * CONDITIONS AND INCLUDES ESTIMATED INTERCONNECT CAPACITANCE.
 5  * A TOGGLE ANALYSIS OUTPUT REPORT IS ALSO REQUESTED.
 6  *
 7  .GUARDBAND VDD=4.5,5.5 TEMP=0,70 CONDITION=NOM
 8  .INPUT ZYP>ZYPSIM>ZY5LIB>LAYOUTCAP
 9  *
10  *          3u SI GATE CHMOS ARTWORK CAPACITANCE ESTIMATION
11  *
12  *
13  * THE FOLLOWING COMMAND CAUSES CAPACITANCE TO BE ADDED TO
14  * EACH NODE BASED ON FANOUT, AND IS AN ESTIMATE OF THE FINAL
15  * ARTWORK CAPACITANCE.
16  *
17  .ACE 5 1 .1 2 .125 3 .15 4 .25 5 .3 .00707
18  *
19  * THE VALUE GENERATED WILL BE K * .00707 * SQRT(#GRIDS)
20  *
21  * WHICH IS NORMALIZED TO 20000 GRIDS IE, K*SQRT(GRIDS/20000) HENCE .0070
22  *
23  * K = .1   FANOUT  = 1
24  * K = .125 FANOUT  = 2
25  * K = .15  FANOUT  = 3
26  * K = .25  FANOUT  = 4
27  * K = .3   FANOUT >= 5
28  *
29  * FOR EXAMPLE IF YOUR CHIP SIZE IS 120000 GRIDS, A NODE DRIVING 4
30  * INPUTS WILL HAVE AN ESTIMATED VALUE OF .25*.00707*SQRT(120000)=.61 pF.
31  *
32  * TOO SEE THE VALUES GENERATED, INCLUDE THE COMMAND .XREF IN YOUR
33  * NETLIST. THE FILE FNAME.XREF WILL CONTAIN THE CAP VALUES AND OTHER
34  * USEFUL TIMING AND FANOUT DATA.
35  *
36  * THE VALUES FOR K WILL BE UPDATED FROM TIME TO TIME AS MORE DATA IS
37  * ADDED TO THE DEVICE DATA BASE.
38  *
39  * LAST UPDATED 04/19/83    STU.
40  *
41  .TOGGLE
42  *
43  *        NETWORK DESCRIPTION
44  *
45  *          PIN ASSIGNMENT
46  *
47  .PADS CLK1:1 EN1:2  QA1:3  QB1:4  QC1:5  QD1:6  RST1:7  VSS:8
48  +     CLK2:9 EN2:10 QA2:11 QB2:12 QC2:13 QD2:14 RST2:15 VDD:16
49  *
```

```
50  *
51  *           PADS
52  *
53  CLOCK1        PAA1 CLK1 N15
54  ENABLE1       PAB1 EN1  N9
55  QA1           PBC1 N1   QA1
56  QB1           PBC1 N2   QB1
57  QC1           PBC1 N3   QC1
58  QD1           PBC1 N4   QD1
59  RESET1        PAB1 RST1 N10
60  VSS           PDD1 VSS  GROUND
61  CLOCK2        PAA1 CLK2 N32
62  ENABLE2       PAB1 EN2  N26
63  QA2           PBC1 N18  QA2
64  QB2           PBC1 N19  QB2
65  QC2           PBC1 N20  QC2
66  QD2           PBC1 N21  QD2
67  RESET2        PAB1 RST2 N27
68  VDD           PDC1 VDD  POWER
69  *
70  *           INVERTERS
71  *
72  N1  IAA3 N5  N1
73  N2  IAA3 N6  N2
74  N3  IAA3 N7  N3
75  N4  IAA3 N8  N4
76  N18 IAA3 N22 N18
77  N19 IAA3 N23 N19
78  N20 IAA3 N24 N20
79  N21 IAA3 N25 N21
80  N11 IAA3 N17 N11
81  N17 IAA3 N16 N17
82  N28 IAA3 N34 N28
83  N34 IAA3 N33 N34
84  *
85  *           NAND GATES
86  *
87  N16 GAA1 N15 N9  N16
88  N33 GAA1 N32 N26 N33
89  N12 GAA1 N5  N17 N12
90  N29 GAA1 N22 N34 N29
91  *
92  N13 GAE1 N5  N6  N17 N13
93  N30 GAE1 N22 N23 N34 N30
94  *
95  N14 GAG1 N5  N6  N7  N17 N14
96  N31 GAG1 N22 N23 N24 N34 N31
97  *
98  *           FLIP FLOPS
99  *
100 N5  FAG1 N11 N10 N5  NN5
101 N6  FAG1 N12 N10 N6  NN6
102 N7  FAG1 N13 N10 N7  NN7
103 N8  FAG1 N14 N10 N8  NN8
```

```
104  N22 FAG1 N28 N27 N22 NN22
105  N23 FAG1 N29 N27 N23 NN23
106  N24 FAG1 N30 N27 N24 NN24
107  N25 FAG1 N31 N27 N25 NN25
108  *
109  * 50 PF LOAD CAPACITORS (SENTRY LOADING)
110  *
111  CL1 CAP IN=QA1:50
112  CL2 CAP IN=QB1:50
113  CL3 CAP IN=QC1:50
114  CL4 CAP IN=QD1:50
115  CL5 CAP IN=QA2:50
116  CL6 CAP IN=QB2:50
117  CL7 CAP IN=QC2:50
118  CL8 CAP IN=QD2:50
119  *
120  *                    PATTERN DEFINITION
121  *
122  .TESTER SENTRY7                    % DEFINE TESTER
123  *
124  .TPERIOD 250                       % 4MHz DEVICE
125  *
126  * CLOCK TIMING AND DATA
127  *
128  .TGEN RZ 10 135 CLK1 CLK2
129  *
130  .TTABLE CLK1 CLK2
131      18     1    1                 % CHECK COUNTER AND POSITIVE CLOCK
     LOCAL COUNT =      0
     LOCAL COUNT =     18
132      15     1    1                 % PREPARE FOR INHIBIT/RESET TEST
     LOCAL COUNT =     33
133       3     0    0
     LOCAL COUNT =     36
134  *
135  * RESET TIMING AND DATA
136  *
137  .TGEN NRZ 50 RST1 RST2
138  *
139  .TTABLE RST1,1 RST2,1
140       1     1    1                 % START WITH A RESET
     LOCAL COUNT =      0
     LOCAL COUNT =      1
141       1     0    0                 % REMOVE RESET IN CYCLE 2
     LOCAL COUNT =      2
142      33     0    0
     LOCAL COUNT =     35
143       1     1    1                 % END WITH A RESET
     LOCAL COUNT =     36
144  *
145  * ENABLE TIMING AND DATA
146  *
147  .TGEN NRZ 15  EN1  EN2
148  *
```

```
149  .TTABLE EN1,1  EN2,1
150     31      1     1
     LOCAL COUNT =       0
     LOCAL COUNT =      31
151      2      0     0                    % CHECK INHIBIT
     LOCAL COUNT =      33
152      1      1     1
     LOCAL COUNT =      34
153      2      0     0                    % CHECK NEGATIVE CLOCK
     LOCAL COUNT =      36
154  *
155  * MASK DATA
156  *
157  .TTABLE MASK
158      1      0                          % MASK DURING INITIAL RESET
     LOCAL COUNT =       0
     LOCAL COUNT =       1
159     17      1
     LOCAL COUNT =      18
160     13      0                          % REPETITIVE DATA MASK
     LOCAL COUNT =      31
161      5      1
     LOCAL COUNT =      36
162  *
163  *                      RUN COMMANDS
164  *
165  *
166  .TSTROBE 225    QA1,MASK  QB1,MASK  QC1,MASK  QD1,MASK
167  +               QA2,MASK  QB2,MASK  QC2,MASK  QD2,MASK
168  *
169  .TPRINT  CLK1  EN1  QA1  QB1  QC1  QD1  RST1  VSS
170  +        CLK2  EN2  QA2  QB2  QC2  QD2  RST2  VDD
171  *
172  *
173  .MAXTIME=9000                         % 36 VECTORS * TPERIOD
174  .XREF
175  .GO
```

```
CIRCUIT HAS BEEN COMPILED

  NUMBER OF SIGNALS           =    157
  NUMBER OF MACRO DEFINITIONS =     10
  NUMBER OF GRIDS             =   7200
  NUMBER OF TRANSISTORS       =    336
  GUARDBAND SCALE FACTOR      =   1.00

BASIC          # OF       # OF
CELL        OCCURENCES   GRIDS


PAA1             2         512
PAB1             4        1136
PBC1             8        3016
PDD1             1         220
PDC1             1         126
IAA3            12         336
GAA1             4         180
GAE1             2         120
GAG1             2         154
FAG1             8        1400

NUMBER OF GRIDS IN STANDARD CELLS:       7200
NUMBER OF GRIDS IN TELESCOPING MACROS:      0


  176  .END
---END OF ZYPSIM -- NO ERRORS---
```

8.5 DEMO_TEST.XRE LISTING

The following listing shows the .XRE file output produced when the .XRE command is used in the net list. The first section shows the capacitance on each node broken down into several components. CELL is due to cell input capacitance, LINE is due to CAP primitives normally produced by loading artwork capacitance after routing, and EST shows values generated when the capacitance estimation feature is used. .XRE is the only way to determine the values which have been inserted by ZyPSIM. The listing below shows the values produced when estimation was used on DEMO_TEST.NET.

Two sets of timing values are also shown, absolute and sensitivity/pF. Absolute values show the propagation delays and rise/fall times on each node produced by the indicated driving gate and total capacitance column. The sensitivity values show how the timing would change for a 1 pF change in capacitance. This is useful for detecting potential critical nodes. The absolute rise/fall time data can be used to detect potential high power nodes.

The second listing shows the fanout of each node, indicating the cell type of the driving and driven nodes. This is useful in continuity checking when first debugging the net list.

		NODE CAPACITANCE				ABSOLUTE DELAY				SENSITIVITY/PF			
NODENAME	DRIVING GATE	TOT	CELL	LINE	EST	0-1	1-0	TR	TF	0-1	1-0	TR	TF
N1	N1.IAA3	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N10	RESET1.	0.87	0.72	0.00	0.15	9	11	3	3	2	1	3	1
N11	N11.IAA	0.23	0.17	0.00	0.06	2	0	3	2	3	2	4	1
N12	N12.GAA	0.23	0.17	0.00	0.06	3	0	4	3	2	2	2	1
N13	N13.GAE	0.23	0.17	0.00	0.06	4	1	5	3	2	2	2	2
N14	N14.GAG	0.23	0.17	0.00	0.06	5	2	6	4	2	2	2	2
N15	CLOCK1.	0.55	0.49	0.00	0.06	12	5	7	6	9	3	12	3
N16	N16.GAA	0.38	0.32	0.00	0.06	4	1	4	3	2	2	2	1
N17	N17.IAA	2.00	1.85	0.00	0.15	8	4	3	2	3	2	4	1
N18	N18.IAA	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N19	N19.IAA	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N2	N2.IAA3	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N20	N20.IAA	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N21	N21.IAA	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N22	N22.FAG	2.00	1.85	0.00	0.15	13	12	3	4	3	3	4	3
N23	N23.FAG	1.45	1.36	0.00	0.09	11	10	3	4	3	3	4	3
N24	N24.FAG	0.94	0.87	0.00	0.07	9	8	3	4	3	3	4	3
N25	N25.FAG	0.38	0.32	0.00	0.06	6	6	3	4	3	3	4	3
N26	ENABLE2	0.55	0.49	0.00	0.06	8	11	3	3	2	1	3	1
N27	RESET2.	0.87	0.72	0.00	0.15	9	11	3	3	2	1	3	1
N28	N28.IAA	0.23	0.17	0.00	0.06	2	0	3	2	3	2	4	1
N29	N29.GAA	0.23	0.17	0.00	0.06	3	0	4	3	2	2	2	1
N3	N3.IAA3	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N30	N30.GAE	0.23	0.17	0.00	0.06	4	1	5	3	2	2	2	2
N31	N31.GAG	0.23	0.17	0.00	0.06	5	2	6	4	2	2	2	2
N32	CLOCK2.	0.55	0.49	0.00	0.06	12	5	7	6	9	3	12	3
N33	N33.GAA	0.38	0.32	0.00	0.06	4	1	4	3	2	2	2	1

N34	N34.IAA	2.00	1.85	0.00	0.15	8	4	3	2	3	2	4	1
N4	N4.IAA3	1.53	1.47	0.00	0.06	7	3	3	2	3	2	4	1
N5	N5.FAG1	2.00	1.85	0.00	0.15	13	12	3	4	3	3	4	3
N6	N6.FAG1	1.45	1.36	0.00	0.09	11	10	3	4	3	3	4	3
N7	N7.FAG1	0.94	0.87	0.00	0.07	9	8	3	4	3	3	4	3
N8	N8.FAG1	0.38	0.32	0.00	0.06	6	6	3	4	3	3	4	3
N9	ENABLE1	0.55	0.49	0.00	0.06	8	11	3	3	2	1	3	1
NN22	N22.FAG	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN23	N23.FAG	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN24	N24.FAG	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN25	N25.FAG	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN5	N5.FAG1	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN6	N6.FAG1	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN7	N7.FAG1	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
NN8	N8.FAG1	0.00	0.00	0.00	0.00	10	13	3	4	3	2	4	2
QA1	QA1.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QA2	QA2.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QB1	QB1.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QB2	QB2.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QC1	QC1.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QC2	QC2.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QD1	QD1.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0
QD2	QD2.PBC	50.06	0.00	50.00	0.06	52	23	4	3	0	0	0	0

N1	N1.IAA3	QA1.PBC1
N10	RESET1.PAB1	N8.FAG1_R_TO_CL N7.FAG1_R_TO_CL N6.FAG1_R_TO_CL N5.FAG1_R_TO_CL
N11	N11.IAA3	N5.FAG1_CLK_SLO
N12	N12.GAA1	N6.FAG1_CLK_SLO
N13	N13.GAE1	N7.FAG1_CLK_SLO
N14	N14.GAG1	N8.FAG1_CLK_SLO
N15	CLOCK1.PAA1	N16.GAA1
N16	N16.GAA1	N17.IAA3
N17	N17.IAA3	N14.GAG1 N13.GAE1 N12.GAA1 N11.IAA3
N18	N18.IAA3	QA2.PBC1
N19	N19.IAA3	QB2.PBC1
N2	N2.IAA3	QB1.PBC1
N20	N20.IAA3	QC2.PBC1
N21	N21.IAA3	QD2.PBC1

N22	N22.FAG1_QBUF	N31.GAG1	N30.GAE1
		N29.GAA1	N18.IAA3
N23	N23.FAG1_QBUF	N31.GAG1	N30.GAE1
		N19.IAA3	
N24	N24.FAG1_QBUF	N31.GAG1	N20.IAA3
N25	N25.FAG1_QBUF	N21.IAA3	
N26	ENABLE2.PAB1	N33.GAA1	
N27	RESET2.PAB1	N25.FAG1_R_TO_C	N24.FAG1_R_TO_C
		N23.FAG1_R_TO_C	N22.FAG1_R_TO_C
N28	N28.IAA3	N22.FAG1_CLK_SL	
N29	N29.GAA1	N23.FAG1_CLK_SL	
N3	N3.IAA3	QC1.PBC1	
N30	N30.GAE1	N24.FAG1_CLK_SL	
N31	N31.GAG1	N25.FAG1_CLK_SL	
N32	CLOCK2.PAA1	N33.GAA1	
N33	N33.GAA1	N34.IAA3	
N34	N34.IAA3	N31.GAG1	N30.GAE1
		N29.GAA1	N28.IAA3
N4	N4.IAA3	QD1.PBC1	
N5	N5.FAG1_QBUF	N14.GAG1	N13.GAE1
		N12.GAA1	N1.IAA3
N6	N6.FAG1_QBUF	N14.GAG1	N13.GAE1
		N2.IAA3	
N7	N7.FAG1_QBUF	N14.GAG1	N3.IAA3
N8	N8.FAG1_QBUF	N4.IAA3	
N9	ENABLE1.PAB1	N16.GAA1	
NN22	N22.FAG1_NQBUF		
NN23	N23.FAG1_NQBUF		
NN24	N24.FAG1_NQBUF		
NN25	N25.FAG1_NQBUF		

NN5	N5.FAG1_NQBUF	
NN6	N6.FAG1_NQBUF	
NN7	N7.FAG1_NQBUF	
NN8	N8.FAG1_NQBUF	
QA1	QA1.PBC1	$0004
QA2	QA2.PBC1	$0004
QB1	QB1.PBC1	$0004
QB2	QB2.PBC1	$0004
QC1	QC1.PBC1	$0004
QC2	QC2.PBC1	$0004
QD1	QD1.PBC1	$0004
QD2	QD2.PBC1	$0004

8.6 DEMO_TEST.TPN LISTING

Listed below is the .TPN file for the DEMO_TEST.NET example. The .TPS and .TPN files created by different guardbanding will have an identical format. Refer to the ZyPSIM Test Language document for information on interpreting the data within this file.

```
*ZYPSIM Zy50000 CHMOS DEMO USING THE TEST LANGUAGE

    CEQQQQRVCEQQQQRV
    LNABCDSSLNABCDSD
    K11111TSK22222TD
    1     1 2     2


CYCLE

00000 dDmmmmDmdDmmmmDm
00001 DDmmmmDmDDmmmmDm
00002 DD0000dmDD0000dm
00003 DD1000dmDD1000dm
00004 DD0100dmDD0100dm
00005 DD1100dmDD1100dm
00006 DD0010dmDD0010dm
00007 DD1010dmDD1010dm
00008 DD0110dmDD0110dm
00009 DD1110dmDD1110dm
00010 DD0001dmDD0001dm
00011 DD1001dmDD1001dm
00012 DD0101dmDD0101dm
00013 DD1101dmDD1101dm
00014 DD0011dmDD0011dm
00015 DD1011dmDD1011dm
00016 DD0111dmDD0111dm
00017 DD1111dmDD1111dm
00018 DD0000dmDD0000dm
00019 DDMmmmdmDDMmmmdm
00020 DDmMmmdmDDmMmmdm
00021 DDMMmmdmDDMMmmdm
00022 DDmmMmdmDDmmMmdm
00023 DDMmMmdmDDMmMmdm
00024 DDmMMmdmDDmMMmdm
00025 DDMMMmdmDDMMMmdm
00026 DDmmmMdmDDmmmMdm
00027 DDMmmMdmDDMmmMdm
00028 DDmMmMdmDDmMmMdm
00029 DDMMmMdmDDMMmMdm
00030 DDmmMMdmDDmmMMdm
00031 DDMmMMdmDDMmMMdm
00032 Dd0111dmDd0111dm
00033 Dd0111dmDd0111dm
00034 dD0111dmdD0111dm
00035 dd1111dmdd1111dm
00036 dd0000Dmdd0000Dm
```

8.7 DEMO_TEST.TOG LISTING

Listed below is the toggle analysis output file (.TOG) for the DEMO_TEST.NET circuit. The first data column of this file is a reference sequence number. The second column lists every interconnect signal node between ZyP standard cells. The third column indicates those signals which have had a one or high logic level during simulation. The fourth column identifies nodes which have taken a logic zero state. The fifth column, which is not shown, would flag nodes that have taken a high impedance state (there are no tri-state nodes in this circuit). While the last column identifies nodes which have been driven to a solid logic level during the simulation.

*ZYPSIM Zy50000 CHMOS DEMO USING THE TEST LANGUAGE

	NODE NAME	STATES COVERED		
1	CLK1	ONE	ZERO	DRVN
2	CLK2	ONE	ZERO	DRVN
3	EN1	ONE	ZERO	DRVN
4	EN2	ONE	ZERO	DRVN
5	GROUND		ZERO	DRVN
6	MASK		ZERO	DRVN
7	N1	ONE	ZERO	DRVN
8	N10	ONE	ZERO	DRVN
9	N11	ONE	ZERO	DRVN
10	N12	ONE	ZERO	DRVN
11	N13	ONE	ZERO	DRVN
12	N14	ONE	ZERO	DRVN
13	N15	ONE	ZERO	DRVN
14	N16	ONE	ZERO	DRVN
15	N17	ONE	ZERO	DRVN
16	N18	ONE	ZERO	DRVN
17	N19	ONE	ZERO	DRVN
18	N2	ONE	ZERO	DRVN
19	N20	ONE	ZERO	DRVN
20	N21	ONE	ZERO	DRVN
21	N22	ONE	ZERO	DRVN
22	N23	ONE	ZERO	DRVN
23	N24	ONE	ZERO	DRVN
24	N25	ONE	ZERO	DRVN
25	N26	ONE	ZERO	DRVN
26	N27	ONE	ZERO	DRVN
27	N28	ONE	ZERO	DRVN
28	N29	ONE	ZERO	DRVN
29	N3	ONE	ZERO	DRVN
30	N30	ONE	ZERO	DRVN
31	N31	ONE	ZERO	DRVN
32	N32	ONE	ZERO	DRVN
33	N33	ONE	ZERO	DRVN
34	N34	ONE	ZERO	DRVN
35	N4	ONE	ZERO	DRVN
36	N5	ONE	ZERO	DRVN
37	N6	ONE	ZERO	DRVN

38	N7	ONE	ZERO	DRVN
39	N8	ONE	ZERO	DRVN
40	N9	ONE	ZERO	DRVN
41	NN22	ONE	ZERO	DRVN
42	NN23	ONE	ZERO	DRVN
43	NN24	ONE	ZERO	DRVN
44	NN25	ONE	ZERO	DRVN
45	NN5	ONE	ZERO	DRVN
46	NN6	ONE	ZERO	DRVN
47	NN7	ONE	ZERO	DRVN
48	NN8	ONE	ZERO	DRVN
49	POWER		ZERO	DRVN
50	QA1	ONE	ZERO	DRVN
51	QA2	ONE	ZERO	DRVN
52	QB1	ONE	ZERO	DRVN
53	QB2	ONE	ZERO	DRVN
54	QC1	ONE	ZERO	DRVN
1	QC2	ONE	ZERO	DRVN
2	QD1	ONE	ZERO	DRVN
3	QD2	ONE	ZERO	DRVN
4	RST1	ONE	ZERO	DRVN
5	RST2	ONE	ZERO	DRVN

APPENDIX A

The ZyP system is, by design, computer independent. ZyP software commands are intended to work identically on any computer system on which ZyP is installed. However, operating system commands and structures are often different on computers from differing manufacturers. This appendix describes computer specific ZyPSIM commands for the DEC VAX and PRIME computers.

a. VAX Commands

The command line given below is required for layout capacitance estimation and is inserted into ZyPSIM .NET file. For a detailed explanation refer to the .INPUT command section of this document.

```
.INPUT SYS$ZYPROOT:[ZYPSIM.x]LAYOUTCAP.
```

where x is Zy4LIB for the Zy40000 library
or Zy5LIB for the Zy50000 library.

b. PRIME Commands

The command line given below is required for layout capacitance estimation and is inserted into a ZyPSIM .NET file. For a detailed explanation refer to the .INPUT command section of this document.

```
.INPUT ZYP>ZYPSIM>x>LAYOUTCAP
```

where x is Zy4LIB for the Zy40000 library
or Zy5LIB for the Zy50000 library.

5

ZyPSIM REFERENCE MANUAL

Doc: 20-010-209 Rev: C	Issued: September 1984	ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: C
Table of Contents	Rev: C
Pages 1-1 through 1-2	Rev: C
Pages 2-1 through 2-4	Rev: C
Pages 3-1 through 3-1	Rev: C
Pages 4-1 through 4-6	Rev: C
Pages 5-1 through 5-40	Rev: C
Pages 6-1 through 6-21	Rev: C

TABLE OF CONTENTS

5

1 INTRODUCTION

1.1 SCOPE

This manual documents the capabilities and syntax of ZyPSIM, the ZyMOS proprietary logic simulator. However, the ZyPSIM enhancement known as the Test Language is described in the separate document 20-010-309 ZyPSIM Test Language.

1.2 OVERVIEW

ZyPSIM is a general purpose logic simulator which was designed for MOS circuits and the ZyP system in particular. As such, it includes several features which are not generally found in other logic simulators. Major features include:

a. Modeling of the strength of gates.

b. Resistors and capacitors.

c. Fanout delays as a function of capacitive load.

d. Ability to specify the capacitance of each element input.

e. ROM and RAM. The ROM can also serve as a PLA or a truth table.

f. Ability to specify setup and hold times for flip-flops, RAM's, ROM's, or any other structures.

g. Ability to print at fixed intervals, or only when certain signals change state.

h. Two methods of specifying test patterns: as waveforms, or as a table.

i. Ability to check a signal for minimum high and low pulse width.

To run ZyPSIM interactively, type:

ZYPSIM [filename [lib]]

Where filename is the prefix to be used for a family of files. filename.NET is the name of the file containing the circuit description, filename.OUT contains the listing and summary, filename.PRT contains the default simulation output, etc.

Where lib is the library to use. The following are available:
Zy4 = Zy40000 cell library.

Zy5 = Zy50000 cell library.

1.3 RELATED DOCUMENTS

It is assumed that you are conversant with ZyPSIM and the ZyP system in general. Documents which should be considered necessary reading for a full understanding of this document are as follows:

20-010-009	ZyPSIM User's Guide.
20-010-309	ZyPSIM Test Language
20-010-010	ZyP Specification and Test User's Guide.
20-010-210	ZyTEST User's Guide.

1.4 DOCUMENT CONVENTIONS

The following conventions are used in this document to describe command format and syntax.

a. Arguments which you need to provide are shown in lower case, enclosed in angles. Eg:

   .TESTER <tester_type>

   You must supply a value for <tester_type>. The angles should not be included in your response.

b. Optional arguments are also enclosed in square parentheses. Eg:

   .TSTROBE <strobe_time> <node>[,<mask>]

   You may optionally specify argument <mask>. It must be separated from <node> by a "," comma.

c. One or more periods are used to indicate multiple arguments. Eg:

   .TPRINT <node 1> <node 2> ... <node n>

   You must specify values for nodes to be printed, as required.

d. The continuation character (+) may be needed to indicate that a syntax definition is being continued on another line. This does not mean that it is required when the command or element is actually used.

2 TERMS AND CONCEPTS

This section defines the major terms and concepts used by ZyPSIM. ZyPSIM will ignore any characters past column 80 in the netlist.

2.1 NAMES

Names in ZyPSIM may be 23 characters in length, and may contain any of the following characters: A-Z (upper case only), 0-9, *, _, and period. The first character must be one of the letters A-Z.

2.1.1 SIGNAL NAMES

Signal names are user defined names which refer to a single interconnection node.

NOTE: Do not use a period within a signal name unless you wish to refer to a signal inside of a macro. (See section on MACROS for details.)

2.1.2 ELEMENT NAMES

Element names tell ZyPSIM what 'kind' of element is being used. Standard cell names are element names (e.g. IAA1, FAC2, etc.) as are the names of ZyPSIM primitives (e.g. ORAND, WIDTH, UXFR). All these element names are predefined. However, when the element in question is a user defined macro then the element name is the macro name by which that macro will later be called. This macro name may be up to 11 characters long.

2.1.3 ELEMENT ID'S

An element ID is a user defined name which refers to a particular logic gate or circuit element in order to uniquely identify it. For example, if there were two inverters in a circuit with the same element name, the first could have an element ID of INV1, the second, INV2. The element ID is also used to identify macros as they are placed in the circuit. ZyPSIM will use a macro's element ID to construct names for its internal gates and signals. The full element ID may be up to 23 characters long (see the section on MACROS for more details).

2.1.4 SPECIAL NAMES

.ONE and .ZERO are special names reserved for any signal which is to be tied to a logical one or zero, respectively.

In addition, the signal names VSS, VSSn, VDD, VDDn, UNUSED and UNUSEDm where n = 1-9 and m = 1-30 are reserved for power, ground and unused pins.

2.2 SIGNAL STRENGTH

In MOS circuits, gates of different sizes can be wired-ored together (see Section 2.5). Each gate may have a different strength. ZyPSIM has the ability to model gates of different strength, with the result that in a wired-or situation, the strongest signal dominates, over-riding all others. In the ZyPSIM hierarchy, clocks, power and ground are the strongest, followed by normal gates, followed by weak gates (e.g. resistors), followed by gates in the high impedance state.

2.3 SIGNAL LEVEL

As well as strength, a signal may have logic levels 1, 0, or undefined/indeterminate. The following table shows how the different combinations of strength and level are represented.

STRENGTH	ZERO	ONE	INDETERMINATE
FORCING	0	1	X
NORMAL	0	1	X
WEAK	L	H	*
HI-Z	-	+	Z
SPIKE	.	'	~

2.4 SPIKES

Definition: Consider a gate with its output node in a given state (0/L or 1/H). Then the inputs change such that the output would eventually change. Then the inputs change again such that the output would return to its original state. If the time between the two input changes is less than the propagation delay for the output transition initiated by the first change then we have a spike.

The spike will be flagged, but in the simulation the spiking output will not in fact change. If the output was at a 0 or L and would have spiked to a 1 or H level it will be indicated by a period (.) in the spike file (fname.SPI). If the output was at a 1 or H level and would have spiked to a 0 or L level it will be indicated by an apostrophe ('). So the period indicates a 'spiking zero' not a 'spike to zero' and the apostrophe indicates a 'spiking one'. For example, if an inverter has 0-to-1 and 1-to-0 propagation delays of 20 nS and it is given the following input we would get the result shown in Figure 2-1.

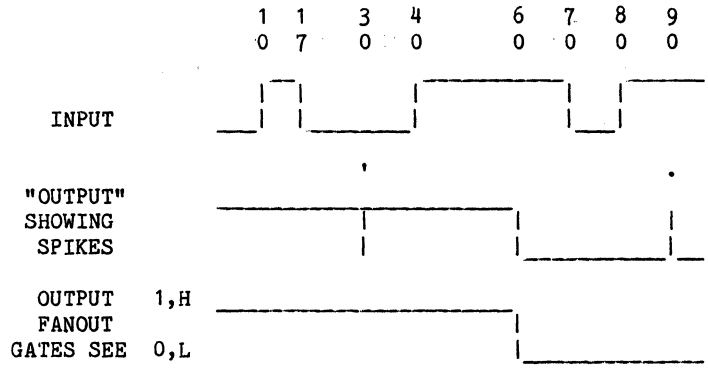


FIGURE 2-1. SPIKE EXAMPLE

2.5 WIRE OR

Standard cells may have their outputs wire-ored simply by using the same name for the outputs that are to be tied together. For example:

```
U54   IAA4  IN=U53         OUT=U54
U55   IAH3  IN=U40 EN=U53  OUT=BUSC
U56   IAH3  IN=U41 EN=U54  OUT=BUSC
```

Here BUSC is the commmon output node.

2.6 TIME

The ZyPSIM time unit is the nanosecond (nS). All element delays, input and output times, etc. are expressed in terms of nanoseconds.

Any input or output timing information specified by the user must be in integers (i.e. integral nanoseconds). Simulation output information will also be reported in integral nS, unless the .RESOLUTION command is used, in which case it will be reported to a tenth of a nanosecond (e.g. 1344.2).

ZyPSIM is an event driven simulator, which means that only those gates whose inputs change are evaluated. Thus, if the inputs to a circuit were stable from time 1000 to 2000, and all the gate outputs became stable at time 1100, then when time 1100 was reached ZyPSIM would immediately jump to time 2000, when the next input changed.

2.7 COMMANDS

Commands are directives to ZyPSIM to do something. This includes everything except the network description. All commands start with a period. Commands are used to print signals, to set the maximum simulation time, to describe the input test pattern, etc.

3 OVERVIEW OF INPUT

All input to ZyPSIM must be upper case except for comments and the title line. Any characters past column 80 in the netlist will be ignored.

3.1 GENERAL FORMAT

The first line read from the input file (filename.NET) will be used as a title. This title will appear at the top of every output page. The last line read from the input file should be the command .END, and signifies the end of data. Everything in between is free-format. There are no fixed columns (except that the comment '*' and the continuation '+' must be in column 1), and there is no required order to the various statements. Three exceptions to this rule are:

a. The .GUARDBAND command must precede all cells.

b. Macros must be defined before they are used.

c. The .GO command executes immediately.

We suggest that the title line start with the comment character ('*') to make explicit the fact that it is not part of the circuit description. It is also a good idea to put the pathname of the .NET file in the title line. All input to ZyPSIM must be upper case except for the title (first line), and comments (either comment lines or in-line comments).

3.2 CONTINUATION

If a gate or command does not fit onto one line, it may be continued on the next line if a plus sign is the first character of the line.

3.3 COMMENTS

A comment line is one whose first character is an asterisk.

An 'in line' comment may be made by using the '%' character followed by the comment. The % must be preceded by at least one character that is not a blank or plus sign (+).

If you want to separate sections of the netlist with some open space use comment lines that contain only the *. Don't use blank lines.

These points are illustrated in the following example.

```
*THIS IS THE TITLE LINE      CA29>DEMO.NET
*
*  THE FOLLOWING IS THE CIRCUIT
*
* THE GAA1 IS A TWO INPUT NAND GATE
N3  GAA1  IN1=N1 IN2=N2 OUT=N3
*
.CLK0 N1 100 R 0
.CLK0 N2 200 R 0
.PRINT N1 N2
+    / N3     %CONTINUATION OF PRINT STATEMENT
* SIMULATE FROM 0 THROUGH 100 nS
.MAXTIME=100
.GO
.END
```

3.4 KEYWORDS

The inputs and outputs of a standard cell may be specified in one of two ways: 1) by using keywords, or 2) by typing the signal names in the correct order. For most cells the most efficient method will be by typing the signal names in the correct order. The order that must be followed is the order of the signals shown in the MODEL FORMAT statement on the last page of the data sheet for the cell. The signal names used in this format statement are also the keywords that must be used if the keyword method is chosen. Keywords are most useful for elements which have different kinds of inputs or outputs, flip-flops for instance. If keywords are used they may follow the cell name in any order.

EXAMPLE: The MODEL FORMAT statement for the FAC1 D flip-flop is

```
{ user i.d. }  FAC1  D  CL  R  Q  NQ
```

therefore the following are exactly equivalent.

```
U1Q    FAC1  D=BUS0 CL=DCLK R=RESET Q=U1Q NQ=U1NQ

U1Q    FAC1  CL=DCLK R=RESET NQ=U1NQ D=BUS0 Q=U1Q

U1Q    FAC1    BUS0    DCLK    RESET   U1Q    U1NQ
```

4 MACROS

4.1 OVERVIEW

SYNTAX:

Macro definition:

```
.MACRO          <macro_name>  [<IO_sig1>...<IO_sigN>]
[.OFF]
[<element_ID>  <element_name>  <element_sig>... ]
.
.
.
[.INFO ...]
.EOM
```

Macro call:

```
<macro_ID>      <macro_name>     <arg1> ... <argN>
```

'Element' can be a standard cell, another macro, or a ZyPSIM primitive.

The number of element signals in the macro definition does not need to equal the number of macro IO signals, but the number of macro IO signals must equal the number of arguments supplied in the macro call.

For a user defined macro, <macro_name> must not begin with either A, P, or Z.

Whenever identical blocks of logic are used in several places in a circuit, the macro facility may be used. By using a macro the logic only needs to be specified once, then it may be placed in the circuit as many times as required, with only the input and output signals changing. The macro must be defined before it is used.

The use of macros does not decrease (or increase) the number of gates in the simulation, or the simulation time required.

The first line of a macro definition must be the command .MACRO followed by the name of the macro and a list of I/O signals. These are the signals which are the connections to the outside world. The names used in the I/O list may be used as keywords, if desired. The signals which appear in the argument list when the macro is called will be substituted for these signals wherever they appear inside the macro (see first example below). Following the .MACRO statement is the definition of the macro. It consists of any number of gates and macro references. The end of the macro is indicated by a .EOM (End of Macro).

Macros may be nested by using macro calls as elements in a macro definition. A macro definition, however, may not be included within the definition of another macro. Every '.MACRO' in the netlist must be followed by a '.EOM' before another '.MACRO' may appear.

EXAMPLE:

```
.MACRO NORLATCH S R Q NQ
* GAB1 IS A NOR GATE AND IAA4 IS AN INVERTER
M1 GAA1  S  INTERNAL1  INTERNAL2
M2 GAA1  R  INTERNAL2  INTERNAL1
M3 IAA4  INTERNAL2     Q
M4 IAA4  INTERNAL1     NQ
.EOM
*
*  PLACE THE NOR LATCH TWICE
*
LATCH1 NORLATCH A B C D
LATCH2 NORLATCH S=E R=F Q=G NQ=H
```

4.2 NESTED ELEMENT ID'S

Every element ID of the netlist must be unique. It is up to the user to make sure that every element ID he defines is different from any other. ZyPSIM will give an error message if this is not the case. However, if a macro is called more than once the same elements get used each time. This is not a problem because ZyPSIM generates unique ID's by adding a prefix to the element ID's that is a concatenation of of the macro ID's down through the nesting levels. In the NORLATCH example above the macro is called twice, so there will be two sets of the elements M1,M2,M3 and M4. The first call of NORLATCH (LATCH1) will expand to the following set of gates:

```
LATCH1.M1 GAB1  A  LATCH1.INTERNAL1  LATCH1.INTERNAL2
LATCH1.M2 GAB1  B  LATCH1.INTERNAL2  LATCH1.INTERNAL1
LATCH1.M3 IAA4  LATCH1.INTERNAL1   C
LATCH1.M4 IAA4  LATCH1.INTERNAL2   D
```

and the second call will expand to:

```
LATCH2.M1 GAB1  E  LATCH2.INTERNAL1  LATCH2.INTERNAL2
LATCH2.M2 GAB1  F  LATCH2.INTERNAL2  LATCH2.INTERNAL1
LATCH2.M3 IAA4  LATCH2.INTERNAL1   G
LATCH2.M4 IAA4  LATCH2.INTERNAL2   H
```

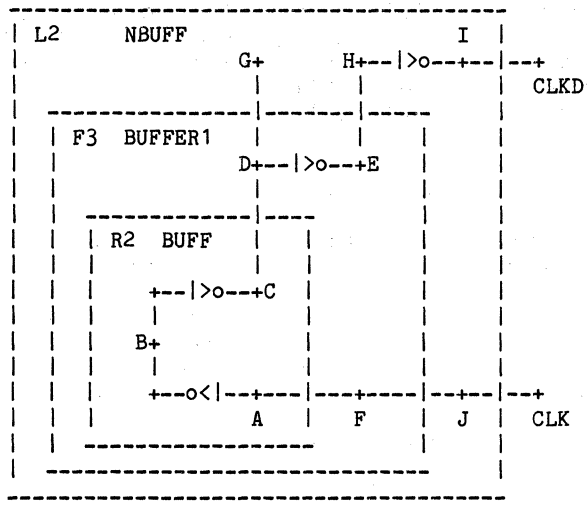
ZyPSIM uses periods (.) to separate the macro ID's of the different levels. If an element with an ID of U2 were in a macro that was called with a macro ID of FF1, which was in turn called by a macro with a macro ID of CTR3, which was called by a macro with a macro ID of M9, then that element would be given the unique element ID of M9.CTR3.FF1.U2 by ZyPSIM. These "expanded" element ID's can be up to 23 characters long (including periods). M9.CTR3.FF1.U2 is 14 characters.

4.3 MACRO INTERNAL SIGNAL NAMES

A macro internal signal is one that does not appear in the macro IO list. No external signal name is passed to it when the macro is called. In order to assure unique names for these signals when macros are called ZyPSIM uses prefixes in the same way as for element ID's. Each time a macro is called, the caller's element ID is added to the end of the current prefix, and the prefix is placed in front of the original signal name. An external signal of a macro will become an internal signal if the macro is nested in another and the signal is not brought out to that macro's IO list. The following netlist example and schematic illustrate how naming is handled.

```
.MACRO  BUFF   A  C
IN1   IAA4   A  B
IN2   IAA4   B  C
.EOM
*
.MACRO  BUFFER1  F  D  E
R2    BUFF   F  D
IN1   IAA4   D  E
.EOM
*
.MACRO  NBUFF  J  I
F3    BUFFER1  J  G  H
IN1   IAA4   H  I
.EOM
*
L2    NBUFF  CLK  CLKD
*
```

In the following schematic of this netlist signals that cross a macro's "boundary" are signals that are in the IO list.




```
A,F,J,CLK = CLK
   I,CLKD = CLKD
    C,D,G = L2.G
      E,H = L2.H
        B = L2.F3.R2.B
```

Because of the way that signal names are passed down through macro IO lists the "signals" A, F and J do not exist in the netlist--they have been replaced by CLK. In order to look at node "C" the signal L2.H must be printed.

The expanded signal name may be 23 characters long. This includes periods so L2.F3.R2.B is 10 characters.

4.4 CMACROS

Sometimes it's important that logic elements be placed physically adjacent to one another on the finished chip in order that critical timing paths, etc., perform as expected. ZyMOS allows the designer to constrain placement in this way through the use of Cluster Macros (CMACRO's).

SYNTAX:

Macro definition:

```
.CMACRO          <macro_name>   [<IO_sig1>...<IO_sigN>]
[<element_ID1>  <element_name1>  <element_sig1>... ]
.                                          .
.                                    .
.                                  .
[<element_IDJ>  <element_nameJ>  ... <element_sigM>]
.EOM
```

where M $\geq$ N and M - N = number of cell outputs not connected to anything either inside or outside the macro.

Macro call:

```
<macro_ID>      <macro_name>     <arg1> ... <argN>
```

'Element' must be a stand alone digital cell. It may not be an analog or telescoping digital cell, or another macro.

Every element input or output must have a unique signal name within the macro. The signal name of every input or output that connects to any other input or output either inside or outside of the macro must appear in the macro IO list. The number of macro IO signals must equal the number of arguments supplied in the macro call.

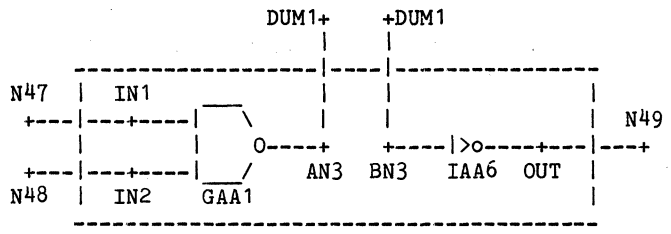
For a user defined macro, <macro_name> must not begin with either A, P, or Z.

The biggest difference between the syntax of a "regular" macro and a CMACRO is that every input and output of the elements in the macro must appear as a uniquely named signal in the macro IO list if it is to be connected to anything else. Any connections between these inputs and outputs must be made when the macro is called.

For example if an AND were to be created from a GAA1 NAND and an IAA6 Inverter a regular macro could be used as follows:

```
.MACRO  ANDBUFF  IN1  IN2  OUT
N3  GAA1  IN1  IN2  N3
OUT IAA6  N3   OUT
.EOM
*
N49  ANDBUFF  N47  N48  N49
```

But if a CMACRO is needed it must be set up as:




```
.CMACRO  ANDBUFF  IN1  IN2  OUT
+                 AN3  BN31
N3  GAA1  IN1  IN2  AN3
OUT IAA6  BN3  OUT
.EOM
*
N49  ANDBUFF  N47  N48  N49
+             DUM1 DUM1
```

The connection between AN3 and BN3 is made when the macro is called by assigning the same "dummy" signal name to it.

The cell order in the CMACRO (from .CMACRO to .EOM) is the order from left to right in which the cells will be placed on the row of the actual chip.

NOTE: A CMACRO may not call a regular macro in its definition, but a regular macro may call a CMACRO. Thus a CMACRO can be made to look like a regular macro by using a "dummy" call to redefine it. For example, we could make ANDBUFF above as:

```
.CMACRO  CMABUFF  IN1  IN2  OUT
+                 AN3  BN31
N3  GAA1  IN1  IN2  AN3
OUT IAA6  BN3  OUT
.EOM
*
.MACRO   ANDBUFF  IN1  IN2  OUT
OUT  CMABUFF  IN1  IN2  OUT
+             DUM1 DUM1
.EOM
*
N49  ANDBUFF  N47  N48  N49
```

By doing this the dummy nodes are connected once, not every time the macro is called. An extra level of macro nesting is used, however.

5 COMMANDS

5.1 .ACE

The .ACE command (Artwork Capacitance Estimation) allows the capacitance of each signal node to be estimated as a function of the chip size and the number of gates to which the signal is connected.

SYNTAX:

.ACE <N> <f1 k1> ... <fN kN> <c1>

0 < N ≤ 50
fN = integer > 0
f2 > f1, ... fN > f(N-1)
k's, c1 may be negative

There are N pairs of f and k. Each f is a fanout number and k is its corresponding constant. C1 is a global normalizing constant. Let n be an integer from 1 through N-1. Then the estimated connection capacitance in picofarads for a given node is:

for fn ≤ fanout number < f(n+1)

est = kn * c1 * sqrt(#grids in full IC)

for fanout number < f1 then kn = 0 (i.e. est=0)

for fanout number ≥ fN then kn = kN

The fanout number for the .ACE command is the number of gates that an output drives not the number of inputs. If an output goes to three inputs of a single gate and nowhere else then its fanout is one, not three. This is because .ACE is used to get gate-to-gate interconnect capacitance estimates and it is assumed that all the inputs of a gate are close enough together that the routing capacitance from another gate is the same whether just one or more than one of the inputs are driven. The additional input capacitance will be accounted for, but not by the .ACE command.

The estimates can be seen by looking at the filename.XRE file (generated if the netlist contains the .XREF command).

The .ACE command will not generate an estimate for the output node of a primitive element. If a standard cell drives a primitive that primitive will count as a fanout gate of the cell.

EXAMPLE:

```
.ACE 3    2 .05    5 .08    12 .11    .125
G0 IAA1 CLK S1
G1 IAA1 S1  OUT1
G2 IAA1 S1  OUT2
G3 GAA1 S1  OUT1 OUT3
```

NOTES:

a. The values of the variables of the syntax statement can be seen as:

```
                N   f1  k1   f2  k2   f3  k3    c1
          .ACE 3    2 .05    5 .08   12 .11   .125
```

The fanout of S1 is 3, so since f1 $\leq$ 3 < f2 then "n" is 1 in the equation above and kn=k1=.05. Therefore if the total number of grids were 10000 the estimate for S1 would be .05*.125*sqrt(10000) = .625 = .62 pF when truncated to two decimal places.

b. Since signal OUT1 drives only one gate its fanout number is 1 which is less than f1, so the estimate for OUT1 is 0.

5.2 .CLK

The .CLK command (Clock) is one of the ways to generate input patterns for a simulation.

SYNTAX:

```
        0  L
.CLK<1>[W]  <sig_name>  [<t1> <t2>...<tN> [R <tr>]]
        H
```

Where t1 > 0
t2 > t1, ... tN > t(N-1)
tN > tr ≥ 0

If R is specified the signal will repeat the toggle pattern from tr through tN from tN on.

Number of .CLK statements per netlist ≤ 99

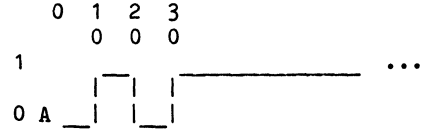
The two basic forms of the .CLK command, .CLK0 and .CLK1, specify that the starting state of the signal is 0 or 1 respectively. The normal suffixes for primitive gates may be appended to specify input signals of lower strengths. Following the .CLK command is the name of the signal and its toggle times. Repeating patterns may be specified by an 'R' followed by the time from which the toggle pattern is to be repeated.

EXAMPLES:

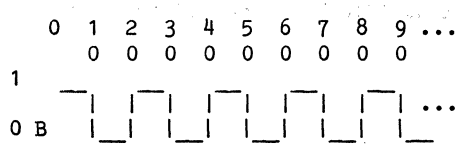
```
.CLK0  A  10 20 30

.CLK1  B  10 R 0

.CLK1W C  15 35 45 R 20
```

NOTES:

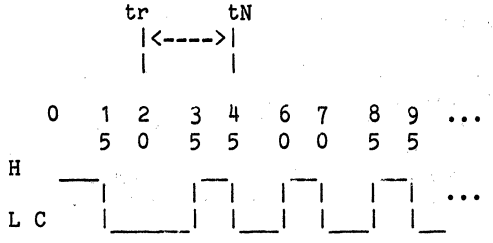
a. Signal 'A' starts off in a zero state, switches to a 1 at 10 nS, back to 0 at 20 nS and back to 1 at 30 nS. It will remain at a 1 level for the remainder of the simulation.



b. Signal 'B' starts off in the 1 state and toggles every 10 nS.



c. Signal 'C' is a weak signal. It starts at a weak 1 logic state (H), goes to a weak 0 logic state (L) at 15 nS, goes to H at 35 and L at 45. The last time before the R is 45, so tN=45, while tr=20. The toggle pattern from tr through tN is the pattern from 20nS through 45nS and is 'toggle after 15nS, toggle again after another 10nS'. This pattern is repeated from 45ns on to give the following pattern.



5.3 .CMACRO

The .CMACRO command (Cluster Macro) is used when logic elements must be placed physically adjacent to one another on the finished chip in order that critical timing paths, etc., perform as expected.

SYNTAX:

Macro definition:

```
.CMACRO          <macro_name>   [<IO_sig1>...<IO_sigN>]
[<element_ID1>  <element_name1>  <element_sig1>... ]
  .                                      .
  .                                   .
  .                                .
[<element_IDJ>  <element_nameJ>  ... <element_sigM>]
.EOM
```

Where M > N and M - N = number of cell outputs not connected to anything either inside or outside the macro.

Macro call:

```
<macro_ID>      <macro_name>     <arg1> ... <argN>
```

For a user defined macro, <macro_name> must not begin with either A, P. or Z.

See the previous section on CMACRO's for more details.

5.4 .DELTAT

The .DELTAT command (Delta t) is one of the ways to specify at what time points the values of the signals in the .PRINT statement(s) will be printed. See also the .POC command.

SYNTAX:

.DELTAT [=] <t>

Where t > 0

The number t specifies the nanosecond increment between print times. For example, with a .DELTAT=5, the output would be printed at times: 0, 5, 10, etc.

If there is more than one .DELTAT command in the netlist only the last one that occurs will be in effect.

A .DELTAT command and a .POC command are not both allowed in the same netlist.

5.5 .END

The .END command is the last statement in the simulation file seen by ZyPSIM. It should be the last statement in the netlist and if any lines do follow it they are ignored.

SYNTAX:

.END

Arguments are not required or permitted.

The .GO command should immediately precede the .END command. Therefore the pair

```
.GO
.END
```

should be the last two lines of every netlist.

5.6 .GO

The .GO command tells ZyPSIM to start running the simulation. Any lines that come between the .GO command and the .END command are ignored.

SYNTAX:

.GO

Arguments are not required or permitted.

The .GO command should immediately precede the .END command which should be the last line of the netlist. Therefore the pair

```
.GO
.END
```

should be the last two lines of every netlist.

5.7 .GUARDBAND

The .GUARDBAND command is used to allow simulations at extremes of temperature, supply voltage and processing parameters.

SYNTAX:

.GUARDBAND VDD=<Vmin,Vmax> TEMP=<Tmin,Tmax [CONDITION= <c>]

Where: $1.5 \leq$ Vmin,Vmax ≤ 6.0 for the Zy40000 library
$2.0 \leq$ Vmin,Vmax ≤ 6.0 for the Zy50000 library
$-55 \leq$ Tmin,Tmax ≤ 125
c = FAST, SLOW or NOM

Vmin, Vmax, Tmin and Tmax must all be specified. They must also appear in min, max order within their pair, i.e. V/Tmin $\leq$ V/Tmax (note: $-55 \leq -45 \leq 40$).

The CONDITION keyword determines what temperature, voltage and process data will be used for the simulation according to this table:

CONDITION	VDD	TEMP	PROCESS	.TPRINT OUTPUT EXTENSION
FAST	Vmax	Tmin	FAST	.TPF
SLOW	Vmin	Tmax	SLOW	.TPS
NOM	5.0	27	SLOW	.TPN

The keywords TEMP, VDD and CONDITION must be used for this command, though they may appear in any order and CONDITION is optional.

Only one .GUARDBAND command is allowed per netlist. If no .GUARDBAND command is included, or if the CONDITION keyword is not used, the simulation will run under NOM conditions, i.e. VDD=5.0, TEMP=27 and PROCESS=SLOW.

EXAMPLES:

```
.GUARDBAND  TEMP=0,70  VDD=4.75,5.25  CONDITION=SLOW

.GUARDBAND  VDD=4.75,5.25  TEMP=0,70  CONDITION=FAST

.GUARDBAND  TEMP=0,70  VDD=4.0,5.5
```

NOTES:

a. The first example would simulate performance at 70 degrees C with a positive supply voltage of 4.75 volts and SLOW process parameters. If the netlist contained a .TPRINT command its output file would be filename.TPS.

b. The second example would simulate performance at 0 degrees C with a positive supply voltage of 5.25 volts and FAST process parameters. If the netlist contained a .TPRINT command its output file would be filename.TPF.

c. The third example would simulate performance at 27 degrees C with a positive supply voltage of 5.0 volts and SLOW process parameters. If the netlist contained a .TPRINT command its output file would be filename.TPN.

5.8 .INFO

The .INFO command (Information) is used to give ZyPSIM grid count, transistor count and cell type information for standard cells.

SYNTAX:

```
.INFO  [<grids> [<trans> [<power> <type>]]]
```

Where: grids = number of grids = integer $\geq$ 0
trans = number of transistors = integer $\geq$ 0
power = reserved variable, must be 0
type = 0, 1, 2, 3, 4, 5
0: digital stand alone cell
1: digital control telescoping cell
2: digital body telescoping cell
3: analog left end cap cell
4: analog cell, not end cap
5: analog right end cap cell

Every time ZyPSIM sees a .INFO statement it adds the <grids> and <trans> values to its present grid and transistor totals. If a .INFO command appears within a macro definition then this happens each time the macro is called. Thus the .INFO commands within the standard cell primitive macros allow each cell to accurately increment these totals as it is used in the netlist. As long as the total number of grids or transistors used in a netlist is less than 10,000,000 the values printed in the .OUT file are correct.

EXAMPLES:

```
.MACRO  RX7  IN  OUT
OUT  INV  IN=IN OUT=OUT
.INFO 42 5 0 4
.EOM

.INFO 10 2

.INFO 50
.INFO 29 16
```

NOTES:

a. Whenever the macro RX7 is called 42 grids and 5 transistors will be added to those totals, and ZyPSIM will know that an analog cell was used in the netlist.

b. When ZyPSIM sees the second example 10 grids and 2 transistors will be added to the grids/transistor totals.

c. The pair of .INFO commands in the third example will add 79 grids and 16 transistors to the netlist totals.

If only a small portion of a circuit is being simulated, interconnect capacitance estimates (using .INPUT ...LAYOUTCAP) can be more accurate if a .INFO command is used to add enough grids to make the total equal the full chip estimated grid count. For example if 5000 grids of logic were being simulated out of an estimated 30000 grid chip, then the command .INFO 25000 would be appropriate. It should be removed, of course, from the final netlist.

5.9 .INPUT

The .INPUT command is used to insert the contents of a file into the netlist.

SYNTAX:

.INPUT [=] <pathname_of_file>

You must have read rights to the file in order for it to be input.

A .INPUT command may not appear within a ZyPSIM command or within a ZyPSIM primitive. For example, a .INPUT command may be used to put a .TABLE command into the netlist as long as the whole .TABLE command (including .TABLE sig_name1 sig_name2 ... etc.) is contained in the file that will be input. So if the file TABLEINFO contained

```
.TABLE RES CLK BUS1 BUS2
0      1 0 0 0
100    0 0 0 0
200    0 1 0 0
300    0 0 1 1
400    0 1 1 1
```

then it would be OK to put the command

.INPUT TABLEINFO

in the netlist. However, if TABLEINFO only contained

```
0      1 0 0 0
100    0 0 0 0
200    0 1 0 0
300    0 0 1 1
400    0 1 1 1
```

then it would be illegal to use

```
.TABLE RES CLK BUS1 BUS2
.INPUT TABLEINFO
```

to try to do the same thing. This is because the .INPUT command would be within the .TABLE command since the .TABLE command can't end without at least the time=0 line of signal values.

The .INPUT command may be nested to any level, i.e. the file that is to be input may itself contain a .INPUT command, etc.

EXAMPLES:

.INPUT=CA18>Z50589A>LOG>COUNTER

.INPUT DECODER

5.9.1 LAYOUTCAP

In order to allow interconnect capacitance estimates to be included in a ZyPSIM simulation a special .INPUT command is used. The exact computer specific command is shown in the Appendix.

This command will put the appropriate LAYOUTCAP file into the circuit. This file contains a special .ACE command whose parameters have been determined by ZyMOS on the basis of real interconnect capacitance values from all ZyMOS' previous circuits as a function of grid count, fanout and process technology (cell library used).

This .INPUT command may appear anywhere in the netlist, though it is usually near the beginning.

5.10 .LIBRARY

The .LIBRARY command allows a private cell library to be specified which will be searched for cell information before going to the global library (Zy40000 or Zy50000) chosen for the simulation.

SYNTAX:

.LIBRARY= <UFD_pathname>

You must have read rights to the specified library.

Every usage of a cell in the netlist is a macro call to a macro with the same name as the cell. When a cell is used ZyPSIM first searches the netlist (including any files brought in by .INPUT commands) for a macro whose name is the same as the cell's. If one is not found ZyPSIM then searches the global library chosen for a file with the same name as the cell and expects to find a macro with the same name as the cell inside that file (which it then uses to simulate the performance of the cell). If a .LIBRARY command is used, ZyPSIM will still search the netlist first, but if the cell information is not found then the UFD specified in the .LIBRARY command will be searched next. The global library will be searched last.

For example, if a cell named IAF4 was requested, and the following .LIBRARY command was active:

```
.LIBRARY=MYNAME>CELLS        %PRIME pathname example
```

ZyPSIM would look for a file by the name of MYNAME>CELLS>IAF4 to get the .MACRO definition of the cell.

It is important that the .LIBRARY command appear before the cells in its library are invoked. Also, every time a new .LIBRARY command is read, the name of the library to be searched will be changed. Therefore a .LIBRARY command is in effect for the cells between it and the next .LIBRARY command in the netlist.

5.11 .MACRO

The .MACRO command is used to define a macro before it is used.

SYNTAX:

Macro definition:

```
.MACRO          <macro_name>  [<IO_sig1>...<IO_sigN>]
[.OFF]
[<element_ID>  <element_name>  <element_sig>... ]
.
.
.
[.INFO ...]
.EOM
```

Macro call:

```
<macro_ID>      <macro_name>     <arg1> ... <argN>
```

'Element' can be a standard cell, another macro, or a ZyPSIM primitive.

The number of element signals in the macro definition does not need to equal the number of macro IO signals, but the number of macro IO signals must equal the number of arguments supplied in the macro call.

For a user defined macro, <macro_name> must not begin with either A, P, or Z. See the previous section on MACRO's for more details.

5.12 .MAXTIME

The .MAXTIME command (Maximum Time) specifies the time at which the simulation is to end.

SYNTAX:

.MAXTIME [=] [<maxtime>]

Where: $0 \leq$ maxtime $\leq$ 999999999 (i.e. < 1 Sec)

But if the .RESOLUTION command is used, then
$0 \leq$ maxtime $\leq$ 999999999 (i.e. <0.1 Sec)

If <maxtime> is not specified, or if no .MAXTIME command is used at all then maxtime = 0. This can be useful, since for maxtime = 0 ZyPSIM will compile the network and perform all its netlist syntax checks, as well as generate a .OUT file with grid count totals, etc.

Multiple .MAXTIME commands are allowed, but only the last one in the netlist is in effect.

EXAMPLES:

.MAXTIME 432600

.MAXTIME=9600

.MAXTIME = 0

In the first example the circuit will be simulated, using whatever input patterns have been specified, from time=0 to time=432600 nS. The times refer to circuit performance within the simulation and not to CPU time of the computer that ZyPSIM is running on.

5.13 .NORUN

The .NORUN command (No Runtime errors) is used to turn off error messages that result from setup, hold, width or slope violations.

SYNTAX:

.NORUN

Arguments are not required or permitted.

5.14 .NOSPIKE

The .NOSPIKE command (No Spike) is used to suppress spike reporting for specific nodes/signals or for the whole circuit.

SYNTAX:

.NOSPIKE [<sig1> ... <sigN>]

Any spikes on the signals sig1 through sigN above will not appear in the filename.SPI file and they will not be reported in the output file from a .PRINT command.

If no signals follow .NOSPIKE (i.e. a "bare" .NOSPIKE is used) then none of the spikes in the circuit will be reported. A filename.SPI file will still be produced, but it will not contain anything.

If there is more than one .NOSPIKE command in the netlist this is equivalent to one .NOSPIKE followed by the signals from all the .NOSPIKE commands.

EXAMPLES:

```
.NOSPIKE          \
.NOSPIKE  A B      |
.NOSPIKE  CLK D    | = .NOSPIKE A B CLK D E
.NOSPIKE  E B     /
```

Therefore in order to suppress reporting of spikes for all the signals of a circuit there cannot be any signal names in the signal lists of any of the .NOSPIKE commands (i.e. nothing but bare .NOSPIKE commands).

EXAMPLES:

```
.NOSPIKE ALU1 CARRYOUT ALLZERO

.NOSPIKE

.NOSPIKE DECODE1.N1 DECODE1.N3
```

NOTE:

In the third example, spikes on the internal nodes N1 and N2 of a macro called with a macro ID of DECODE1 will not be reported.

5.15 .OFF

The .OFF command is used to prevent the inclusion of macro internal signals in the toggle (filename.TOG) file.

SYNTAX:

.OFF

Arguments are not required or permitted.

The .OFF command is designed to prevent the internal signals of the primitive macros used to model standard cells from being included in the .TOG file. The .OFF command should precede any element usages (gates, macro calls, etc.) in the macro which it is in, and it acts only on the level of the macro that it's in. It is 'reset' by the macro's .EOM command. When the .TOGGLE FULL command is the last .TOGGLE command in the netlist it overrides all .OFF commands.

EXAMPLES:

```
.MACRO THX1138 IN OUT
N2  INV  IN=IN  OUT=N2
OUT INV  IN=N2  OUT=OUT
.EOM
.MACRO HAL IN OUT2
* THIS MACRO CONTAINS .OFF
.OFF
M1  THX1138 IN  A
OUT2 INV  IN=A  OUT=OUT2
.EOM
```

In this example the .OFF command in macro HAL would prevent its internal signal A from appearing in the .TOG file. However, it would not prevent the internal signal N2 of macro THX1138 (called by HAL) from appearing, since N2 is a level lower. If THX1138's N2 were to be suppressed, then THX1138 must have its own .OFF command (and the FULL argument must not be used with the .TOGGLE command).

Do not use a .OFF command outside of a macro, or after any element within a macro.

5.16 .PADS

The .PADS command is used to tell the artwork generation software which signals are pads, and their respective pad numbers.

SYNTAX:

.PADS <sig1>:1 <sig2>:2 ... <sigN>:<N>

N = number of pins on package selected for chip.
<sig1> through <sigN> must be $\leq$ 8 characters long.
<sig1> through <sigN> may be any of the reserved signals VSS, VDD, VSSn, VDDn, UNUSED and UNUSEDm, where n = 1-9 and m = 1-30.
<sig1> through <sigN> must all be specified (no gaps) and they must be in strict numerical order.

EXAMPLES:

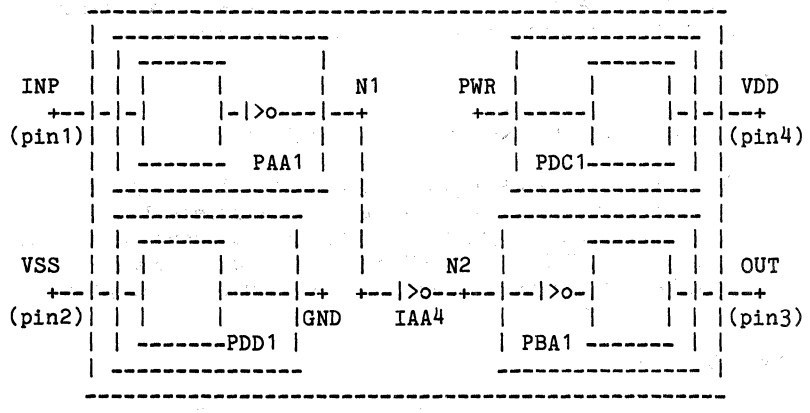
```
.PADS  ABUS0:1 ABUS1:2 VSS:3 ABUS2:4 ABUS3:5 BBUS0:6
+      BBUS1:7 BBUS2:8 BBUS3:9 VDD:10 SEL:11 Q1:12
+      Q2:13   Q3:14

.PADS  ABUS0:1 ABUS1:2 VSS:3 ABUS2:4 UNUSED1:5 ABUS3:6
+      BBUS0:7 UNUSED2:8 BBUS1:9 BBUS2:10 BBUS3:11 VDD:12
+      SEL:13  Q1:14 Q2:15 Q3:16
```

NOTES:

a. In the first example the chip is being put into a 14 pin package. Note that VDD and VSS are included.

b. In the second example the same chip is being put into a 16 pin package, so UNUSED1 and UNUSED2 are used to indicate which package pins will not be connected to the chip.

The signal names that are required in the .PADS command are the signals that go into or come out of the chip from the outside world. For example, in the following four pad circuit the .PADS statement would be as shown in the netlist portion provided.




```
* FOUR PAD CIRCUIT
.PADS   INP:1 VSS:2 OUT:3 VDD:4
N1   PAA1  INP  N1
GND  PDD1  VSS  GND
OUT  PBA1  N2   OUT
PWR  PDC1  VDD  PWR
N2   IAA4  N1   N2
```

5.17 .POC

The .POC command (Print On Change) is one of the ways to specify at what time points the values of the signals in the .PRINT statement(s) will be printed. See also the .DELTAT command.

SYNTAX:

```
.POC  [<sig1>  ... <sigN>]
```

The .POC command tells ZyPSIM to print the signals in the .PRINT commands when any of the signals in the .POC command changes. A 'change' is when a signal goes from one of the values 0 L - + H 1 X * Z to any of the others. It therefore may give more information than a .PRINT statement associated with a .DELTAT command which prints at fixed intervals. The .POC command will automatically suppress printouts for times when the signals in its signal list don't change. If no signals are specified, the signals in the .PRINT commands will be used instead.

EXAMPLES:

```
.PRINT A B C / D E
.POC   A D

.PRINT A B C / D E
.POC  A B C D E

.PRINT A B C / D E
.POC
```

NOTES:

a. The first example will print the signals A, B, C, D, and E whenever signals A or D change state.

b. The second and third examples are equivalent, printing the output whenever any signal in the .PRINT statement changes.

c. A line will be printed in the output print file whenever a spike occurs but this is independent of the .POC command (see section on SPIKES).

A .POC and a .DELTAT command are not both allowed in the same netlist.

5.17.1 Multiple .POC Commands

If there is more than one .POC command in the netlist this is equivalent to one .POC followed by the signals from all the .POC commands.

EXAMPLES:

```
.POC          \
.POC A B       |
.POC CLK D     | = .POC A B CLK D E
.POC E B      /
```

Therefore in order for the print-on-change signals to default to the ones in the .PRINT statements there cannot be any signal names in the signal lists of any of the .POC commands (i.e. nothing but 'bare' .POC commands).

ANOTHER EXAMPLE:

```
.PRINT, CNT3.TEST  Q32 Q31 Q30
.PRINT, CNT4.TEST  Q42 Q41 Q40
.POC
```

This will print the values of Q32, Q31 and Q30 to the output file CNT3.TEST and the values of Q42, Q41 and Q40 to the output file CNT4.TEST whenever any of those signals changes. If the statement '.POC STROBE' were added to the netlist then the values would be printed only when STROBE changed. In either case the TIME column of the output file would be the same for both CNT3.TEST and CNT4.TEST.

5.18 .PRINT

The .PRINT command specifies a list of signals which are to be printed to the output file. These signals may be printed as binary numbers, or several may be grouped together and printed as hex or octal numbers. The "binary" signals will be printed such that the the logic levels of low, high and indeterminate will be indicated--technically a trinary system, but we will still call these binary signals. The .PRINT command is used in conjunction with the .DELTAT or .POC commands.

A spike will be noted on the printout with an 'S' in the left margin at the time that the spike occurs.

The values in the printout will indicate the strength of the signal as well as its logic level, according to the following table:

STRENGTH	ZERO	ONE	INDETERMINATE
FORCING	0	1	X
NORMAL	0	1	X
WEAK	L	H	*
HI-Z	-	+	Z
SPIKE	.	'	~

5.18.1 SIMPLE .PRINT COMMAND

SYNTAX:

.PRINT [,<filename[.<ext>]>] [<sig1>...[/]...<sigN>]

If <filename.ext>	is specified the output file will be FILENAME.EXT. No .PRINT information will appear at the terminal.
If only <filename>	is specified the output file will be FILENAME.PRT. No .PRINT information will appear at the terminal.
If [,<filename[.ext]>]	is not specified the output file will default to X.PRT where X.NET is the name of the netlist file. The .PRINT information will also appear at the terminal.
If <filename>	is CRT and there is no extension the output will appear only at the terminal and no output file will be created for the signals of that .PRINT statement. The last .PRINT,CRT statement in the netlist has priority for any output to the terminal.

Number of signals plus /'s $\leq$ 119

Number of .PRINT commands per netlist $\leq$ 7

The simplest format of the .PRINT command is the one which specifies only binary data to be printed. The slash (/) is used to specify where blank columns are to appear.

EXAMPLE:

```
.PRINT A B C / D E
```

The above statement would cause the following printout to go to both the CRT and a file by the name of filename.PRT, if filename.NET is the name of the netlist.

```
TIME   ABC DE

   0   110 11
   5   111 10
  10   011 11
   .
   .
   .
```

5.18.2 HEX AND OCTAL FIELDS

To specify hex or octal fields, the keyword .HEX or .OCT must be given within the signal list portion of the .PRINT statement. Following the .HEX or .OCT is the name to be used in the header, enclosed in square brackets. Following the right bracket must be an equal sign and a list of signals which are to be used to construct the hex or octal number from the most significant bit to the least significant bit. These signal names must be separated from each other by commas.

EXAMPLE:

```
.PRINT BUS0 BUS1 BUS2 BUS3 BUS4/
+       .HEX[BUS]=BUS0,BUS1,BUS2, BUS3,
+       BUS4 / .OCT[OBUS]=BUS0,BUS1,BUS2,BUS3,BUS4
```

This would produce the following printout:

```
          BBBBB B  0
          UUUUU U  B
          SSSSS S  U
     TIME 01234    S


        0 00000 00 00
        5 11010 1B 32
       10 10001 11 21
       15 100X0 1X 2X
       20 X*001 XX X1
        .
        .
        .
```

NOTE:

If any of the signals comprising a hex or octal digit goes to an indeterminate state (X,*,Z), that digit will be printed as indeterminate (X).

Normally the column headings of the .PRT file are the same as the signal name being printed. By using HEX or OCTAL files in the .PRINT command, the heading labels can be arbitrarily specified and if the list of signals used to construct the hex or octal number contains only one signal name then the printed data will be binary. This technique may be used to format the .PRT output.

5.18.3 MULTIPLE .PRINT COMMANDS

Multiple print statements are allowed as long as the output of each .PRINT statement is directed to a separate file, or to the CRT as mentioned above. As many as seven .PRINT statements may be used. For example, if the netlist were named CPU.NET then the following .PRINT statements

```
.PRINT               CLK SIG1/ SIG2  ...
.PRINT ,CPU2         SIG3  CIN COUT  ...
.PRINT, SIG9.TEST    SIG9
```

would produce the output files CPU.PRT, CPU2.PRT and SIG9.TEST. Only the CPU.PRT information would appear at the terminal.

5.19 .RESOLUTION

The .RESOLUTION command is used to allow greater simulation accuracy by directing ZyPSIM to round propagation delays, etc., to the nearest tenth nanosecond rather than the nearest whole nanosecond.

SYNTAX:

.RESOLUTION

Arguments are not required or permitted.

The .RESOLUTION command must appear prior to any cell usage or macro call (between the title and .GO, of course). If the .RESOLUTION command is used ZyPSIM will round propagation delays, setup and hold times, slope times, etc., to the nearest tenth nanosecond. The output results from .PRINT commands will also be reported to a tenth nanosecond (e.g. 26743.9 as an output time point).

If the .RESOLUTION command is not used, the default condition will result in a simulation whose times are rounded to the nearest whole nanosecond.

5.20 .TABLE

The .TABLE command is one of the ways to generate input patterns for a simulation.

SYNTAX:

```
.TABLE  [<sig1>       ... <sigM>]
0        <sig1_init> ... <sigM_init>
[<t1>    <sig1_val1> ... <sigM_val1>
  .
  .
  .
 <tN>    <sig1_valN> ... <sigM_valN>
[R <tr>]]
```

Where: $t1 > 0$
$t2 > t1, \ldots tN > t(N-1)$
$tN > tr \geq 0$
$M \leq 78$ (absolute max., see below)

sig_init, sig_val may be: 0 1 X L H * - + Z

The spike symbols . and ' are not allowed

If R is specified the signals will repeat their pattern from tr through tN from tN on.

Number of .TABLE statements per netlist ≤ 99

The .TABLE command is followed by a list of signals. These signals may be continued on additional lines as needed. On the lines following the list of .TABLE signals is a table of values to be used. The format is the same as the output from a .PRINT statement. For these 'data' lines no continuation characters (+) are required or allowed at the beginning of the lines--this is to allow the output of one simulation to be used directly as the input to another simulation. However, the '+' may be used as a signal value in the table since it is also the standard print symbol for a Hi-Z logic level 1 signal.

NOTE:

If the .PRINT output is from a simulation that used the .RESOLUTION command, then it will not be compatible with .TABLE format because a .TABLE can only accept integral nanosecond as time points.

The number of signals that may be included in one .TABLE is a function of how many data columns can be fit into 80-character lines (ZyPSIM ignores everything past column 80 in the netlist). If tN is 9nS or less and there are no spaces between the data columns then 78 data columns are possible (80 minus one for the time value and one for the space between the time and the first signal value) so 78 signals can be specified. A more practical number would be 60 to 70 signals.

EXAMPLE:

```
.TABLE A B C D        would be        .TABLE A B C D
   0    00 10                           0    00 10
  10    00 10        equivalent        14    *0 Z1
  14    *0 Z1                          16    1L X1   ___
  16    1L X1          to:             23    H+ 10   repeat
  23    H+ 10                          29    -0 *1   pattern
  29    -0 *1                          34    01 01   ___
  34    01 01                          41    H+ 10
  R 16                                 47    -0 *1
                                       52    01 01
                                              .
                                              .
                                              .
```

NOTE: The same signal D would be generated by the .CLK statement

```
.CLK0  D  14  23  29  41  R  23
```

5.21 .TESTER

The .TESTER command defines the type of tester to be used for device testing.

SYNTAX:

.TESTER <tester_type>

Valid values for <tester_type> are:

SENTRY7
SENTRY20

EXAMPLE:

.TESTER SENTRY20

The default when .TESTER is omitted will be SENTRY7.

5.22 .TGEN

The .TGEN command (Timing Generator) defines the timing and formatting of a signal defined in a .TTABLE.

SYNTAX:

.TGEN <mode> <t1> [<t2>] <sig1>[,<dir1>] ... <sigN>[,<dirN>]

<mode> specifies the formatting mode. Valid modes are NRZ, RZ, RO, IONRZ, IORZ, and IORO.

<t1> specifies the delay from T0 for NRZ mode signals.

<t1> and <t2> specify leading and trailing edges of a pulse (from T0) for RZ and RO mode signals.

<sig1> thru <sigN> define the signals controlled by the .TGEN.

<dir1> thru <dirN> define the direction control signals, required when an IO mode is specified. These signals must be defined in a .TTABLE but they do not appear as <sig1> through <sigN> signals in any .TGEN statements.

$10 \leq t1,t2=(2*TPERIOD)-40$
t1 = 1 allowed for NRZ, IONRZ (special case)
$t2 - t1 \geq 10$
Number of .TGEN commands per netlist ≤ 6

EXAMPLES:

```
.TGEN NRZ   50 ABC XYZ
.TGEN RO   100 150 LOW
.TGEN IORZ  50 250 HIGH,HIGH_DIR
```

NRZ signals ABC and XYZ occur 50nS after T0. RO mode signal LOW is a 50nS negative going pulse occurring 100nS after T0.

Bi-directional RZ mode signal HIGH is a 200nS positive going pulse occurring 50nS after T0 when in input mode. Signal HIGH_DIR controls the direction (1=input).

5.23 .TOGGLE

The .TOGGLE command is used when one wants ZyPSIM to produce the .TOG file.

SYNTAX:

.TOGGLE [FULL]

The only argument allowed is the optional FULL.

When the .TOGGLE command is used ZyPSIM will produce the filename.TOG file. This toggle file lists each signal name in the circuit along with a ONE if the signal went high (1,H,+) at any time during the simulation, a ZERO if the signal went low (0,L,-), an HIZ if the signal went to a high impedance condition (-,+,Z) and a DRVN if the signal was driven (0,1,X,L,H,*). There is no indication of when these levels occurred, just that they did occur sometime during the simulation. DRVN will also appear for each signal that is driven during the simulation. This is useful for tri-state signals.

If .TOGGLE is used without the FULL option every signal in the circuit will be included, even the signals that are internal to the standard cell primitive macros, except the internal signals of macros that contain the .OFF command. If the argument FULL is given then every signal in the circuit will be included, even the signals that are internal to the standard cell primitive macros, even if the macros have .OFF commands in them. If the .TOGGLE command is not used then the .TOG file will not be produced at all.

EXAMPLES:

.TOGGLE

.TOGGLE FULL

If there is more than one .TOGGLE command in the netlist the last one that appears will be effective, i.e. the last .TOGGLE command will determine whether FULL is in effect or not. If it is in effect it will override all .OFF's for the whole simulation, no matter where they are (or where they are called) in the netlist.

5.24 .TPERIOD

The .TPERIOD command (Tester Period) specifies the absolute value of the tester period (and simulation cycle) in nanoseconds.

SYNTAX:

.TPERIOD <period>

<period> $\geq$ 100nS for SENTRY7
<period> $\geq$ 50nS for SENTRY20

EXAMPLE:

.TPERIOD 400

Will produce a test rate of 2.5MHz.

5.25 .TPOC

The .TPOC command (Test language Print On Change) is used to minimize output by printing output data to the .TPN/S/F file only when signals in the .TPRINT signal list change state.

SYNTAX:

.TPOC

Arguments are not required or permitted.

5.26 .TPRINT

The .TPRINT command (Test language Print) defines the output signals which are to be printed to the .TPN, .TPS, or .TPF files.

SYNTAX:

.TPRINT <sig1> <sig2> ... <sigN>

<sig1> through <sigN> are the nodenames to be printed.

Only signals which appear in .TGEN or .TSTROBE commands (both for I/O signals) are permitted. (Exceptions are the reserved signal names VDDn, VSSn, and UNUSEDm).

All signals which are device pins should appear in the .TPRINT statement in device pin order. Signals internal to the chip should not be included.

Output is directed to filename.TPN, filename.TPS, or filename.TPF, depending on the .GUARDBAND command (see section for that command). Only one .TPRINT statement is allowed per netlist.

EXAMPLES:

```
.TPRINT   N1  N2  N3

.TPRINT   PIN1  VSS PIN3  ABUS0 VDD  VSS2 BBUS0
```

5.27 .TSTROBE

The .TSTROBE command (Tester Strobe) defines the time at which an output specified in a .TPRINT statement will be printed.

SYNTAX:

.TSTROBE <strobe_time> <sig1>[,<msk1>] ... <sigN>[,<mskN>]

<strobe_time> is specified in nanoseconds with respect to T0 and defines the time at which output signals are sampled and printed to the .TPRINT file.

<sig1> thru <sigN> define the signals to which <strobe_time> applies.

<msk1> thru <mskN> represent optional mask control signals. When present, a mask signal value of "0" means the output is masked, "1" means it will be tested. The default (when masking data is not supplied) is "1". The mask signals must be defined in a .TTABLE, but they may not appear in a .TGEN statement.

$10 \leq$ strobe_time $\leq$ TPERIOD -20

but if single cycle IO switching occurs:

Fast test head: $15 \leq$ strobe_time $\leq$ TPERIOD -25
Slow test head: $50 \leq$ strobe_time $\leq$ TPERIOD -60

Number of .TSTROBE commands per netlist ≤ 2

EXAMPLES:

```
.TSTROBE  200    ABC     XYZ
.TSTROBE  100    ABC,MASK
```

In the first statement, signals ABC and XYZ are strobed 200nS after T0 and are always tested. Since masking signals are not supplied, a default of "1" is used for the mask signal to specify continuous testing of both outputs.

In the second case, the strobe occurs 100nS after T0 and signal ABC will be tested during every T0 cycle for which MASK has a "1" in its .TTABLE.

5.28 .TTABLE

A .TTABLE command (Test language Table) allows cycle based input data to be defined in a table format. Using a .TTABLE is the only means of defining tester compatible simulation input data.

SYNTAX:

.TTABLE <sig1>[,<iv1>] <sig2>[,<iv2>] ... <sigN>[,<ivN>]

Where <sig1> through <sigN> are the signals to be driven and <iv1> through <ivN> are optional initial values for NRZ mode signals (may be 0 or 1).

A .TTABLE statement must be followed by table data as follows:

<count> <data1> <data2> ... <dataN>

<count> specifies the number of times that the associated data will be applied.

<data1> thru <dataN> are "0" or "1" corresponding to the signals specified in the .TTABLE.

Data statements may be defined in a subroutine contained within a .TTABLE. Subroutines are defined as follows:

```
(<label_name>
<count>  <data1> <data2> ... <dataN>
.
.etc
.
<count>  <data1> <data2> ... <dataN>
)[<*n>]
```

(<label_name> defines a routine called <label_name>. The name must immediately follow the left parenthesis without spaces. Names may contain up to 16 characters.

)<*n> terminates the routine.

n specifies the number of times to execute the routine. If n is 0, the routine is defined but not executed. If n is omitted, the routine is repeated as many times as necessary until .MAXTIME is reached. If <*n> is omitted altogether, a default of *1 (one execution) will be used.

A previously defined routine is executed by placing a "call" within a .TTABLE as follows:

(<label_name>)*n

Subroutines may be nested to any level.

EXAMPLES:

```
.TTABLE    ABC    XYZ,1
   20       1      0
(SUB_1
   10       1      1
  100       0      0
    5       1      0
)*5
(SUB_2
   23       1      1
   11       0      1
(SUB_3
    5       1      1
    2       0      1
    3       1      0
)*2
    2       0      0
)*3
(SUB_1)*10
(SUB_2)*20
```

Signal XYZ is initialized to 1 (implying it was defined as NRZ in its associated .TGEN).

Subroutine SUB_1 is executed 5 times when defined and later called 10 times.

Subroutine SUB_2 contains a further subroutine SUB_3. SUB_2 is executed 3 times when defined and later called 20 times.

5.29 .XREF

The .XREF command (Cross Reference) is used when one wants ZyPSIM to produce the .XRE file.

SYNTAX:

.XREF

Arguments are not required or permitted.

When the .XREF command is used ZyPSIM will produce the filename.XRE file. This cross reference file has two sections. The first lists each signal name in the circuit followed by the time delay and rise/fall time information for that signal, both intrinsic and load dependant. The second section lists each signal and tells which elements drive that signal and which elements that signal drives.

Internal signals within the primitive macros used to model standard cells are not included in the .XRE file. If the .XREF command is not used then the .XRE file will not be produced at all.

EXAMPLE:

.XREF

6 PRIMITIVE ELEMENTS

When a ZyPSIM standard cell is used in a netlist this is actually a macro call to a macro with the standard cell's name that resides in one of the global libraries. For instance the following line in a netlist

```
U54  IAA4  IN=U53  OUT=U54
```

is really a call of the macro named IAA4 which resides in the Zy40000 and Zy50000 global libraries (which one is used depends on which library is invoked when ZyPSIM is run). These macros are different from a user defined macro because they are composed of ZyPSIM primitive elements in such a way that they accurately model the electrical performance of the physical standard cell to which they correspond. The primitive elements themselves, though, have no physical reality and the primitive logical elements that are used to model the cell may or may not correspond closely to the actual gate configuration of the physical cell. For example, a standard cell buffer that was composed of three inverters in series might be modeled using a single primitive inverter that included the loading and delay information needed to mimic the three stages of the actual cell. Thus when ZyPSIM is run the electrical modeling information is called up but when the circuit is routed the artwork database needed to make the various mask layers will be used instead. Why go into this? Because it emphasizes the fact that primitives are not physical entities, and it explains why the ZyPSIM model information may not be what one would expect after looking at the data sheet schematic for a given cell.

Normally users will only use primitives indirectly as mentioned above. There is one exception, though, and that is the use of CAP primitives on the output cells to simulate SENTRY loading. Because these CAP's are primitives they cannot cause real caps to appear on the final chip; that would happen only if standard cell analog capacitor slices were used.

6.1 GENERAL INFORMATION

The syntax for primitive elements varies from element to element, but the general format is:

```
                              L
<gate_ID>  <gate_name>[W]  [timing_info]
                              H
+ <in_keywd1>=<i1_sig1>[:<i1_cap1>] ...  [,<i1_sigN>[:<i1_capN>]]
+      .
+      .
+ <in_keywdM>=<iM_sig1>[:<iM_cap1>] ...  [,<iM_sigN>[:<iM_capN>]]
+ [<out_keywd1>=<out1_sig>[:<out1_cap>]]
+ [<out_keywd2>=<out2_sig>[:<out2_cap>]]

     where 1 ≤ M,N ≤ 12
```

To describe a primitive, its ID, name, and keywords plus I/O signals must be specified. Timing information (propagation delays, etc.) and input and

output capacitances are optional. The ID is any name which uniquely specifies that particular element. The name is one of the following: AND, OR, NAND, NOR, INV, TAND, TOR, TNAND, TNOR, TINV, XOR, XNOR, ANDOR, ANDORI, ORAND, ORANDI, DFF, JKFF, RAM, ROM, RES, UXFR, CAP, WIDTH, SETUP, HOLD, SLOPE. Primitive elements have NORMAL as their normal output strength. If a weak element is desired, simply append a 'W' to the element name. If the output is to be strong only in its low state, append an 'L'. If the output is to be strong only in its high state, append an 'H'. This L, W, or H suffix may be appended to the element name of any primitive, however, for the following primitives: RES, CAP, RAM, ROM, UXFR, WOR, SETUP, HOLD, SLOPE, WIDTH it has no effect and these elements will respond as if they did not have the L, W, or H appended. Examples of inverters of various strengths:

			OUTPUT
NORMAL	INV	1,1 IN=A OUT=B	0,1,X
WEAK	INVW	1,1 IN=A OUT=B	L,H,*
STLOW	INVL	1,1 IN=A OUT=B	0,H,*
STHIGH	INVH	1,1 IN=A OUT=B	L,1,*

6.2 LOGIC LEVEL AND SIGNAL STRENGTH

A ZyPSIM signal has both a logic level and a signal strength. The logic level is either Zero, One, or Indeterminate, and there is a strength hierarchy that subdivides each of these levels. But at the input of a gate only the logic level is important; a '-' is the same as a '0' and a '+' is the same as a '1'. A signal's strength comes into play when the outputs of two or more signals are 'fighting' at a single node. The strongest signal will win and the logic level and signal strength of the node will have the values of the strongest signal. Once this is determined for a given node at a given time the strength becomes unimportant since the gate inputs that the node feeds respond only to logic levels and treat all the strengths of a logic level as the same (i.e. as Zero, One, or Indeterminate).

When different logic levels of the same strength fight one another the result is an Indeterminate of that strength.

Spikes constitute a special case. When a spike fights with any other signal the output is always a spike. Thus, in one sense, the spike is the strongest ZyPSIM signal. However its logic level is essentially a 'null' since it will not be seen by the input of any gate (except the WOR gate). For these reasons spikes have been left off the STRENGTH HIERARCHY CHART near the end of this document.

6.3 COMPLEMENTED INPUT SIGNALS

To add flexibility, any input signal to any primitive element may be preceeded by a minus sign to indicate that the complement of the signal is to be used as the logic level. This must not be used for the inputs of standard cell library macros (e.g. IAA4, GAA1, etc.) but is only for

primitives. The use of this feature does not add any overhead to the simulation, whereas the inclusion of an inverter does.

EXAMPLE: GATE1 AND 5,10 IN=-A,B OUT=C

This element would be essentially equivalent to

```
NA     INV      IN=A     OUT=NA
GATE1  AND 5,10 IN=NA,B OUT=C
```


6.4 PROPAGATION DELAYS

Propagation delays, setup and hold times, and slope times comprise the timing information that may be used by a ZyPSIM primitive. Setup and hold times are only used by the D and J/K Flip-Flop primitives. Propagation delays (and slope times) are expressed as a function of the element itself and the load which the element drives. There is an intrinsic, or no-load, delay, plus a delay which is a linear function of the capacitance driven by the element. There is a separate delay for low-to-high and high-to-low transitions of the output. The general expression follows and is referred to as "timing_info" in syntax statements.

```
  <iplh>[+<cplh>][,<iphl>[+<cphl>]]
+ [,S:<set_lh>[,<set_hl>]][,H:<hold_lh>[,<hold_hl>]]
+ [:<islh>[+<cslh>][,<ishl>[+<cshl>]]]
```

where:

iplh = intrinsic low-to-high propagation delay

cplh = low-to-high delay per picofarad of load capacitance

iphl = intrinsic high-to-low propagation delay

cphl = high-to-low delay per picofarad of load capacitance

islh, cslh, ishl, cshl are similarly defined except that they refer to slope times.

The delays are not limited to integer values -- the time delay can be expressed as a decimal fraction of a nanoseconds per picofarad (pF). Once the entire network has been read, and the load is known for each node, a value will be calculated for the low-to-high and high-to-low delays for each element. The delay will be rounded to the nearest nanosecond unless the .RESOLUTION command is used in which case the delay will be rounded to the nearest tenth nanosecond. If the element has the same delay for both low-to-high and high-to-low time, the delay need only be specified once. If the delay is omitted, a zero delay will be used.

Delay specified	iplh	cplh	iphl	cphl
<null>	0	0	0	0
A	A	0	A	0
A+B	A	B	A	B
A,B	A	0	B	0
A,B+C	A	0	B	C
A+B,C	A	B	C	0
A+B,C+D	A	B	C	D

EXAMPLES:

```
GATE1 INV 2.4+0.3,1.2+0.7 IN=INPUT OUT=OUTPUT

GATE2 INV 2.4+0.3         IN=INPUT OUT=OUTPUT

GATE3 INV 5,10            IN=INPUT OUT=OUTPUT

GATE4 INV 1               IN=INPUT OUT=OUTPUT

GATE5 INV                 IN=INPUT OUT=OUTPUT
```

NOTES:

GATE1: The low to high propagation delay is 2.4 nanoseconds plus 0.3 nanoseconds per picofarad of load. The high to low delay is 1.2 nS plus 0.7 nS/pF.

GATE2: Both the low to high and high to low delays are 2.4 nanoseconds plus 0.3 per picofarad of load.

GATE3: The low to high delay is 5, the high to low delay is 10.

GATE4: Both the low to high and high to low delays are 1 nanosecond.

GATE5: Both the low to high and high to low delays are zero.

6.5 SETUP AND HOLD TIMES

JK flip-flops and D flip-flops may have their setup and hold times specified within the primitive statement. The setup time specifies how long the data (i.e. J,K,D) must be held stable before the rising edge of the clock, while the hold time specifies how long it must remain stable after the rising edge. After the propagation delays, a ',S:' or ',H:' may be used to specify the setup and hold times respectively.

EXAMPLES:

```
FF1 JKFF 2.4+.3,1.2+.7,S:12,H:10 D=DIN CLK=CLKIN Q=A NQ=NA

FF2 DFF 2.4+0.3,S:10 D=DIN CLK=CLKIN SET=SET RST=RST Q=A NQ=NA

FF3 DFF ,S:20,H:5 D=DIN CLK=CLKIN Q=Q3
```

NOTES:

FF1: The setup time is 12 nS, the hold tine is 10 nS.

FF2: the setup time is 10 nS. Hold time is 0 nS since not specified.

6.6 SLOPE TIMES

In order to check for slowly rising or falling signals (especially on the clock input to flip-flops), another parameter may be specified: the length of time from when the output starts to change and when it reaches its final state, excluding the time it takes before the output starts switching. For complex gates, this may be significantly different than the propagation delay. We'll call this rise/fall time the slope time with the understanding that we are referring to a time in nanoseconds, not a voltage/time ratio. The slope times are load dependent and are used in conjunction with the SLOPE gate. The slope times are looked at once: when the entire circuit has been read, the slope times are compared against any SLOPE gates for that signal. After that, the slope times and SLOPE gates are removed from the circuit. The slope times must follow all other timing information. A colon must be given, followed by the rising slope time and the falling slope time in the same format as the rise and fall delays.

EXAMPLES:

```
GATE1 INV 2.4+0.3,1.2+0.7:1.1+0.2,0.7+0.7 IN=INPUT
+         OUT=OUTPUT

GATE2 INV 2.4+0.3:1.1+0.3                  IN=INPUT OUT=OUTPUT
```

NOTES:

GATE1: The rising slope time is 1.1 nanoseconds plus 0.2 nanoseconds per picofarad of load, the falling slope time is 0.7 nanoseconds plus 0.7 nanoseconds per picofarad of load.

GATE2: Both the rising and falling slope times are 1.1 nanoseconds plus 0.3 nanoseconds per picofarad of load.

6.7 INPUT LOADS

When the input signals are specified, the load for the input may also be given. If it is left out, a unit load, 1 picofarad, will be used. The load may be given by typing the signal name, a colon, and the number of picofarads for that input. For example, an inverter which presented a load of .3 pf to signal CO would be entered as:

```
CO_LOAD INV 12+2.5,10+1.8 IN=CO:0.3  OUT=CO*
```

6.8 WIRE OR

Primitive gates may have their outputs wire-ored implicitly or explicitly. An implicit wire-or is when the same name is used as the output of two or more gates. An explicit wire-or is when each gate has a unique output name, and a WOR gate is used. The following are equivalent:

```
NEN    INV  IN=EN  OUT=NEN
GATE1 TINV  ENABLE=EN  IN=IN1 OUT=BUSC
GATE2 TINV  ENABLE=NEN IN=IN2 OUT=BUSC

NEN    INV  IN=EN  OUT=NEN
GATE1 TINV  ENABLE=EN  IN=IN1 OUT=BUSA
GATE2 TINV  ENABLE=NEN IN=IN2 OUT=BUSB
GATE3  WOR  SIG=BUSA,BUSB SIG.OUT=BUSC
```

6.9 KEYWORDS AND PRIMITIVES

We mentioned above how standard cells can be specified in a netlist by either using keywords or following the model format order for inputs and outputs. Primitives must use keywords. The keywords for each element are given in the following sections which describe the elements in detail and a PRIMITIVE KEYWORD REFERENCE is also included.

6.10 MACROS AND PRIMITIVES

The macro syntax is exactly the same for macros composed of primitives as it is for macros made of standard cells. The only differences have to do with element ID and signal name length restrictions.

6.10.1 NESTED ELEMENT ID'S

We said before that an element ID expanded through all its macro nesting levels may be up to 23 characters long. This applies only to the element ID's of standard cells. The expanded element ID for a primitive gate may be 80 characters long.

6.10.2 INTERNAL SIGNAL NAMES

The internal signal names of primitive macros may be up to 80 characters long. One way to tell whether a signal name may be 23 or 80 characters long is to ask, "Will this signal be a physical node on the actual chip?" If so, then its expanded name may be 23 characters long, if not, then it may be 80 characters long.

6.10.3 MACRO USE

The choice of ID's inside primitive macros can be important, especially if the macros are placed in a library for general use. It

is also true if setup times, hold times, or signal width checks are performed, since the gate ID is used as part of the error message. The gate ID could help the user to identify the source of the error.

For example, if a macro was named FAC1, and contained a D flip-flop and a Width gate, a macro could be written as follows:

```
.MACRO FAC1 D CLK SET RST Q NQ
FAC1 DFF 40+5,S:10,H:5 D CLK SET RST Q NQ
FAC1_CLOCK_WIDTH WIDTH 50,50 CLK
FAC1_RESET_WIDTH WIDTH 0,20 RST
.EOM
```


If the user had 50 flip-flops in a circuit, some of which were FAC1, and had named them FFLOP1 thru FFLOP50, and the setup time was violated for one of them, the following message would be printed:

```
***SETUP TIME VIOLATION FOR ELEMENT: FFLOP10.FAC1 AT TIME:  119***
```

If the clock width was too narrow, the following message would appear:

```
***WIDTH OF SIGNAL: ABC WAS NOT HIGH AT LEAST 50 IN ELEMENT:
   FFLOP10.FAC1_CLOCK_WIDTH AT TIME=  119***
```

Notice that FFLOP10.FAC1_CLOCK_WIDTH is 24 characters long. This is more than 23 but the element is a width gate--a primitive--so this is OK (the limit is 80). The element ID of the standard cell FAC1 would be FFLOP10 (assuming it is not nested), which is only 7 characters and so is less than 23.

6.11 PRIMITIVE ELEMENT TYPES AND SYNTAX

The following sections list the ZyPSIM primitive elements in detail along with their syntax and examples. The ZyPSIM models for the ZyP standard cells are built from these primitives.

6.11.1 SIMPLE GATES (AND,OR,INV,NAND,NOR,XOR,XNOR)

Simple gates are the standard logic elements: AND, OR, INV, NAND, NOR, XOR, XNOR. These gates have only one type of input and the AND, OR, NAND and NOR may have twelve of them. The INV may have only one input while the XOR and XNOR may have up to two inputs. There is only one output. The value of the output depends only on the logic level of the inputs, and is independent of whether the inputs are strong, weak or high-impedance.

KEYWORDS: IN Input signals to the gate.

OUT Output signal

EXAMPLES:

```
GATE1 AND 5,5 IN=IN  OUT=OUT

GATE2  OR 7   IN=A,-B OUT=C

GATE3 INVL 12+2,3 IN=A OUT=C

GATE4 NAND 12+2  IN=A,B,C OUT=OUT

GATE5 XORW 0    IN=A,B OUT=OUT

GATE6 XNOR 1,1     IN=A,B OUT=XNOR
```

NOTES:

GATE1: Has a single input.

GATE2: Uses the complement of signal B as an input.

GATE3: Output is a weak 1 and strong 0.

GATE5: Outputs a weak 1 and a weak 0.

6.11.2 COMPOUND GATES (ANDOR,ANDORI,ORAND,ORANDI)

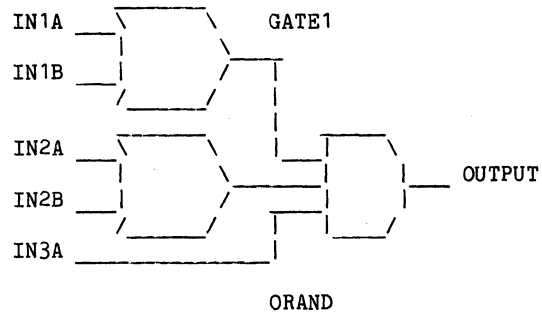
ANDOR and ORAND gates differ from the 'simple' gates in the input format. For these gates there are up to 12 inputs, each of which can be comprised of up to 12 signals for a total of 144 inputs. For example, a 3 wide, 2-2-1 ANDORI gate has three inputs, the first has two signals, the second has two signals, and the third has one signal. The keyword, IN, may be repeated as many times as needed (up to 12 times).

KEYWORDS: IN - Specifies a list of signals to be ANDed together for ANDOR and ANDORI gates, and ORed together for ORAND and ORANDI gates.

OUT Output signal.

EXAMPLES:

```
GATE1 ORAND  4,5 IN=IN1A,IN1B IN=IN2A,IN2B IN=IN3A
+                OUT=OUTPUT

GATE2 ORANDI 4,5 IN=IN1A IN=IN2A,IN2B,IN2C,IN2D
+                OUT=ORANDI_OUT
```



ORAND

6.11.3 TRI-STATE GATES (TAND,TOR,TINV,TNAND,TNOR)

Tri-state gates have an additional input, the enable, which controls whether the gate is on or off. If the enable signal is high, the gate is turned on, and functions as a normal AND, OR, NAND, NOR, or Inverter. When the enable signal is low, the output goes into a high-impedance state with the level being the level of the previous active state of the gate. The input signals will have no effect on the output until the gate is turned back on.

The time delays for the tri-state gates apply to the normal inputs as well as to the enable signal.

KEYWORDS: ENABLE Enable input.

IN Input signals, same as the simple gates.

OUT Output signal

EXAMPLES:

```
GATE1 TOR  5,10 ENABLE=-EN1 IN=A,B OUT=OUT

GATE2 TINV 4,8  ENABLE=EN2 IN=A OUT=INVOUT

GATE3 TNAND 7,8 ENABLE=EN3 IN=A,B,C,D OUT=TNAND

GATE4 TNOR  2+.04 ENABLE=EN4 IN=A,B,C OUT=TNOR
```

NOTES:

GATE1: The signal EN1 is complemented. Thus this gate is enabled when EN1 is low.

6.11.4 SINGLE TRANSISTOR GATE (UXFR)

The UXFR gate models a single transistor. The UXFR is a unidirectional transfer gate which allows the input signal to propagate to the output if the gate is at a logic 1 level. Both the level and the strength of the input signal propagate (except that a FORCING strength is dropped to NORMAL strength).

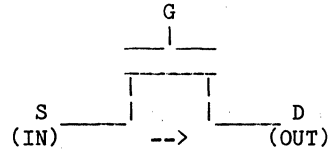
The propagation delay times for the UXFR are gate to drain delays, NOT the time it takes the source to propagate to the drain. The source to drain delay will be zero. All slope times are zero and may not be specified.

The UXFR gate has the further property that loads to either the source or drain side are seen by both sides. In other words, a gate which drives the source of a UXFR gate will have its capacitive load increased by the load on the drain side of the UXFR as well as the load on the source side. This capacitive load will be the same whether the UXFR is turned on or turned off.

KEYWORDS: G Gate signal

S Source signal

D Drain signal



EXAMPLES:

```
GATE1 UXFR 10,5 G=-GATE S=IN D=OUT

GATE2 UXFR 10,5 G=GATE S=INPUT D=OUTPUT
```

NOTES:

GATE1: Will turn on when signal 'GATE' is low.

6.11.5 WIRE-OR GATE (WOR)

ZyPSIM allows the wire-oring of gates to be handled in two ways; simply use the same name as the outputs of each gate, or use the WOR gate. The following are equivalent:

```
GATE1 TINV  ENABLE=EN  IN=IN1 OUT=BUSC
GATE2 TINV  ENABLE=-EN IN=IN2 OUT=BUSC

GATE1 TINV  ENABLE=EN  IN=IN1 OUT=BUSA
GATE2 TINV  ENABLE=-EN IN=IN2 OUT=BUSB
GATE3  WOR  SIG=BUSA,BUSB SIG.OUT=BUSC
```

The WOR gate does not permit delay times--i.e. when one of the signals changes values, it affects the output immediately.

KEYWORDS: SIG signals to be wire-ored

SIG.OUT output signal

EXAMPLES:

```
GATE1  WOR SIG=CO,CARRY_OUT  SIG.OUT=COUT

GATE2  WOR SIG=CI,CARRY_IN  SIG.OUT=CIN
```

6.11.6 RESISTOR (RES)

The RES gate is used to simulate the reduction in drive capability that occurs when a resistor is placed in series with the output of a real gate. Therefore the output of the RES gate is generally one increment of strength weaker than the input signal. Thus, if the input was a .CLK signal, or .ONE or .ZERO, or the output of a strong gate, the output of the resistor would be weak. If the input was itself a weak signal, the output would be still weaker. ZyPSIM has 61 different weak signal strengths, each of which can override a signal weaker than itself.

KEYWORDS: IN input signal

OUT output signal

EXAMPLES:

```
ELEMENT1 RES 10,10 IN=.ONE OUT=BUS0

ELEMENT2 RES 20    IN=BUS0 OUT=RBUS0

ELEMENT3 RES 40    IN=RBUS0 OUT=RRBUS0
```

NOTES:

ELEMENT1: Pullup resistor. The output is a weak 1.

ELEMENT2: If BUS0 is driven by a normal element, RBUS0 will be a weak signal. If BUS0 were driven by ELEMENT1, RBUS0 would be 1 level weaker than the normal 'weak' signal.

ELEMENT3: RRBUS0 will be even weaker than the output of ELEMENT2.

6.11.7 CAPACITOR (CAP)

Primitive capacitors may be inserted into the network, typically to simulate the additional load on signals due to the interconnections between gates. These elements have no affect on simulation CPU time, as they are used only to calculate delay parameters. Timing information may not be specified for a CAP gate (since CAP gates do not drive anything).

KEYWORD: IN input signal (node to be loaded)

EXAMPLE:

```
GATE1 CAP IN=INPUT:0.3
```

NOTES:

GATE1: An additional capacitance of 0.3 pF is added to signal 'INPUT'.

6.11.8 D FLIP FLOP (DFF)

The D input to the flip-flop is clocked to the output on the low-to-high transition of the clock input. The set and reset inputs are asynchronous. If the set input is low, the Q output will be set to a one. If the reset input is low, the Q output will be reset to a zero. If both set and reset are low, the Q and NQ outputs will both go to a one until either set or reset goes high.

Propagation delay and slope time information applies to the CLK to Q, NQ; RST to Q, NQ; and SET to Q, NQ transitions.

In addition to the normal delays, the DFF can also have setup and hold times specified. The setup time is the minimum time for which the D input must be stable before the clock goes high, the hold time is the minimum time the D input must be stable after the clock has gone high. Separate setup and hold times may be specified for D = 1 and D = 0 levels.

KEYWORDS:

D	D input
CLK	Latches D input on low-to-high transistion.
SET	Set input--causes Q to go to 1. Active low.
RST	Reset input--causes NQ to go to 1. Active low.
Q	Q output
NQ	Q bar output

NOTES:

The SET and RST inputs are optional. If they are not specified, they will default to logic level 1 (inactive).

Timing information applies to both Q and NQ. Low-to-high timing information applies to the rising edge of Q and the rising edge of NQ

EXAMPLES:

```
FF1 DFF 5+.1,10+0.2,S:5,7,H:4,3:2+.1,3+1  D=A CLK=B SET=C
+    RST=D Q=E NQ=F

FF2 DFF 5+.1,10+0.2,S:5,H:4 D=A CLK=B SET=C RST=D Q=E NQ=F

FF3 DFF 5+0.1,10+0.2,S:5     D=A CLK=B Q=E

FF4 DFF 5,10        D=A CLK=G SET=C RST=D Q=E
```

NOTES:

FF1: All possible timing information is specified. The propagation delay for the low to high transition of Q or NQ from the rising edge of CLK or the falling edges of RST or SET is 5 nanoseconds plus 0.1 nanosecond per pF of load capacitance. For the high to low transition it is 10 nanoseconds plus 0.2 nanoseconds per pF. The setup time for a high level is 5 nanoseconds and 7 nanoseconds for a low level, i.e., the D input must be a stable HIGH at least 5 nanoseconds before the rising CLK edge, or be a stable low at least 7 nanoseconds. The hold times are 4 and 3, i.e., D must continue to be a stable high for 4 nanoseconds after the rising CLK edge or be a stable low for 3 nanoseconds. The rise times (slope times) for the Q and NQ outputs will be 2 nanoseconds plus 0.1 nanoseconds per pF of load and the fall times will be 3 nanoseconds plus 1 nanosecond per pF.

FF2: The setup time is 5 for both a 1 and 0 on the D input. The hold time is 4.

FF3: The setup time is 5 for both a 1 and 0. The hold time is zero. Also, the Set, Reset and NQ signals are not used.

FF4: There are no setup and hold times. The NQ signal is not used.

6.11.9 J/K FLIP FLOP (JKFF)

The J/K inputs to the flip flop are clocked on the low to high transition of the clock input. If J=0 and K=1, a 0 will be clocked to the Q output. If J=1 and K=0, a 1 will be clocked to the Q output. If both are zero, there will be no change. If both are 1, the flip-flop toggles. Like the DFF, the SET and RST inputs are asynchronous and active low, and may be omitted if keywords are used. Also, setup and hold times may be specified similarly to the DFF.

KEYWORDS:

J	J input
K	K input
CLK	Latches the input on the low-to-high transistion.
SET	Set input--causes Q to go to a 1. Active low.
RST	Reset input--causes NQ to go to a 1. Active low.
Q	Q output
NQ	Q bar output

NOTES:

The SET and RST inputs are optional. If they are not specified, they will default to logic level 1 (inactive).

Timing information applies to both Q and NQ. Low-to-high timing information applies to the rising edge of Q and the rising edge of NQ

EXAMPLES:

```
FF1 JKFF 5,10,S:8,H:5 J=A K=B CLK=C SET=D RST=E Q=F NQ=G

FF2 JKFF 5,10,H:5     J=A K=B,CLK=C SET=D NQ=G

FF3 JKFF 10,S:4,5     J=A K=B CLK=C SET=D RST=E Q=F NQ=G
```

NOTES:

FF1: Both the Setup and Hold times are given.

FF2: The Setup time is omitted--only a Hold time is specified. Also, the Reset input and Q output are not specified.

6.11.10 SETUP GATE (SETUP)

The SETUP gate allows setup times to be specified for any signals in the network. It will print an error message if the input signal has not met the low or high setup times with respect to the low to high

transistion of the clock signal. This allows the same checks to be made on specially constructed models for flip-flops, counters, etc., as are provided by the built-in flip-flop models. Separate setups for the 1 and 0 logic levels of the input data may be specified.

The SETUP gate provides zero capacitive load to its input signals.

Choose the SETUP gate ID carefully, since it will be included in the error message if the setup is violated. An informative ID will help in identifying the source of the error, particularly in large networks. The error message for a gate named FAC1_SETUP could be the following:

SETUP TIME VIOLATION FOR ELEMENT: FAC1_SETUP AT TIME: 89

KEYWORDS: IN the input signal

CLK The signal whose rising edge will be used as the reference point.

EXAMPLES:

```
SET1 SETUP 10,5 IN=A CLK=B

SET2 SETUP 5,10 IN=C CLK=-B
```

NOTES:

In gate SET1, signal A must be high 10 nS before the rising edge of B and A must be low 5 nS before the rising edge of B or a runtime error message will be given.

In gate SET2, the setup is with respect to the falling edge of signal

6.11.11 HOLD GATE (HOLD)

The HOLD gate allows hold times to be specified for any signals in the network. It will print an error message if the input signal has not met the low or high hold times with respect to the low to high transistion of the clock signal. In other words, when the CLK signal makes its transition the IN signal must not change state until the specified hold time for that level has been exceeded. This allows the same checks to be made on specially constructed models for flip-flops, counters, etc., as are provided by the built-in flip-flop models. Separate holds for the 1 and 0 logic level of the input may be specified.

The HOLD gate provides zero capacitive load to its input signals.

Choose the HOLD gate ID carefully, since it will be included in the error message if the hold is violated. An informative ID will help in identifying the source of the error, particularly in large networks. The error message for a gate named FAC1_HOLD could be the following:

HOLD TIME VIOLATION FOR ELEMENT: FAC1_HOLD AT TIME: 89

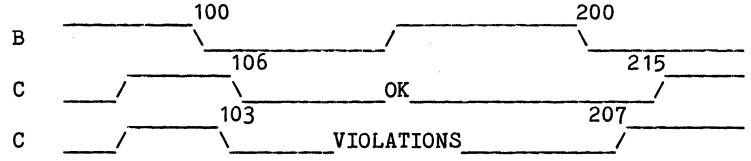
KEYWORDS: IN the input signal

CLK The signal whose rising edge will be used as the reference point.

EXAMPLES:

```
HOLD1 HOLD 10,5 IN=A CLK=B

HOLD2 HOLD 5,10 IN=C CLK=-B
```

NOTE: For Gate HOLD2, the hold time is with respect to the falling edge of signal B. For example:



6.11.12 WIDTH GATE (WIDTH)

The WIDTH gate allows minimum widths to be enforced on critical signals such as the clock and reset signals of a flip-flop. It allows separate high and low widths to be specified. The gate has a single input--the signal to be checked. If the signal does not remain high or low for the required time, the following error message will be printed:

WIDTH OF SIGNAL: X WAS NOT HIGH AT LEAST 5 IN GATE: Y AT TIME 9

The WIDTH gate provides zero capacitive load to its input signal.

KEYWORD: IN the signal to be checked.

EXAMPLES:

```
WID1 WIDTH 40,50 IN=CLK2

WID2 WIDTH 70    IN=CLK3
```

NOTES:

WID1: CLK2 must be high at least 40 ns, and low at least 50 ns.

WID2: CLK3 must be high at least 70 ns, and low at least 70 ns.

6.11.13 SLOPE GATE (SLOPE)

The SLOPE gate is used in conjunction with the slope times. It will cause an error message to be printed out if the maximum rising or falling slope time is violated for the SLOPE gate input signal. It is particularly useful in checking that the clock input to a flip-flop does not exceed a maximum rise or fall slope time.

The SLOPE gate is evaluated only once--after the circuit has been compiled. It does not affect the amount of CPU time which will be spent during the simulation.

EXAMPLES:

```
SLOPE1 SLOPE 40,50 IN=CLK2

SLOPE2 SLOPE 70    IN=CLK3
```

NOTES:

WID1: CLK2 must have a rising slope of less than 40 ns, and a falling slope time less than 50 ns.

WID2: CLK3 must rise no slower than 70 ns, and fall no slower than 70 ns.

6.11.14 ROM

The ROM gate in ZyPSIM may be used as a PLA as well as a ROM. It is composed of two parts: The definition of the signals to be used as ENABLE, READ, Address, and Outputs, followed by a table of values which will be used as the contents of the ROM. This table is composed of an address, followed by a slash (/) followed by the contents of that address. The addresses may be specified in any order. In fact, the same address may be specified more than once, which is handy for PLA's. In PLA's, some of the addresses may be specified as "don't care" bits--i.e. they will be ignored when that entry in the ROM is looked at to see if the address matches. At the same time, any of the output signals may be specified as '-', which means that if the address matches, those output signals will be unchanged. Thus, two or more entries in the ROM may be used to construct the outputs for a given address.

The ROM table may contain the symbols 0, 1, *, or - for use in either the address or data fields. When used in the address field, the * will match a signal if it is unknown (see DFF example) the '-' means that the address bit is not to be looked at for this entry. In the data field, the '*' puts the corresponding output in the unknown state, the '-' indicates that the output is to be unchanged.

KEYWORDS: ENABLE When high, the ROM is enabled, when low the ROM output is tri-stated. This keyword is optional--if omitted, the ROM is always enabled.

R Read signal--when high, the ROM can be read. When low, the ROM output is tri-stated. This keyword is optional--if omitted, the ROM can always be read.

ADDR Address signals.(MSB -> LSB)

OUT Output signals. (MSB -> LSB)

EXAMPLE:

```
ROM1 ROM 10,5 ENABLE=E R=B ADDR=A,B,C,D
+             OUT=Q1,Q2,Q3
+        0000/111  /101
+        10--/1--
+        1--1/--0
+        1--0/--1
+        1-1-/-1-
+        1-0-/-0-
+        ----/000
```

NOTES:

a. Location zero contains all ones. Location 1 contains a 101.

b. If the first address bit is high, the output is dependent only on the 3rd and 4th address bit--the 2nd address bit is ignored.

c. All unspecified addresses contain the default value of '000'

Example of a DFF using a ROM with an output which feeds back to one of the inputs:

```
FAC ROM ADDR=D,CLK,SET,RST,Q,NQ,I  OUT=Q,NQ, I
+           - - 0 1 - - - /  1 0 1
+           - - 1 0 - - - /  0 1 1
+           - - 0 0 - - - /  1 1 1
+           - - 1 1 1 1 - /  * * 0
+           - - 1 1 * * 0 /  * * 0
+           - 0 1 1 0 1 - /  0 1 0
+           0 1 1 1 - - 0 /  0 1 1
+           - 1 1 1 0 1 1 /  0 1 1
+           1 1 1 1 - - 0 /  1 0 1
+           - 1 1 1 1 0 1 /  1 0 1
```

The ROM table may be given in hex or octal by specifying a .HEX or .OCT in front of the data. For example:

```
ROM2 10,5 ADDR=A,B,C,D OUT=Q1,Q2,Q3,Q4,Q5
+  .HEX
+  0/0A /1B /0C
+  .OCT
+  10/17 /16 /37
```

6.11.15 RAM

The RAM gate is composed of two parts: The definition of the signals to be used ENABLE, READ, Address, Inputs, and Outputs, followed by a table of values which will be used as the initial contents of the RAM. This table is composed of an address, followed by a slash (/) followed by the contents of that address. The addresses may be specified in any order.

The RAM table may contain the symbols 0, 1, or - for use in the address field, and the symbols 0, 1, or * for use in the data field. If the address field is composed of all '-', the corresponding data field will be used as the contents of all uninitialized RAM locations. In the data field, the '*' puts the corresponding output in the unknown state.

KEYWORDS:

ENABLE	When high, the RAM is enabled, when low the RAM outputs are tri-stated. This keyword is optional--if omitted, the RAM is always enabled.
R	Read signal--when high, the RAM can be read. When low, the RAM is written into. When the RAM is written into, the outputs are tri-stated.
ADDR	Address signals.
IN	Input data signals.
OUT	Output signals.

EXAMPLE:

```
R1 RAM 10,5 ENABLE=E R=B ADDR=A,B,C,D IN=I1,I2,I3 OUT=Q1,Q2,Q3
+        0000/111   /101
+        ----/000
```

NOTES:

a. Location zero contains all ones. Location 1 contains a 101. These will change if the location is written to.

b. All unspecified addresses contain the default value of '000'

6.12 STRENGTH HIERARCHY CHART

When we say that ZyPSIM is a 64 state simulator we mean there are 64 strength levels. We will designate them by the numbers 63 through 0 where 63 is the highest/strongest and 0 is the lowest.

#	Name	Symbols	How generated
63	FORCING	0,1,X	0,1,X clock; .ZERO, .ONE
62	NORMAL	0,1,X	Normal gate output
61	WEAK	L,H,*	Weak gate output; L,H,* clock; resistor from 63.
60	WEAK	L,H,*	Resistor(s) on clock signal or on output of gate.
.			
.			
.			
2	WEAK	L,H,*	
1	WEAK	L,H,*	
0	HI-Z	-,+,Z	Tri-state gate when not enabled; -,+,Z clock; Resistors (many) on clock signal or gate output.

NOTES:

a. "clock" above means the output of a .CLK or .TABLE statement. A .CLK can only put out 0/1 or L/H.

b. For strengths 63 and 62 the first resistor added drops the strength level by 2 (i.e. 63 to 61 and 62 to 60).

c. For strengths 61 through 1 every resistor added drops the strength level by 1 (e.g. 61 to 60, 42 to 41).

d. Any -/+/Z signal has the same strength level (0) as any other. There is no lower level.

e. A UXFR gate does not change the strength of a signal passing through it except that it drops a FORCING signal to a NORMAL one (63 to 62).

f. The 'Symbols' above are given in the order Logic Level Zero, One, Indeterminate.

6.13 PRIMITIVE KEYWORDS REFERENCE

Simple Gates

```
AND     IN   OUT
INV     IN   OUT
NAND    IN   OUT
NOR     IN   OUT
OR      IN   OUT
XNOR    IN   OUT
XOR     IN   OUT
```

Compound Gates

```
ANDOR   IN   OUT
ANDORI  IN   OUT
ORAND   IN   OUT
ORANDI  IN   OUT
```

Tri-state Gates

```
TAND    ENABLE  IN   OUT
TINV    ENABLE  IN   OUT
TNAND   ENABLE  IN   OUT
TNOR    ENABLE  IN   OUT
TOR     ENABLE  IN   OUT
```

Memory elements

```
DFF     D    CLK   SET   RST   Q   NQ
JKFF    J  K CLK   SET   RST   Q   NQ
RAM     ENABLE  R  ADDR      OUT
ROM     ENABLE  R  ADDR  IN  OUT
```

Misc. Gates

```
CAP     IN
RES     IN   OUT
UXFR    G   S   D
WOR     SIG   SIG.OUT
```

Check Gates

```
HOLD    IN   CLK
SETUP   IN   CLK
SLOPE   IN
WIDTH   IN
```

In Alphabetical Order

```
AND      IN   OUT
ANDOR    IN   OUT
ANDORI   IN   OUT
CAP      IN
DFF      D    CLK   SET   RST   Q   NQ
HOLD     IN   CLK
INV      IN   OUT
JKFF     J  K CLK   SET   RST   Q   NQ
NAND     IN   OUT
NOR      IN   OUT
OR       IN   OUT
ORAND    IN   OUT
ORANDI   IN   OUT
RAM      ENABLE  R  ADDR      OUT
RES      IN   OUT
ROM      ENABLE  R  ADDR  IN  OUT
SETUP    IN   CLK
SLOPE    IN
TAND     ENABLE  IN   OUT
TINV     ENABLE  IN   OUT
TNAND    ENABLE  IN   OUT
TNOR     ENABLE  IN   OUT
TOR      ENABLE  IN   OUT
UXFR     G   S   D
WIDTH    IN
WOR      SIG   SIG.OUT
XNOR     IN   OUT
XOR      IN   OUT
```

APPENDIX A

The ZyP system is, by design, computer independent. ZyP software commands are intended to work identically on any computer system on which ZyP is installed. However, operating system commands and structures are often different on computers from differing manufacturers. This appendix describes computer specific ZyPSIM commands for the DEC VAX and PRIME computers.

a. VAX Commands

The command line given below is required for layout capacitance estimation and is inserted into ZyPSIM .NET file. For a detailed explanation refer to the .INPUT command section of this document.

```
.INPUT SYS$ZYPROOT:[ZYPSIM.x]LAYOUTCAP.

    where x is Zy4LIB for the Zy40000 library
           or Zy5LIB for the Zy50000 library.
```

b. PRIME Commands

The command line given below is required for layout capacitance estimation and is inserted into a ZyPSIM .NET file. For a detailed explanation refer to the .INPUT command section of this document.

```
.INPUT ZYP>ZYPSIM>x>LAYOUTCAP

    where x is Zy4LIB for the Zy40000 library
           or Zy5LIB for the Zy50000 library.
```

5

5

ZyPSIM TEST LANGUAGE

Doc: 20-010-309 Rev: B | Issued: July 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: B
Table of Contents	Rev: B
Pages 1-1 through 1-2	Rev: B
Pages 2-1 through 2-7	Rev: B
Pages 3-1 through 3-35	Rev: B
Pages 4-1 through 4-20	Rev: B
Pages 5-1 through 5-10	Rev: B
Pages 6-1 through 6-7	Rev: B
Pages A-1 through A-3	Rev: B
Index	Rev: B

Table of Contents

List of illustrations

1 INTRODUCTION

1.1 SCOPE

This document provides information about a major enhancement to the ZyP system's logic simulator, ZyPSIM. The enhancement is part of a number of enhancements incorporated into ZyPSIM II, and consists of a set of tester oriented commands collectively referred to as the ZyPSIM Test Language. The document is organized as follows.

Section 2 provides a brief description of the tester to refresh failing memories.

NOTE: The ZyP Specification and Test User's Guide provides a more detailed description of the tester.

Section 3 introduces the language in a tutorial fashion.

Section 4 is a summary of things to do (and things not to do) when designing logic and developing test schemes. Read this for a happy ending to your chip.

Section 5 presents an example design showing how the test language is used to produce a tester compatible functional test.

Finally, section 6 is a reference section for the test language command syntax.

1.2 OVERVIEW

The Test Language has been designed to assist ZyP Design System users in the task of generating functional test patterns, which are guaranteed to be tester compatible. ZyPSIM I pattern generation facilities (.CLK and .TABLE) are cumbersome to use and provide too much freedom in terms of timing specifications. This can result in patterns which execute successfully in the simulation environment, but may not do so in the test environment.

The Test Language was therefore developed to address both problems. ZyPSIM can now emulate (from a timing and functional test standpoint) the capabilities of ZyMOS testers. You, the ZyP system user, have the capability to develop tester compatible functional test patterns quickly and conveniently.

In conjunction with ZyTEST, the ZyPSIM Test Language will accurately translate your functional test and timing scheme into the test environment, as well as eliminate errors due to basic simulation/tester incompatibilities. In addition, the structured "only tester timing allowed" features of the language will greatly aid you in developing a clean timing scheme.

1.3 RELATED DOCUMENTS

It is assumed that you are conversant with ZyPSIM and the ZyP system in general. Documents which should be considered necessary reading for a full understanding of this document are as follows:

20-010-009 ZyPSIM User's Guide.
20-010-011 ZyP Specification and Test User's Guide.
20-010-020 ZyTEST User's Guide.

1.4 DOCUMENT CONVENTIONS

The following conventions are used in this document to describe command format and syntax.

a. Arguments which you need to provide are shown in lower case, enclosed in angles. Eg:

   .TESTER <tester_type>

   You must supply a value for <tester_type>. The angles should not be included in your response.

b. Optional arguments are also enclosed in square parentheses. Eg:

   .TSTROBE <strobe_time> <node>[,<mask>]

   You may optionally specify argument <mask>. It must be separated from <node> by a "," comma.

c. One or more periods are used to indicate multiple arguments. Eg:

   .TPRINT <node 1> <node 2> ... <node n>

   You must specify values for nodes to be printed, as required.

2 ABOUT THE TESTER

As mentioned previously, the test language attempts to emulate the tester environment to achieve simulation/tester compatibility. Let's refresh our knowledge of the tester before proceeding with language definition so that we can later see how the commands relate to the tester hardware.

At this point it is assumed that you are familiar with using ZyPSIM and have a rudimentary knowledge of the SENTRY tester gained from the ZyP Specification and Test User's Guide. This discussion will use the simplified tester block diagram shown in Figure 2-1.

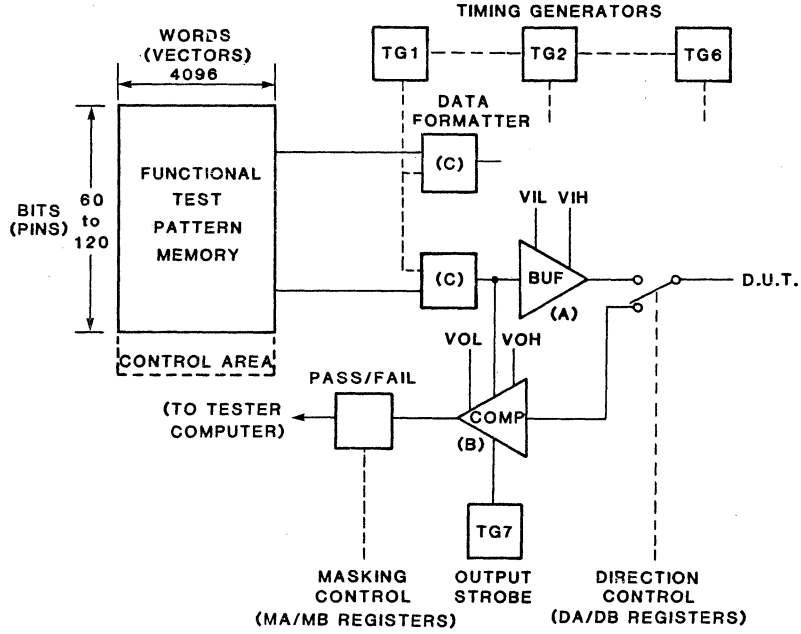


FIGURE 2-1. SIMPLIFIED TESTER BLOCK DIAGRAM

2.1 FUNCTIONAL TEST PATTERN

Test vectors (=patterns, =words) are stored in the high speed pattern memory (hereafter referred to as RAM). During test execution, the RAM is addressed at a fixed frequency (test rate) and the RAM data is applied to, or compared against the device under test (DUT). The clock rate is referred to as the test period and the start of each test period is known as T0 (Tee-zero).

2.1.1 DIRECTION CONTROL

Since the test vectors in RAM can be either input stimuli (driving the DUT), or expected output data (to be compared to the DUT outputs), the tester needs to know which RAM bits (device pins) are inputs and which are outputs. The direction control registers, of which there are two (DA/DB), are used for this function. Each register has as many bits as there are in each word of the RAM, one for each pin. A "1" in a particular bit position defines that pin as an input and means that the corresponding data in RAM will drive the DUT for that pin. A "0" means that the DUT pin is an output and the corresponding data in RAM is the expected DUT output. The direction register is used to control the pin electronics such that the DUT is driven via the buffer (A in Figure 2-1) or drives the comparator input (B in Figure 2-1).

The tester may be commanded (under software control) to use DA or DB on a cycle by cycle basis as the test progresses. In this way, I/O switching of bidirectional pins can easily be accommodated. Note that if more than two sets of I/O definitions are required, the DA/DB registers need to be redefined during test execution. The data for the redefinition is stored in the RAM alongside normal functional test vectors.

2.1.2 MASKING CONTROL

Even though a pin is defined as an output, it is often desired to mask (not to test) a pin or pins for certain test vectors, for example, during device initialization or other periods of uncertainty. Masking control is achieved by enabling/disabling the output of the comparator (B in Figure 2.1). The mask registers, of which there are two (MA/MB), are used for this function. For the mask registers, "1" means test this pin, "0" means don't test this pin. Mask registers can be redefined during test execution and data used in redefinition of register contents is also stored in RAM.

Note that since redefining mask or direction registers costs one memory location, available memory for actual test vectors is reduced each time this occurs.

2.1.3 VOLTAGE LEVELS

For input pins, the actual VIL/VIH values applied to the DUT are determined by the VIL/VIH values supplied to the drive buffer. The data in RAM is used to select the appropriate level: "0" = VIL, "1" = VIH. These values will depend on the type of input pad being driven, CMOS or TTL etc. Two pairs of VIL/VIH levels are available.

Similarly for output pins, the VOL/VOH values supplied to the comparator will be selected by the data in RAM according to the expected logic level from the DUT. Only one VOL/VOH level pair is available. Note also that the comparator is strobed, indicating that comparison is made at a particular point in the cycle (tester period). More on this later.

2.2 FUNCTIONAL TEST TIMING

2.2.1 INPUT TIMING

When data is clocked out of the RAM at a rate determined by the tester clock period, it will be synchronous with T0 time. However, the time at which data is required to be applied to the DUT may be anywhere during the cycle. The timing generators and data formatting control (C in Figure 2-1) are used for this purpose.

2.2.1.1 Timing Generators

A timing generator has two associated timing values, T1 and T2. Both values are with respect to T0 and may be used to:

a) define a single data transition time within a test period (T1 only required).

b) define the leading (T1) and trailing (T2) edges of a pulse within a test period.

T1 has a minimum (hardware dependent) value of 10nS and T2 must be greater then T1 by at least 10nS. The minimum pulse width (T2-T1) is therefore also 10nS.

2.2.1.2 Formatting

The formatting logic can support various types of data formatting (see the Specification and Test User's Guide Figure 2.2). The test language supports the three most commonly used formats, so only these formats will be described here.

2.2.1.2.1 NRZ Format

The data in RAM is essentially already in NRZ format. NRZ stands for "non-return to zero" which simply stated means that when a change of level occurs in a cycle, it does not change again in the same cycle. The format of the data in RAM is NRZ T0 mode, since changes occur synchronously with T0. In this mode, data is passed to the DUT unmodified as shown in Figure 2-2a.

If it is desired to make data transitions at a particular point in the cycle other than T0, then the mode becomes NRZ TG implying the use of a timing generator. This format utilizes (requires) only one timing value, T1. Whenever data in RAM changes from one cycle to the next, the change will occur at time T1 from T0 as shown in Figure 2-2b.

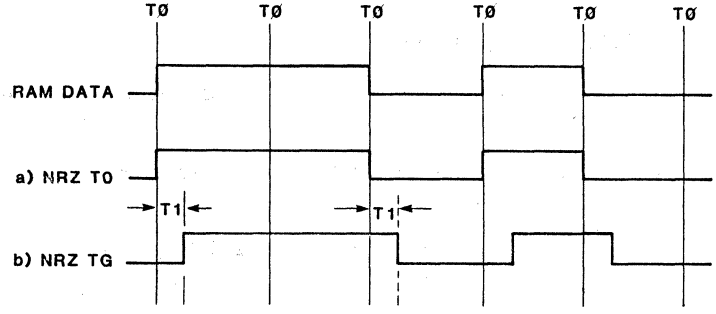


FIGURE 2-2. NRZ FORMAT TIMING

2.2.1.2.2 RZ FORMAT

Return to zero format is used to create a positive going pulse whenever the RAM contains a logic "1". If the RAM contains a logic "0", a pulse will not be generated for that cycle. Both T1 and T2 are required to define the leading and trailing edges of the pulse. This is shown in Figure 2-3.

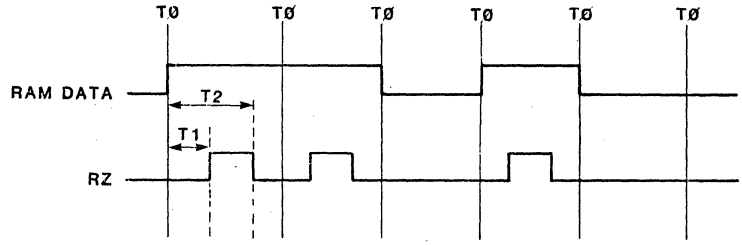


FIGURE 2-3. RZ FORMAT TIMING

2.2.1.2.3 RO FORMAT

Return to one format is used to create a negative going pulse whenever the RAM contains a logic "0". If the RAM contains a logic "1", a pulse will not be generated for that cycle. This is exactly opposite from RZ formatting and is shown in Figure 2-4.

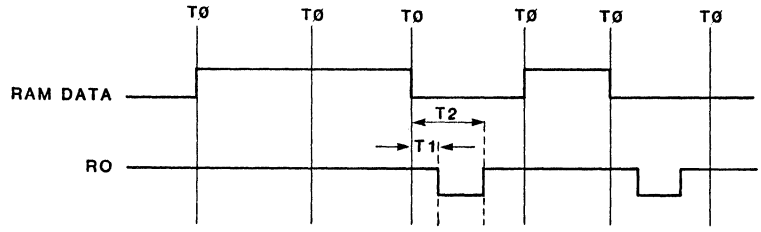


FIGURE 2-4. RO FORMAT TIMING

2.2.2 OUTPUT TIMING

The comparison of RAM data and output data from the DUT is made at a specific point in the cycle relative to T0. Thus, a single timing value is required to define this time which is referred to as strobe time. (See Figure 2-5)

The strobe time should be selected such that all outputs are stable. The point furthest from the event which causes the output to happen is the safest (highest margin) value to use. A point close to the expected value allows a tight guarantee on propagation delay, but has greater possibilities of failure.

When different types of output on a chip have widely different propagation delays, it is sometimes desirable to strobe them at a unique time appropriate to their expected propagation delay values. Dependent on the actual tester used, more than one output strobe may be used. However, it is unlikely that more than two will be available so, at best, outputs can only be separated into two groups from the standpoint of output strobing. Figure 2-5 shows output strobe timing relationships.

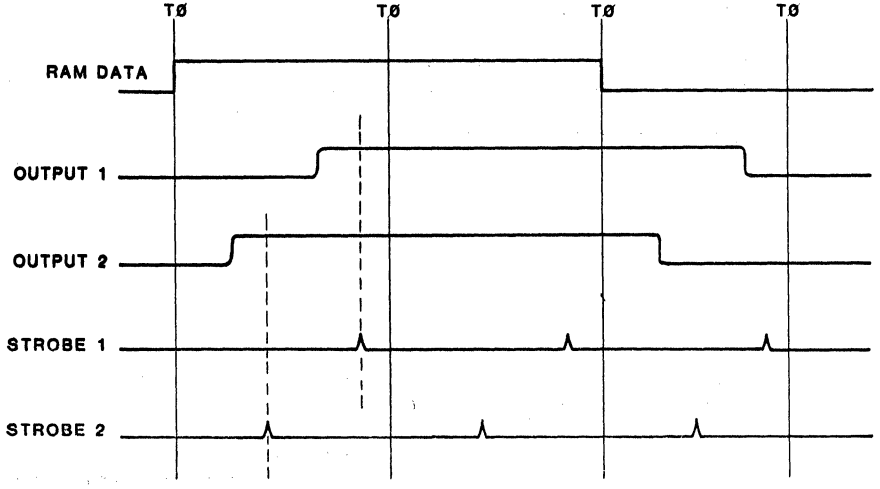


FIGURE 2-5. OUTPUT STROBE TIMING

2.2.3 PIN ELECTRONICS

The pin electronics module contains the drivers, comparators, and control logic associated with each pin. It is located in a "test head" which is physically detached from the tester. The test head interfaces the DUT to the pin electronics via a printed circuit board known as a performance board or load board.

From the user's standpoint, capacitance is the most significant factor regarding this piece of hardware. A value of 50pF is the standard value used as the load capacitance presented to the DUT by the tester.

3 THE TEST LANGUAGE

3.1 OVERVIEW

This section presents information in a tutorial manner. Each command is introduced with examples and diagrams showing typical command usage. An example of a complete chip showing the commands in action in a real chip environment is presented in section 5.

The term RAM data, referred to in the discussion of the tester, can be almost universally related to the simulation data in the ZyPSIM environment. Here's a summary of the basic language features which will allow you to manipulate the simulation data in a similar way to which the tester hardware handles the RAM data.

3.1.1 BASIC LANGUAGE FEATURES

* The data in RAM is essentially the raw simulation data, without timing, and is cycle based. The .TTABLE command provides a means of defining simulation data in terms of cycles rather than nS or uS.

* The .TPERIOD command converts the arbitrary timeless cycles into an absolute value. This period directly equates to the tester hardware clock rate.

* As the raw tester data is modified by a timing generator and data formatter, the simulation data defined by a .TTABLE is modified by a .TGEN command.

* The .TSTROBE command defines when the simulation output data will be sampled, and determines the data that will eventually end up in the tester RAM after conversion by ZyTEST. The value specified in a .TSTROBE command directly equates to the tester hardware strobe.

* The functional test data, direction control and masking control information are combined into a single output file using the .TPRINT command. From this file, ZyTEST is able to create the test vectors (stored in the tester RAM) to exercise the DUT exactly as in the simulation.

3.2 GENERAL INFORMATION

3.2.1 COMMAND CONVENTIONS

As with all other ZyPSIM commands, the test language commands start with a "." period. In addition, they all begin with the letter "T". A tester oriented command will therefore always begin ".T" and can be easily identified. It's that simple.

Many commands take required arguments, some take optional arguments and others do not require arguments at all. Whenever ZyPSIM reads a test language command and detects too many or too few arguments, an appropriate message will be displayed and the rest of the line will be skipped. This allows ZyPSIM to exit from the garbage mode and become intelligible again as soon as possible during network compilation.

3.2.2 CORRELATION TO EXISTING COMMANDS

It will be noticed that some test language commands look almost identical to other ZyPSIM commands, .TPRINT and .PRINT for example. Users familiar with ZyPSIM (if you're reading this you should be, right?) will find this feature helpful. There are differences, however, so be careful. Save yourself some time, don't try and use .TCLK as an equivalent to .CLK, .TCLK does not exist.

3.2.3 SIMULTANEOUS USE OF NORMAL AND TESTER COMMANDS

As the .TPRINT statement mentioned earlier implies, all output generated by test language commands is normally directed to an output file named filename.TPN (or .TPS, .TPF if guardbanding is used). Simultaneous printing of other data (internal node data for example) can be directed to a normal .PRT file. The significance of this will become clear later.

There is generally no restriction in regard to mixing normal and tester commands. However, it's possible to create problems. The tester is essentially a synchronous machine and the test language has been designed so that only synchronous test patterns can be produced. For this reason, the use of .CLK and .TABLE statements is restricted. They should only be used to generate mask and direction control signals in a final netlist.

IMPORTANT Under no circumstances use .CLK or .TABLE to generate input signals which drive device pins.

3.2.4 USING COMMENTS

Comments are important in any source code (yes, the netlist is source code) and their use is strongly recommended. In addition to the "*" (asterisk) which defines a comment line when used as the first character, ZyPSIM II supports the "%" (percent) character to allow in line trailing comments. Any text appearing after a "%" will be ignored.

For example:

```
*
*   THE WHOLE LINE IS A COMMENT
*
I1  IAA1  N1  N2   % HERE'S A COMMENT ABOUT THIS LINE
.
.
. etc
```


3.2.5 MAXIMUM LINE LENGTH

ZyPSIM only reads the first 80 columns of data and any extra characters will be ignored. To continue a logical line (longer than 80 characters) on the next physical line, use the "+" continuation character at the start of the continued line(s).

Most commands have a logical line limit determined by the number of arguments required and the length of each argument, ie: the limit is variable. The .TTABLE statement is an exception to this rule, see the .TTABLE description for details.

3.3 USING THE TEST LANGUAGE

Each command will now be described separately, providing as much detail as necessary to completely explain its function. Refer to the reference section if you just need a concise syntactical description.

3.3.1 .TESTER

The .TESTER command defines the type of tester on which the device will be tested and is used as follows:

a) By ZyPSIM, in conjunction with a set of parameters for the specified tester, to check that hardware dependent timing limits are not exceeded.

b) By ZyPEE, to check that the specified tester can support the hardware features required, eg: number of strobes, number of timing generators etc.

The syntax is as follows:

.TESTER <tester_type>

where <tester_type> may be:

SENTRY7
SENTRY20

For example, using:

.TESTER SENTRY20

will cause parameter and hardware feature checking against SENTRY20 specifications.

If you omit the .TESTER command or supply an incorrect <tester_type>, the default of SENTRY7 will be used.

Some of the other test language commands have arguments whose values have limits which are tester dependent. These limits are summarized in Appendix A.

3.3.2 .TPERIOD

The .TPERIOD command defines the basic rate at which the test will be executed, and hence the rate at which the simulation will be performed. Once defined, each simulation cycle or period assumes the specified value.

The value for .TPERIOD will usually be defined to correspond to the maximum device frequency. This is not always the case, however. Refer to section 4.8 for guidelines.

The syntax is as follows:

.TPERIOD <period>

where <period> is the required test (and simulation) period in nS.

As an example, a device with a 5MHz clock would normally have a .TPERIOD of 200nS.

.TPERIOD 200

The minimum .TPERIOD values are:

SENTRY7	100nS
SENTRY20	50nS

There are certain operating conditions which might increase these values (reduce the maximum test rate) and they are described in section 4.7.

3.3.3 .TTABLE

The .TTABLE command operates similarly to the .TABLE command except for several very important differences which will become apparent in this section. The .TTABLE command (in conjunction with the .TGEN command) is the only means of generating tester compatible input patterns. The .TTABLE command emulates the vector capabilities of the tester RAM and supports vector repetition plus subroutine definition and calls.

The basic syntax is as follows:

.TTABLE <node 1>[,<iv 1>] <node 2>[,<iv 2>] . <node n>[,<iv n>]

where <node 1> through <node n> are the nodenames which the .TTABLE data will drive and <iv 1> through <iv n> are optional initialization values for NRZ mode signals.

3.3.3.1 Data

Following the .TTABLE statement defining the nodes to be driven, the node data must be specified. Data may appear as an explicit vector or as a call to a previously defined subroutine. More on subroutines later.

The syntax of a vector definition is as follows:

<count> <value 1> <value 2> ..... <value n>

where <count> specifies the number of times the vector will be applied, and <value 1> through <value n> specify the logic values for each corresponding node in the .TTABLE.

For example:

```
.TTABLE  A   B   C
   5     0   0   0
  100    1   1   0
```

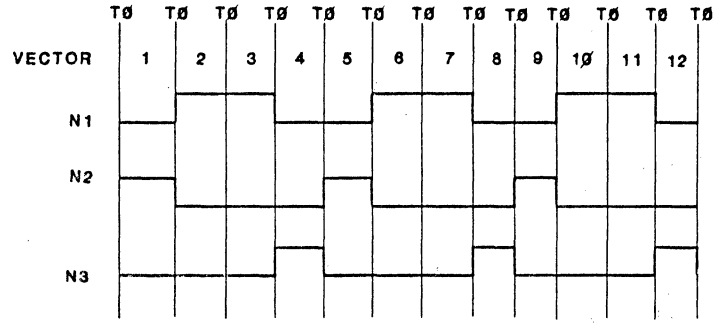
Nodes A, B and C will receive pattern "0 0 0" for 5 cycles, followed by pattern "1 1 0" for 100 cycles. Three important differences from the .TABLE command should be noted.

a) DC initialization is not achieved through the use of an explicit time zero vector. A discussion of DC initialization follows.

b) Data is not specified in absolute nS, but in terms of cycles or tester periods. A tester period is equal to a simulation period, which is defined by the .TPERIOD command.

c) Cycles are referred to in relative terms not in absolute cycle numbers. In the above example, 5 means the first 5 cycles, 100 means the next 100 cycles.

The following example will clarify this concept. Suppose you want to drive three nodes N1, N2 and N3 with the data shown in Figure 3-1.



This would be achieved by using the following:

```
.TTABLE   N1   N2   N3
    1      0    1    0
    2      1    0    0
    1      0    0    1
    1      0    1    0
    2      1    0    0
    1      0    0    1
    1      0    1    0
    2      1    0    0
    1      0    0    1
```

FIGURE 3-1. EXAMPLE .TTABLE DATA

The first field specifies the number of times a vector (pattern) is to be repeated. Subsequent fields correspond to the supplied nodenames and define the node value. Note the use of blanks to line up the data for readability.

3.3.3.2 Basic Rules

The following rules and features apply to .TTABLEs.

a) The .TTABLE command may be continued on as many physical lines as necessary to define all the nodes in the .TTABLE.

b) The .TTABLE data statements must be completely defined on one physical line, continuation data statements are not permitted.

c) All .TGEN's associated with signals in a .TTABLE must precede the .TTABLE.

d) Only "1" and "0" are valid .TTABLE data values, since the tester can only produce "strong" inputs.

e) Blank spaces may be used between 0/1 data values to aid readability, but they are not required.

f) The absolute maximum number of nodes in a .TTABLE is 78 and can only be achieved when the <count> is less than 10 and blanks are not used within the data fields. In practice then, the limit will be somewhat less.

g) Blank lines, "*" comment lines, and "%" trailing comments are permitted.

h) Termination of a .TTABLE occurs when a non .TTABLE related statement is processed.

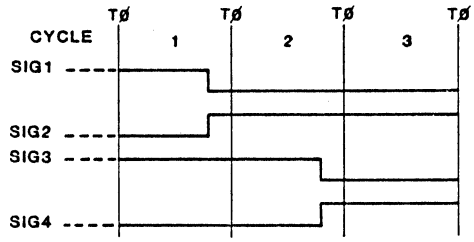
i) Optional DC initialization may be supplied for NRZ mode signals.

3.3.3.3 DC Initialization

The need to provide DC initialization results from the way in which NRZ mode signals are processed. In NRZ mode, a transition is only produced when the vector data value for the current cycle is different from the previous cycle. Since a previous cycle does not exist for the first vector in a .TTABLE, an alternative means of providing this information (when required) is used.

On the other hand, RZ and RO mode signals do not require any initialization due to their basic characteristics (eg: RZ starts at "0" transitions to "1" and then returns to "0"). Thus, the data is completely defined within each cycle without regard for the previous cycle.

The following example illustrates the effect of DC initialization.



.TTABLE	SIG1,1	SIG2	SIG3,1	SIG4
1	0	1	1	0
2	0	1	0	1

Rules for NRZ mode initialization:

a) If there is a transition required in the first cycle (vector 1), the initialization value should be the opposite value to the .TTABLE vector 1 value.

b) If there is not a transition during vector 1, the initialization value should be the same as vector 1.

c) The default initialization value is 0 which may also be explicitly specified (eg: SIG2,0) if desired.

d) Specifying initialization data for RO and RZ signals is not permitted.

Initialization data is used to:

a) Initialize the network and precondition the NRZ mode signals to provide correct operation during vector 1. Note: This "DC" initialization does not guarantee that all nodes are valid "1" or "0", only that the network is stable.

b) Initialize the tester to produce corresponding conditions when the device is tested. This is achieved by printing an initialization vector to the output file as "cycle 0".

NOTE: An initialization vector is always printed to the output file, regardless of whether initialization data is supplied or not. NRZ mode signals default to "0", RZ is always "0", RO is always "1" and outputs are always "m" (masked "0").

3.3.3.4 Subroutines

In defining .TTABLE data, it is highly desirable to be able to repeat data already defined. Our previous example in fact uses three explicit definitions (two repeats) of the same pattern. (What a coincidence!) Let's take a look at how it can be compacted.

The first thing to do is tell ZyPSIM that you are defining a subroutine and give it a name. Subroutines are defined using a left parenthesis and a label as follows:

(<label_name>

where <label_name> is any user defined name. Names may be up to 16 characters in length and may contain any alphanumeric character. Note that spaces are not allowed between the opening parenthesis and the <label_name> definition. The definition must be the only contents of the line. Failure to observe these rules will cause every eleventh production part to have an intermittent fault on days with an R in the month.

Having defined the subroutine, next you need to supply the data. Normal vector data statements are used to do this. Using the example of Figure 3-1, the routine starts to take shape as follows:

```
(LABEL3
    1       0    1    0
    2       1    0    0
    1       0    0    1
```

Finally, you need to terminate the subroutine definition. In its simplest form this consists of a right parenthesis, which must also appear as the only item on the line.

ZyPSIM also allows you to decide whether the definition is simply that - a definition, or whether it will be executed as well as defined - at the time of definition.

This is easily achieved when the subroutine is terminated by supplying an optional argument. The general syntax for subroutine termination is as follows:

)[<*n>]

where n is a multiplier for the number of times to execute the subroutine. To suppress execution, just use a multiplier of 0. If only a closing parenthesis is used, an implied "*1" will be used which results in the default of one execution. Using "*" without a multiplier will cause an infinite repeat until the end of the simulation.

The termination options can be summarized as follows.

```
)                    equivalent to *1
)*0                  defined only, not executed.
)*n                  defined and executed n times
)*                   defined and indefinitely executed
```

Returning to our example, it's appropriate to execute the routine three times concurrent with the definition. The final .TTABLE would appear as follows:

```
.TTABLE    N1   N2   N3
(LABEL3
    1       0    1    0
    2       1    0    0
    1       0    0    1
)*3
```

If you now want to use this block of data again, elsewhere in your pattern definition, you can simply refer to it by name and specify the desired repeat count. The syntax to call a previously defined subroutine is as follows:

```
(<label_name>)*n
```

where <label_name> and *n have the same meaning as for a subroutine definition. Note that placing such a statement within a .TTABLE is an implied call to the previously defined <label_name> and means call <label_name> n times.

The data statements defined for <label_name> will be inserted "n" times at the location in the .TTABLE where the call statement is placed. The "call" must be the only statement on the line and subroutines must be defined before they are used. A composite .TTABLE encompassing our example routine might be as follows:

```
.TTABLE    N1   N2   N3
    2       0    0    0
    5       1    0    1
(LABEL3
    1       0    1    0
    2       1    0    0
    1       0    0    1
)*3
   10       0    1    1
    3       0    0    0
(LABEL3)*5
   15       1    1    1
(LABEL3)*20
    5       1    1    1
.
.
etc
```

3.3.3.5 Subroutine Nesting

Subroutines may be nested to any practical level. Nesting is permitted at either the definition or usage stage. For example:

```
*
* NESTING EXAMPLE #1
*
.TTABLE   N1   N2   N3
(LEVEL1
   1       0    0    0
(LEVEL2
   2       1    0    1
   3       1    1    0
)*2
   3       1    1    1
)*4
```

In this case, nesting has occurred at the definition stage. This means that a new routine may be defined inside a routine which itself is being defined, and so on.

```
*
* NESTING EXAMPLE #2
*
.TTABLE   N1   N2   N3
(LEVEL2
   2       1    0    1
   3       1    1    0
)*0
(LEVEL1
   1       0    0    0
(LEVEL2)*2
   3       1    1    1
)*4
```

Here nesting has occurred at the usage stage, since the LEVEL2 routine was predefined. Both of the above examples will produce the same result. You will find nested subroutines helpful in producing complex patterns with a minimum of effort. Let's hear it for nested subroutines!

3.3.3.6 Other Properties

Label names are not global for all .TTABLEs and you may safely use the same name in different .TTABLEs. They will be unique to the .TTABLE in which they are defined. If you need a similar pattern in more than one .TTABLE, you'll have to copy it so that it's defined in each routine. However, it's unlikely this would arise, since different .TTABLES will not be talking to the same nodes.

That brings up another rule for .TTABLEs. A node name can not appear in more than one .TTABLE statement. Multiple .TTABLE statements are permitted, of course, and the first vector in each one will start at cycle one. All .TTABLEs will execute their vectors concurrently. If any .TTABLE runs out of vectors before .MAXTIME is reached, a warning message will be printed and the last vector in the offending .TTABLE will be repeated until .MAXTIME is reached.

3.3.3.7 USING MULTIPLE .TTABLES

Using multiple .TTABLEs can significantly reduce the amount of data statements required to define a specific pattern. Apart from reducing the typing effort required (not an engineer's favorite task), it also reduces the chance of errors and uses a smaller amount of the total storage allocated for .TTABLE data.

As an example, let's create a binary input for an eight bit word for all combinations. If a single .TTABLE were used, 256 vectors would have to be explicitly defined as follows:

```
*
*       EXAMPLE .TTABLE FOR EXPLICIT 8 BIT BINARY INPUT
*
.TTABLE     B7  B6  B5  B4  B3  B2  B1  B0

   1         0   0   0   0   0   0   0   0
   1         0   0   0   0   0   0   0   1
   1         0   0   0   0   0   0   1   0
   1         0   0   0   0   0   0   1   1
.
.etc
.
   1         1   1   1   1   1   1   1   1
```

That's a lot like hard work!

Here's a better way using multiple .TTABLEs for data compression.

```
*
*       EXAMPLE 8 BIT BINARY INPUT USING MULTIPLE .TTABLEs
*
.TTABLE     B7
 128         0
 128         1
*
.TTABLE     B6
(L1
  64         0
  64         1
)*2
*
.TTABLE     B5
(L2
  32         0
  32         1
)*4
*
.TTABLE     B4
(L3
  16         0
  16         1
)*8
*
.TTABLE     B3
(L4
   8         0
   8         1
)*16
*
.TTABLE     B2   B1   B0
(L5
   1         0    0    0
   1         0    0    1
   1         0    1    0
   1         0    1    1
   1         1    0    0
   1         1    0    1
   1         1    1    0
   1         1    1    1
)*32
*
```

This method reduces the number of statements required from 257 to 34, a worthwhile saving.

However, true to life, there's no such thing as a free lunch. Since nodenames cannot appear in more than one .TTABLE, we have lost the ability to talk to the eight bit word as a group. For example, to "reset" the word, the following statement would normally be used:

```
.TTABLE    B7  B6  B5  B4  B3  B2  B1  B0

   1        0   0   0   0   0   0   0   0
```

Having already defined nodes B0-B7 in six separate .TTABLEs, we must also use them to define our reset condition. If this was a onetime occurrence, it would be somewhat inconvenient. However, if it will be used often, we would define it as a subroutine and nothing is really lost.

The bottom line for you is: weigh carefully the use of multiple versus single .TTABLEs. Dependent on your pattern requirements, single may be the best choice. A few moments of pleasure now (reduced typing!) may have to be paid for later in the pattern.

3.3.3.8 Summary

By now the advantages of using the language versus previous methods should be apparent, and we've really only looked at two commands! By using cycles instead of real time (.TPERIOD takes care of that), you're free to define vectors without worrying about every last nS.

The .TTABLE command works in relative cycles and generates vector data statements with repeat counts. Add to that the subroutine capability and we have the makings of a very powerful tool.

However, there's still something missing. We now have cycle based data which can only change at the rate defined by .TPERIOD. This is exactly how data is stored in the tester RAM. We now need the ability to format the data according to the requirements of our device. Let's find out how that's done.

3.3.4 .TGEN

In the same way that data in the tester RAM is modified by timing generators and data formatting before application to the DUT, the raw data provided by a .TTABLE is modified by a .TGEN command before application to the network inputs.

The .TGEN statement provides the capability to specify timing and formatting. The basic syntax is as follows:

.TGEN <mode> <T1> [<T2>] <nodelist>

where the arguments are defined as follows.

3.3.4.1 Mode

<mode> specifies the formatting mode and defines how the raw .TTABLE data will be modified to provide the simulation inputs. Valid modes are listed below.

MODE	DESCRIPTION
NRZ	Non-return to zero
RZ	Return to zero
RO	Return to one
IONRZ	Non-return to zero, signal is bidirectional
IORZ	Return to zero, signal is bidirectional
IORO	Return to one, signal is bidirectional

The three basic modes (NRZ, RZ, RO) correspond exactly to the format descriptions in section 2, Figures 2-2, 2-3 and 2-4. The additional modes (prefixed by IO) are used whenever the associated pin is an I/O pin. Formatting in the input mode will be identical to NRZ, RZ, and RO.

Some examples of usage will be presented later in this section. In addition, section 4 provides some guidelines for formatting mode selection.

NRZ TO mode is unique in that it does not require timing information. The data is unformatted and is essentially raw .TTABLE data. This mode is described in "special case" section 3.4.

3.3.4.2 Timing

<T1> and <T2> define the times (in nS) within the cycle (with respect to T0) when data changes will occur. <T1> is required for all <mode> specifications. <T2> is only required when RZ or RO (IORZ, IORO) is specified for <mode>.

Figure 3-2 illustrates how the timing values relate to the formatting mode and .TTABLE data.

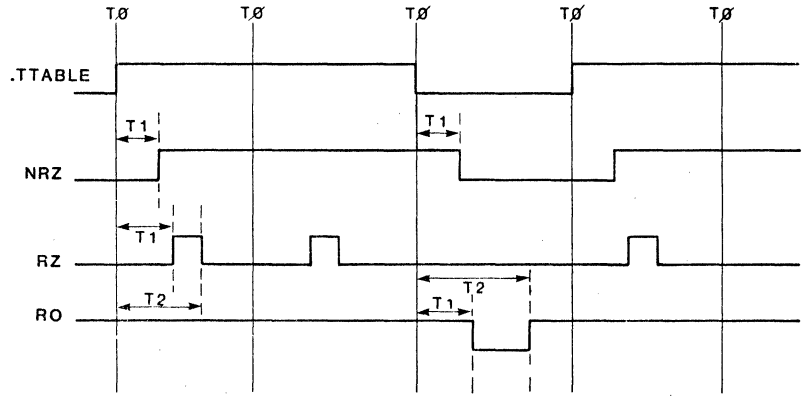


FIGURE 3-2. .TGEN AND .TTABLE RELATIONSHIPS

Note that both values are with respect to T0 so that pulse width is effectively T2 - T1 when using RZ or RO modes.

There is a tester dependent minimum value of 10nS which may be specified for T1 and T2. Although the tester "thinks" that the pulse is negative or positive going, dependent on mode selection, your device can interpret the data differently. This can be used to overcome the minimum timing value should this ever be a problem. Section 4.8 provides more details on this topic.

3.3.4.3 Nodelist

The <nodelist> portion of the .TGEN statement defines the nodes to which the formatting and timing apply. The form of <nodelist> is as follows:

<node 1>[,<dir 1>] <node 2>[,<dir 2>] ... <node n>[,<dir n>]

where <node 1> through <node n> are the nodes to be driven. They must also appear in a .TTABLE statement.

<dir 1> through <dir n> are optional nodenames which are only required when the mode is specified as IONRZ, IORZ or IORO. Whenever an I/O mode is specified, a controlling node associated with each data node must also be specified. These are the <dir> nodes which are used to define the direction of the associated signal.

A value of "1" for <dir> defines the signal as an input and the .TTABLE data for that node, formatted according to the .TGEN mode and timing, will drive the node. A value of "0" for <dir> effectively disables the .TTABLE/.TGEN from the node and data is now expected to be supplied by the device. When <dir> is "0", the node is treated by the .TPRINT statement as an output signal. More on .TPRINT later.

The <dir> signals must be user defined to change whenever the associated device pad cell signal is switched from input to output. You may use .CLK, .TABLE, .TTABLE, ZyPSIM primitives or regular chip signals for this purpose. The internal chip signal which causes the output to change direction is a possible candidate.

NOTE: Refer to section 3.4 (special cases) for more information regarding direction signals.

3.3.4.4 Using The .TGEN Statement

This all sounds complicated but it's really quite straightforward. Suppose we want to create the input shown in Figure 3-3 for signal BUS0.

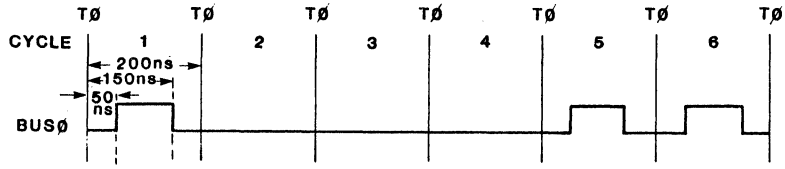


FIGURE 3-3. EXAMPLE .TGEN INPUT SIGNAL

We'll use the following three step approach.

a) First we would define the cycle time using .TPERIOD.

b) Next we need a .TGEN statement to define timing and formatting. .TGEN commands must precede any associated .TTABLEs.

c) Finally, .TTABLE data is required to say "1" for 1 cycle, "0" for 3 cycles and "1" for 2 cycles.

By inspection, the formatting is obviously RZ (right?). The following statements would be used to define BUS0.

```
.TPERIOD 200
*
.TGEN  RZ  50  150  BUS0
*
.TTABLE  BUS0
   1      1
   3      0
   2      1
```

Note that although we require a pulse width of 100nS, T2 is specified as 150 (since it's specified with respect to T0) to obtain T2-T1 = 100nS.

That was quite painless wasn't it? Let's make it a bit more complicated.

Assume that BUS0 becomes an output in cycle 3 of Figure 3-3. The circuit of Figure 3-4 would be a typical configuration for an I/O pin. Signals in parentheses refer to standard syntax for cell PCC1.

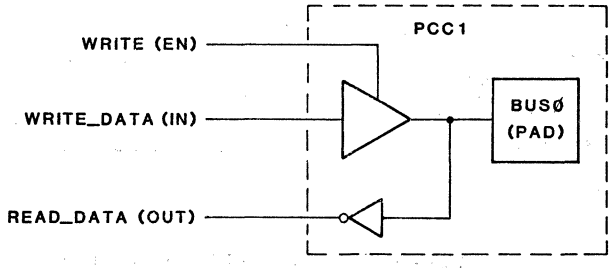


FIGURE 3-4. TYPICAL I/O CONFIGURATION

Referring to Figure 3-5, we can use the signal WRITE to control the direction of signal BUS0 in the .TGEN statement. However, most ZyP library I/O cells use an enable signal polarity of "1" to enable the cell for output, whereas <dir> needs "0". In such cases, create another direction signal with the complement data and drive it from a .TTABLE statment only.

NOTE: In this simple illustration, "single cycle" I/O has been used. This can lead to problems - see section 4.7 for more on this topic.

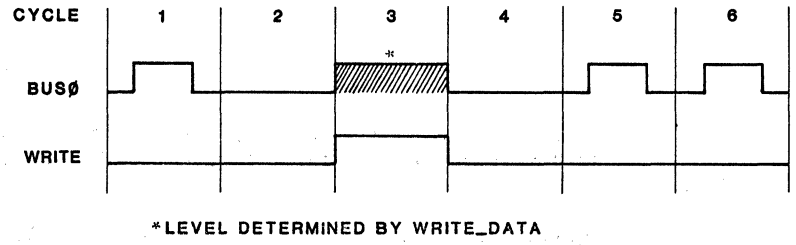


FIGURE 3-5. I/O DIRECTION CONTROL

The .TGEN statement now becomes:

```
.TGEN  IORZ  50  150  BUSO,NWRITE
```

IMPORTANT The <dir> node is sampled at T0 time to determine the direction of the data signal. In our example, signal NWRITE was synchronous with T0 and was therefore able to be used. In practice, an enable signal to an output cell might not change until after T0. In such a case, a user defined signal must be created to control direction, unless the enabling signal is available at some earlier point in the logic and occurs prior to or concurrent with T0.

NOTE: Refer to section 3.4 (special cases) for more information regarding generation of masking signals.

3.3.4.5 Direction and Mask Register Data

Whenever an IO mode is specified in a .TGEN statement, ZyPSIM will automatically generate the correct conditions for the mask and direction registers. For a <dir> node level of "1", the data signal is an input. Therefore, the direction bit will also be "1" (=input) and the mask bit for that pin will be "0" (=don't test). The composite output of functional data, direction register data and mask register data is available in the .TPN, .TPS, or .TPF file.

3.3.5 .TSTROBE

Just as the tester hardware strobe determines the time of RAM/DUT comparison, the .TSTROBE command determines the time when output signals in the .TPRINT list (defined later) will be sampled and printed to the output file. The syntax is as follows:

.TSTROBE <strobe_time> <nodelist>

where the arguments are defined as follows:

3.3.5.1 Strobe Time

<strobe_time> is specified in nanoseconds and defines the time within the cycle when output signals are sampled. The same value will also be used for the tester hardware strobe position. Section 4.8.5 describes some restrictions regarding strobe position.

Although ZyPSIM can take a "snapshot" of the simulation outputs at <strobe_time>, the tester hardware strobe has a finite minimum width of 10nS. The tester must see valid data for the duration of the strobe width, since the strobe is used to gate the comparison, not latch it.

Two issues arise from this.

a) In the simulation, <strobe_time> is aligned with the leading edge of the hardware strobe.

b) Data must be valid for at least 10nS after <strobe_time>.

ZyPSIM will check that outputs do not change state more than once per cycle to ensure that all is well. Figure 3-6 shows the simulation/hardware strobe relationship.

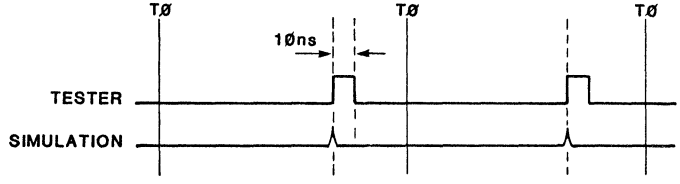


FIGURE 3-6. SIMULATION VERSUS HARDWARE STROBE RELATIONSHIP

3.3.5.2 Nodelist

The <nodelist> portion of the .TSTROBE statement defines the outputs which will be sampled at <strobe_time>. The form of nodelist is as follows.

<node 1>[,<mask1>] <node 2>[,<mask2>] ..[<node n>[,<maskn>]

where <node 1> through <node n> are the nodes to which <strobe_time> applies. The specified nodes can only be output or I/O signals. Input signals are illegal in a .TSTROBE statement.

The optional [,<mask>] argument supplies information to control the mask register. You may use a nodename driven by .CLK, .TABLE, .TTABLE, ZyPSIM primitives or regular chip signals as a <mask> argument. The <mask> nodename argument must immediately follow the comma. If the value is "1", the output will be tested. A value of "0" disables testing. If <mask> is omitted, a default value of "1" will be used and the output will be tested.

Whenever a device output is in an unknown state, and you have determined that this is expected and acceptable (such as during initialization and cycle slips), you should use a <mask> argument which has a "0" value for the periods of uncertainty. The value of <mask> is sampled at T0 time so be sure to use an appropriate signal.

NOTE: You do not have to supply masking information for I/O pins to disable testing during the time when the pin is an input. The <dir> argument in the .TGEN statement takes care of this.

3.3.5.3 Mask Register Data

Composite masking is thus a combined function of user supplied inputs and can be summarized as follows:

A signal defined in a .TSTROBE statement will always be tested unless:

a) A user defined <mask> nodename associated with the signal has a value of "0".

b) The signal is an I/O signal and the <dir> nodename supplied in the associated .TGEN statement has a value of "1", ie: the pin is switched to an input.

NOTE: ZyTEST is used to detect problems due to unmasked outputs and may also be used to generate ZyPSIM compatible masking data.

3.3.5.4 Multiple Strobes

More than one strobe can be used (dependent on tester type) to optimize the test. Outputs with different propagation delays can be separated into groups, each group defined in a .TSTROBE with an appropriate <strobe_time>.

3.3.5.5 Input Strobing

.TSTROBE applies only to outputs. Inputs are automatically printed to the .TPN, .TPS or .TPF file at a time when they are at the correct logic level. This will depend on the .TTABLE data and the timing and formatting used. The value is sampled and printed at time <T1> specified in the associated .TGEN statement.

Thus you can select <strobe_time>(s) at the best time for your device outputs without regard for the input conditions.

3.3.5.6 Example .TSTROBE Statements

```
.TPERIOD 500
.TGEN IONRZ 250 LATE,LATE_DIR LATER_STILL,LATER_STILL_DIR
.TSTROBE 450 LATE,LATE_MASK LATER_STILL
.TSTROBE 200 EARLY,EARLY_MASK
```

Signal LATE has a long propagation delay and is strobed at time 450. It will not be tested when direction signal LATE_DIR is high or when masking signal LATE_MASK is low.

Signal LATER_STILL is also strobed at time 450, but is masked only when the direction signal LATER_STILL_DIR is high.

Signal EARLY which is not an I/O, simply an output, has a short propagation delay and is strobed at time 200. It will be tested at all times except when masking signal EARLY_MASK is low.

3.3.6 .TPRINT

The .TPRINT command defines which signals are to be printed to the output file, similiar to the .PRINT which produces a filename.PRT file. A .TPRINT statement produces a filename.TPS, filename.TPF or filename.TPN depending on the process condition; SLOW, FAST, or NOMINAL, as selected with the ZyPSIM .GUARDBAND statement. These files are referred to collectively as .TPx throughout this document. The syntax is:

.TPRINT <node 1> <node 2> ... <node n>

where <node 1> through <node n> are the nodenames to be printed.

3.3.6.1 Requirements and Restrictions

The nodes should correspond to the device pinout and should be arranged in pin number order, 1 through n, where n is the number of pins on your chip. The list should also include VDD and VSS.

NOTE: All nodes in a .TPRINT statement (with the exception of VDD, VSS and UNUSED) must be defined in a .TSTROBE or a .TGEN statement. I/O pins must be defined in both.

So far the .TPRINT statement is compatible with the .PRINT statement. However, there are three major differences:

a) Only one .TPRINT is allowed in the netlist.

b) Output is not displayed at the terminal. (Maintains .PRINT compatibility).

c) The "/" formatting character is not permitted.

3.3.6.2 Using .TPRINT and .PRINT Simultaneously

If you want to print internal nodes of the device (not specified in .TGEN or .TSTROBE), use the regular .PRINT statement as well as .TPRINT. Both may be used at the same time.

Another reason to use .PRINT would be to determine actual propagation delays of the outputs. Since outputs in the .TPRINT file are printed only at .TSTROBE times, the absolute time value of output changes cannot be determined from the .TPx files. During logic design and development, you will (better) be interested in absolute timing data.

3.3.6.3 Examples

```
.TPRINT OUT_SIG1 OUT_SIG2 VDD IN_SIG3 VSS UNUSED IN_SIG4
.POC
.PRINT  OUT_SIG1 OUT_SIG2 INT_NODE3
```

In this example, four device signals are being printed. Output signals OUT_SIG1 and OUT_SIG2 must also appear in a .TSTROBE command. Input signals IN_SIG3 and IN_SIG4 must also appear in .TGEN and .TTABLE commands.

The signals VDD, VSS and UNUSED are reserved for the special purposes implied by their meaning. Special case section 3.4 deals with the topics of power supply pins and unused connections.

The .PRINT statement is being used to display the two output signals using .POC to obtain timing information in absolute nS. An internal node is also being printed.

3.3.6.4 Output Symbols

A .TPx file contains a combination of the device input and output pin states (functional test data) plus direction and masking control data. Prior to the release of ZyPSIM II, direction and mask data was contained in separate .DTN and .MSK files.

Rather than increase the amount of data in a .TPx file, a set of symbols were chosen which allow the state of the output to be indicated at the same time as direction and mask information. The table below shows how this has been implemented.

STANDARD .PRT SYMBOL	DIRECTION CONTROL DATA	MASK CONTROL DATA	.TPx SYMBOL
1	0	1	1
	0	0	M
0	0	1	0
	0	0	m
H	0	1	H
	0	0	W
L	0	1	L
	0	0	w
+	0	1	+
	0	0	T
-	0	1	-
	0	0	t
Z	0	1	Z
	0	0	z
X	0	1	X
	0	0	x
*	0	1	*
	0	0	i
.	0	1	.
	0	0	S
'	0	1	'
	0	0	s
~	0	1	~
	0	0	?
1	1	1	D
	1	0	D
0	1	1	d
	1	0	d

FIGURE 3.7 TABLE OF .TPx FILE SYMBOLS

3.3.6.5 Explanation of Symbols

Note that for output signals (direction control "0") the symbols are identical to .PRINT symbols when the outputs are unmasked. The alternate output symbol set becomes active whenever the output is masked, and has some relation to the original symbol. For example:

When strong 1/0 is masked, M/m (strong masked) is used.

When weak 1/0 = H/L is masked, W/w (weak masked) is used.

When tristate 1/0 = +/- is masked, T/t (tristate masked) is used.

For input signals, masking does not apply. For I/O signals, the <dir> signal causes automatic masking and any user defined masking is overridden. Since the tester can only produce strong signals, only 1 and 0 need be represented. This is achieved using D/d (driving) symbols.

Although the symbols in the .TPx files may not be as easily understood at first glance as the .PRT symbols, they eliminate the need for separate .DTN and .MSK files. This drastically reduces the effort needed to produce tester compatible ZyPSIM output.

NOTE: Since producing tester compatible patterns (using .DTN and .MSK files coupled with .CLK's) is quite challenging, it's well worth the trade off.

In addition, conversion to tester format via ZyTEST and graphical waveform display are easily accommodated. Since the output can also be simultaneously printed to a .PRT file, you can have the best of both worlds.

3.3.6.6 Example Output

The following netlist and .TPN output listing show masked and unmasked output symbols for several logic states.

```
 EXAMPLE OUTPUT
*
* NET LIST TO DEMONSTRATE OUTPUT SYMBOLS
*
*
E1  PBK1     IN1  ENABLE   OUT1
E2  PBK1     IN2  ENABLE   OUT2
E3  PBK1     N1   ENABLE   OUT3
E4  POA1     IN1  DUM1     OUT4
E5  POA1     IN2  DUM2     OUT5
E6  POA1     N1   DUM3     OUT6
E7  IAA1     IN1  N1
E8  IAA1     IN2  N1       % CREATE AN "X"
*
.CLK0 IN1
.CLK1 IN2
*
.TPERIOD=200
.TGEN NRZ 1 ENABLE
*
.TTABLE ENABLE,1 MASK
(SUB
   4      1     0
   4      1     1
   4      0     0
   4      0     1
)*2
*
.TSTROBE 150 OUT1,MASK OUT2,MASK OUT3,MASK OUT4,MASK
+            OUT5,MASK OUT6,MASK
*
.TPRINT ENABLE OUT1 OUT2 OUT3 OUT4 OUT5 OUT6
*
.MAXTIME=6400
.GO
.END
```

1 EXAMPLE OUTPUT

```
        EOOOOOO
        NUUUUUU
        ATTTTTT
        B123456
        L
        E

CYCLE

00000 Dmmmmmm
00001 DmMiWwi
00002 DmMiWwi
00003 DmMiWwi
00004 DmMiWwi
00005 D01*HL*
00006 D01*HL*
00007 D01*HL*
00008 D01*HL*
00009 dtTzWwi
00010 dtTzWwi
00011 dtTzWwi
00012 dtTzWwi
00013 d-+ZHL*
00014 d-+ZHL*
00015 d-+ZHL*
00016 d-+ZHL*
00017 DmMiWwi
00018 DmMiWwi
00019 DmMiWwi
00020 DmMiWwi
00021 D01*HL*
00022 D01*HL*
00023 D01*HL*
00024 D01*HL*
00025 dtTzWwi
00026 dtTzWwi
00027 dtTzWwi
00028 dtTzWwi
00029 d-+ZHL*
00030 d-+ZHL*
00031 d-+ZHL*
00032 d-+ZHL*
```

3.3.7 .TPOC

The .TPOC command operates in a similar manner to the .POC (print on change) command. The syntax is:

.TPOC

Arguments are not permitted.

When the .TPOC command is used, output will only be printed to a .TPx file when any of the signals in the .TPRINT node list change state. This will be found useful in minimizing output which would otherwise be excessive. If your device input/output data is constant for many cycles, .TPOC will be particularly effective.

NOTE: .TPOC is effectively the same as the first level compression feature of ZyTEST, ie: consolidation of identical consecutive vectors.

The output listing below was generated using the same netlist as the example output in the previous section, this time with the addition of the .TPOC command.

```
1   EXAMPLE OUTPUT


        EOOOOOO
        NUUUUUU
        ATTTTTT
        B123456
        L
        E


  CYCLE


  00000 Dmmmmmm
  00001 DmMiWwi
  00005 D01*HL*
  00009 dtTzWwi
  00013 d-+ZHL*
  00017 DmMiWwi
  00021 D01*HL*
  00025 dtTzWwi
  00029 d-+ZHL*
```

3.4 SPECIAL CASES

Now that each command has been covered, it's time to backtrack and take care of some special cases. They are grouped in this section to specifically draw your attention to them. You will definitely want to take advantage of the features offered by these special cases so be sure to read and understand this section.

3.4.1 HOW TO OBTAIN NRZ T0 MODE

Remember this mode? NRZ T0 mode allows vector data to be used without formatting or timing. Data transitions occur synchronously with T0, hence the name. The language is so designed that raw .TTABLE data can not be used except with a .TGEN. Since the tester does not use a timing generator for this mode (data is passed from RAM to DUT unmodified), there is a minor problem (special case!).

To select NRZ T0 mode, a .TGEN statement must be used. Eg:

```
.+.TGEN  NRZ  1   NRZ_MODE_SIGNAL
```

will cause NRZ_MODE_SIGNAL to be driven directly from the .TTABLE data for that node.

Note that a value of 1 is specified for <T1> in the .TGEN statement. Although the minimum value of a timing generator has been violated, the value of 1 in this case is the "special case" and is used to force NRZ T0 mode. Values greater than 1 but less than the minimum value for <T1> and <tester_type> will cause an error message. Furthermore, a value of 1 is only permitted for NRZ and IONRZ modes.

NOTE: Don't worry, a .TGEN used for NRZ T0 mode is not mapped to a physical timing generator and therefore doesn't count in determining the total number of timing generators used.

3.4.2 CREATING MASK AND DIRECTION SIGNALS

The .TSTROBE and .TGEN statements sample the <mask> and <dir> argument signals at T0 time. In many cases, suitable internal chip nodes may not be available and you will need to generate these signals separately. Although .CLK or .TABLE may be used, the most convenient way of doing this is the .TTABLE, since the data is cycle based.

Normally, .TTABLEs can't be used (without a .TGEN) to drive a node directly. However, handling of mask and direction data is given "special case" treatment. Driving <mask> or <dir> arguments of .TSTROBE and .TGEN statements directly from a .TTABLE will not cause an error to be flagged. On the contrary, it is illegal to drive a <mask> or <dir> argument with a signal associated with a .TGEN.

3.4.3 WATCH OUT THERE'S AN I/O ABOUT

When a signal is defined as an I/O, you still need to take care of the .TTABLE data, even during the time when the signal is an output. For example, suppose that you drive an I/O pin from a .TTABLE for several vectors, and then switch the pin into the output mode. ZyPSIM will continue to read vector data from the associated .TTABLE during the time that the pin is an output. Since other data in the .TTABLE may be required at the same time, it's easy to understand why. Two conditions are possible when this situation arises.

a) When the I/O pin is defined in the same .TTABLE as other data, the node data field for the I/O mode must have a value assigned during the time it is an output. The actual value is unimportant, but the syntax requirements of the .TTABLE (n nodes specified must have n data fields) must be satisfied.

For example:

```
.TTABLE INPUT_NODE I-O_NODE

   100      0         1   % I-O_NODE is an input
   200      1         1   % I-O_NODE is an output
   100      1         0   % I-O_NODE is an input
```

Even though I-O_NODE is an output during the execution of the 200 vectors in the second data statement, an arbitrary value (in this case 1) has been included.

b) When the I/O node is the only node in a .TTABLE, a dummy vector must be used during the time when the I/O pin is an output. The vector <count> should be equal to the number of cycles for which the signal is an output. Data value can be either "0" or "1". Failure to do this will result in incorrect vector data when the pin returns to the input mode, and the .TTABLE will eventually run out of vectors before .MAXTIME is reached. At that point the keys on your terminal will melt.

For example:

```
.TTABLE I-O_NODE

   100      1   % I-O_NODE is an input
   200      1   % I-O_NODE is an output, this is a dummy
   100      0   % I-O_NODE is an input
```

3.4.4 DEALING WITH POWER SUPPLY CONNECTIONS

Power supply connections represent another special case. Although they play no part in simulation, it is extremely important that they are included in the net list for the device. The netlist is the common "data base" for logic simulation, placement and routing, and test program generation. The place and route software (ZyPAR) and the tester hardware treat power supply pins in a special way.

Power supply information must be included in the netlist as follows:

a) The .PADS statement must contain the special reserved keywords VDD and VSS. If more than one power and/or ground pins are required then VDD1, VDD2, VSS1, VSS2 etc. should be used. As for any other signal, pin numbers must be specified for VDD and VSS. The .PADS statement is processed by ZyPAR and ZyPEE.

b) Power supply pad cells must be included in the netlist in much the same way as any other pad cell. They are treated as input pads from the point of view of ZyPSIM. The library contains standard cells for this purpose and they will normally be PDC1 for VDD and PDD1 for VSS. Data sheets exist for these cells.

c) The signal names for the power pad inputs are also required to be VDD and VSS (or VDDn, VSSn n = 1-9). The outputs must be assigned unique nodenames, but should not be used.

d) The keywords VDD and VSS (VDDn, VSSn) must also appear in the .TPRINT statement to keep the tester happy. The appropriate data for <tester_type> will be written to a .TPx file. As for any other signal, VDD and VSS must be in pin number order in the .TPRINT statement.

The following example shows how all of the above are included in the netlist.

```
*
* SKELETON NETLIST SHOWING POWER SUPPLY HANDLING
*
.PADS SIGA:1 SIGB:2 VDD:3 SIGC:4 VSS1:5 SIGD:6 VSS2:7
*
POWER    PDC1    VDD    POWER
GROUND1  PDD1    VSS1   GROUND1
GROUND2  PDD1    VSS2   GROUND2
*
*
.TPRINT  SIGA  SIGB  VDD  SIGC  VSS1  SIGD  VSS2
```

3.4.5 DEALING WITH UNUSED PINS

Whenever there are more package pins available than are required by your design (usually it's the other way around!), specific handling of the unused pins is required.

The reserved keyword UNUSED is used to indicate to ZyPAR (via .PADS) and the tester (via .TPRINT) that a particular pin is not used. Since more than one pin may be unused, the keyword can be extended to UNUSEDn where n = 1-30. ZyPAR will take appropriate action when allocating space for bonding pads and ZyPSIM will create output data as though the pin were a masked output to maintain tester compatibility.

For example,

```
*
.PADS SIGA:1 SIGB:2 UNUSED1:3 VDD1:4 UNUSED2:5
*
.TPRINT SIGA SIGB UNUSED1 VDD1 UNUSED2
```

4 TOWARDS A BETTER TEST PROGRAM

4.1 OVERVIEW

As stated in the introduction to this document, the test language does not actually address the issue of testability. It does provide the basic tools, however, and this section presents some useful hints and basic do's and don'ts to help you get the best from using it.

4.2 GENERAL TESTING CONCEPTS

When developing a functional test for an integrated circuit, the system designer is faced with a twofold problem.

* A functional test pattern must be developed which will exercise the device logic as completely as possible to verify functionality and detect all possible failure modes.

* The functional test must be developed using only the available device pins and within any constraints imposed by the test hardware and/or software.

Most systems designers will be familiar with the problems associated with generating a system test. However, the concepts of doing so within the constraints of an IC and IC tester will probably be somewhat new. It is important to get a feel for the problem during (or before) the logic design phase, to ensure a smooth transition from design verification to test program generation with a minimum of iteration. Specific items which must be considered include:

a) Testing must be performed synchronously, whereas the system logic being integrated may be asynchronous in its application. Be prepared to test the device in a different manner to how it normally operates.

b) Access to internal nodes is limited by available pins so initialization of the device and general test visibility may necessitate the inclusion of logic specifically for test purposes.

c) Improper logical operation and/or unstable timing can not be corrected by the proverbial "jumper wire" or "big C" as on a PC board or breadboard. The test scheme should be thorough and be implemented as part of the normal design and simulation procedure so that the need for such drastic action is avoided.

d) Special provisions must be made for any analog sub-systems and memory functions incorporated into the device.

The logic design will almost certainly (should) be influenced by these (and other device specific) testing requirements. This is important, since it will probably also influence system design. So if you're unfamiliar with IC testing, and interested in designing a chip which works in your system first time out, read on. This section is for you.

4.3 INITIALIZATION

The tester is very unforgiving and close isn't anywhere near good enough. The test must begin from a known state, and proceed always knowing what to expect (except when masked). Its up to you to ensure that this happens.

When a chip is powered up at the beginning of test, there may be a zillion and one possible states. If your initialization routine covers a zillion of them, Murphy will ensure that the tester sees the odd one a disproportionate amount of the time. This results in dismal failure or flaky parts.

The lesson here is that there is no substitute for a direct reset to all storage elements in the network. This not only ensures a known state, it usually allows the known state to be reached in the shortest time (minimum number of applied test vectors). Since time is money, you win two ways.

If there are storage elements in the network without direct resets, then as long as they have appropriate visibility to known clock and data conditions, a known state can be reached. However, this method may require a considerable number of test vectors to reach the initialized state. The trade off is silicon area versus test time and/or vector limitations in RAM, requiring multiple memory loads. The need to go this route is usually brought on by lack of overall system (rather than IC) initialization.

Whenever sequential clocking schemes are required to precondition or initialize various sub-systems or logic blocks, the chances of overlooking an illegal state (preventing initialization) are high. Don't let Murphy catch you out - go for the direct reset every time. It may cost a little more logic, but what price success?

4.4 PIN VISIBILITY

In order to properly test an integrated circuit, the tester must have pin visibility to the logic. That's forty four dollars worth of words meaning if you can't make/see it wiggle, it won't be tested. The tester can only apply stimuli and monitor responses through the device pins. A simple enough concept, but one that is often overlooked.

When a particular function is deeply embedded in the overall logic without adequate access to the outside world, the chances are more than good that it cannot be properly tested. At the very least, it will probably cost an arm and a leg in terms of test time and pattern development, plus a 50% reduction in your desire to remain in engineering.

So, having determined the need for pin visibility, how do we achieve it and how is it going to help. Let's find out.

4.5 TEST MODES

The usual way of obtaining pin visibility (to otherwise inaccessible nodes) is by the use of test mode switching. As the term implies, the device will be switched from its normal mode of operation into the test mode (or modes, dependent on the method employed).

The most common method requires the use of a dedicated device pin. In normal operation, the pin will be held low (or high) and the chip will function normally. For testing specific (or all) parts of the chip, the opposite level is selected. A very simple method, but it does cost a pin which may not be available.

The next best approach is to use redundant input states. (You may think a good design doesn't have any, but we won't pursue that!) Let's suppose we have three input control pins A, B and C. Further assume that pin B is never high when A and C are low, during normal system operation. That's a golden opportunity for generating a test mode, equivalent to having a unique pin available.

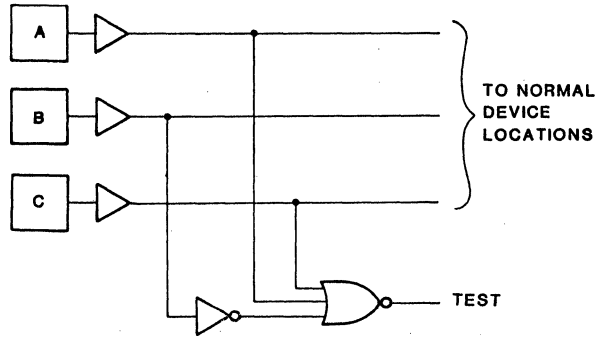


FIGURE 4.1 REDUNDANT STATE TEST MODE GENERATION

In the circuit of Figure 4.1, when "B" goes high and "A" and "C" are low, signal "TEST" can be used to switch from normal to test mode. This method works as long as you can guarantee that:

a) the decoded condition will never occur during normal system operation.

b) the pins used for decoding are not required for testing in the test mode!

c) the test signal is de-glitched somewhere down the line.

Some methods may require complex clocking schemes to precondition the chip into the test mode. This approach should be avoided, if at all possible, since it's easy to overlook unanticipated conditions. If you decide to try a clocked scheme, proceed as though you are walking barefoot through a long dark tunnel, randomly strewn with broken glass. Get the picture? However, sequential methods can be used to expand a single test mode into multiple test modes. That's perfectly acceptable. Figure 4.3 shows one such case, but first let's consider a simple method of test mode expansion.

Assume it is required to generate up to eight test modes in conjunction with a dedicated test mode pin. The scheme shown in Figure 4.2 could be used.

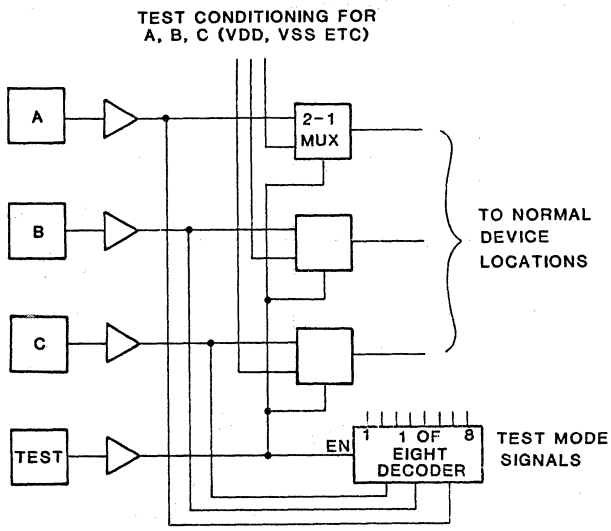


FIGURE 4.2 MULTIPLE TEST MODES WITHOUT PIN LIMITATIONS

The test mode pin switches the inputs being used for test mode decoding into an acceptable state. The decoder is enabled so that pins "A", "B" and "C" can determine which test mode (TEST1 through TEST8) will be selected.

When multiple pins are not available, to permit simple decoding, a counter and one additional pin (unused for the particular test using test modes) can be used. This technique is illustrated in Figure 4.3.

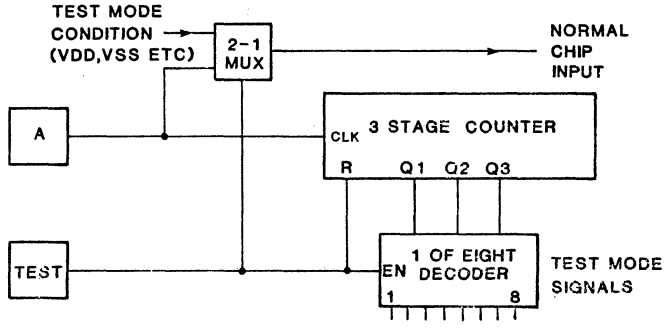


FIGURE 4.3 MULTIPLE TEST MODES USING TWO PINS

OK, We've beaten this subject to death. The important concept here is that we've added logic just to make our design testable. Needless to say, this is best done when the logic is being designed, not during test vector generation when it is discovered that there is a testability problem. Worse still, on the fateful day when parts are received which zip through the test program (pun intended), but refuse to play in your system.

4.5.1 TEST TIME

Now that we have pin visibility through the use of test modes, what exactly have we gained? Well, we have the ability to test things we couldn't test very well (or at all) before, as well as the potential to test things faster.

Look at the logic of Figure 4.4. To fully test that output "C" goes high and low, requires 2^N clock pulses at "A", where N is the number of stages in the counter. In addition, output "D" is required to precondition other parts of the logic before they can be tested, possibly adding to the overall vector count.

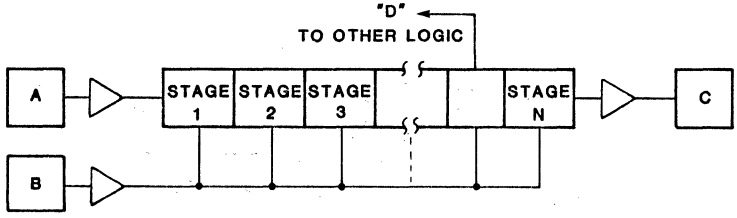


FIGURE 4.4 LONG COUNTER IS BAD NEWS FOR TEST TIME

Now look at Figure 4.5. The counter has been split into two halves. Each part can be tested in parallel in only 2^N/2 clock pulses. Nobody can deny the savings assuming that N is high enough. Again, the concept of adding logic to be used in conjunction with a test mode pays off.

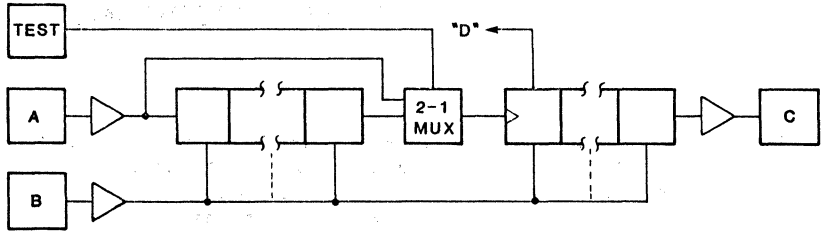


FIGURE 4.5 SPLIT COUNTER SPEEDS THINGS UP

An even bigger payoff can be achieved when the input to a long counter chain is derived within the chip and may occur very infrequently. Test time may now be interminably long. By using the test mode logic to give the counter clock input a path to the outside world, life becomes worth living again. In a case like this, N need not be unduly large before significant savings are made. This would depend on the ratio of regular versus test clock frequency.

4.5.2 NODE ACCESS

In addition to allowing the counter test to be speeded up, node "A" in Figure 4.5 receives input directly from the tester enabling another portion of the device logic to be tested faster.

Accessing internal nodes for output is also made easy by using test modes. A simple 2-1 multiplexer can be used to select different outputs as shown in Figure 4.6.

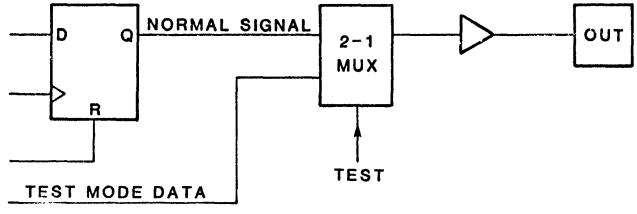


FIGURE 4.6 MULTIPLEXING OUTPUTS IN TEST MODE

4.6 MINIMIZING TEST VECTORS

The tester pattern memory is limited just like any other memory, so it's in your interest to minimize the absolute number of test vectors required. This is a somewhat difficult topic to discuss (out of the context of a particular design), however, the following fundamental concepts always hold true.

4.6.1 DON'T FLOG A DEAD HORSE

Whenever output data is repetitive, we don't need to test it forever. An output which is masked raises the potential for data reduction, since ZyTEST will not consider an output which is masked (untested) when compressing data. If you only associate masks with Halloween, return to section 2. Remember, first level compression consists of consolidating consecutive occurrences of identical vectors.

4.6.1.1 Masking Considerations

In masking a repetitive signal, it is implied that once tested, it is not possible for the output data pattern to change. A typical example of an output which could be masked (after testing) would be a simple divider chain. Outputs which are data dependent should not be "blanket" masked unless all data sequences have been tested.

Also, it should be remembered that changing masking conditions will usually cause redefinition of a mask register and will therefore require an additional memory location. It may not be economical (in terms of memory) to repeatedly test/mask an output. There are certain cases, however, where masking previously verified outputs will reduce the amount of memory required to execute the test.

4.6.1.2 An Example of Repetitive Data Masking

If your device has an output (such as "A" in Figure 4.7) generated by a simple divide by N from the master clock, turn on the mask for that output after it has toggled high and low twice. (Or more if you're paranoid).

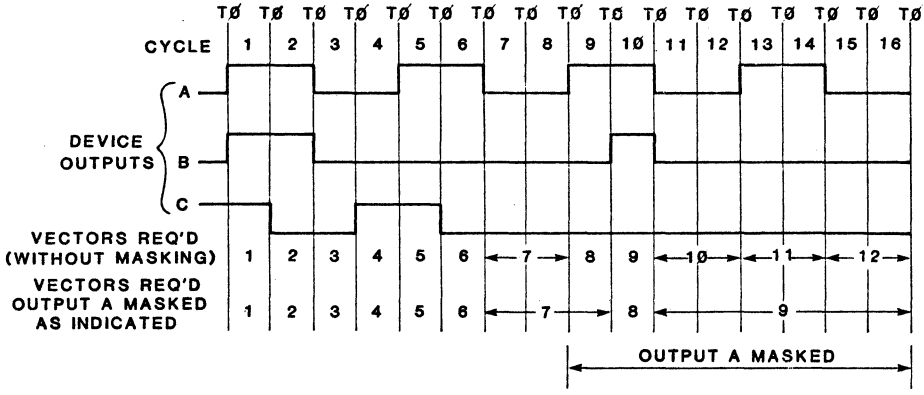


FIGURE 4.7 MASKING OF REPETITIVE DATA

As can be seen, if the output is masked after two toggle cycles, only two unique vectors are required versus five without masking. When such a condition occurs throughout the test, countless wasted vectors are generated. Although ZyTEST may generate subroutines (dependent on actual values of data for other outputs), it is obviously best to mask such outputs immediately after they have been tested.

4.6.2 BUT THAT'S NOT HOW IT WORKS IN THE SYSTEM

Whenever I/O pins are used, potentially high vector counts due to I/O switching can be generated. When a pin is switched from an input to an output, both the direction and mask registers need to be updated. If the new conditions already exist in the alternate registers (remember DA/DB, MA/MB in section 2?), all is well. However, if new conditions are required, the register contents must be redefined. ZyTEST will generate the appropriate register load statements (LSET's) and the data will be stored in the tester RAM. This represents a RAM location which cannot store a functional test vector.

So what? What can I do about it you may ask. Plenty. Don't be too narrow minded in your approach to test methodology in terms of adhering to system operation when testing. It's not necessary and it's extremely costly when it comes to I/O switching.

For example, assume there are one or more I/O pins which during system operation frequently switch from input to output. If the test is performed in the same way, the chances of producing a large pattern are extremely high, due to register loads. A far better approach is to make the pin an output and wiggle it a few times to test it. Then make the pin an input and forget about it. If this is not possible, at least mask the output when repetitive data is produced.

The concept of "non-system" testing is applicable to all parts of the test, not just I/O switching, and its importance cannot be overstressed. The world would not be full of digital watches if it took four years to test the leap year function.

4.6.3 GENERAL GUIDELINES

Some general guidelines which should always be followed are:

a) Test as many parts of the logic as possible in parallel.

b) Don't use "random" input patterns.

c) When data changes can be made at "no vector count in particular", don't choose vector counts arbitrarily, align changes to a vector when other data is already changing.

4.7 OUR FRIEND THE I/O AGAIN

I/O's keep cropping up don't they? They may be the answer to a system designers prayer, but when it comes to testing they are not very helpful at all. You know already that special .TTABLE handling is required and that they cause vectors to be added to the functional test pattern due to register loads. I/O's can also cause difficulties during test execution, dependent on the timing scheme that you implement.

4.7.1 IS IT AN INPUT OR AN OUTPUT

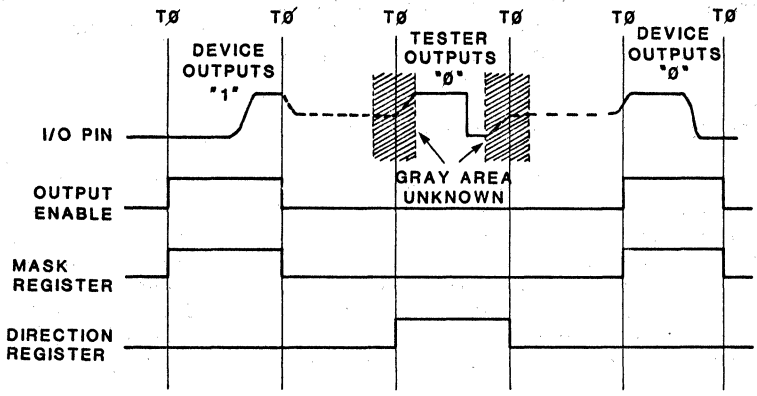
When an I/O pin is switched from an input to an output, the tester must also switch from being a driver to a response monitor. Unfortunately, this doesn't happen instantaneously. The nominal switching point is T0. However, since pipelining is used in the tester hardware, the actual switch over can occur before as well as after T0. The switching point can be +/- 15nS or +/- 50nS dependent on the type of test head used (more on this in section 4.8.5). This can potentially cause some major problems as follows:

a) During the "gray area" (either side of T0), the tester and your device may both be drivers simultaneously. Oops! If the output of your device is low impedance (what output driver isn't?), considerable current is forced into the device. Apart from the possibility of damage, it is possible to cause internal logic to change state, due to power supply disturbance caused by the injected current.

b) If the strobe position is in the gray area, you're really in trouble. Whenever there is bus contention (that's what we've got) the output you're trying to measure will get clobbered, since the tester will always win the fight when it's driving an opposite level to that which the device is driving. This causes an immediate failure due to the strobe sampling a value it did not expect. Bad news.

c) When data transitions are made on an I/O pin just prior to T0, and the device becomes an output in the next cycle, problems of a different nature occur. In this case, if the tester stops driving (in the gray area), the input data seen by the device may be invalid. At best, data setup time will be compromised.

In dealing with this situation, the best possible solution is to avoid single cycle I/O switching. This means don't switch an I/O pin from an input in one cycle to an output in the next and vice versa. Always allow at least one dead cycle between modes and the possibility of a tester/chip conflict is eliminated. This is illustrated in Figure 4.8.



Cycle #	Device	Tester	Mask	Direction
1	Talk	Listen	1	0
2	Tristate	Listen	0	0
3	Listen	Talk	0	1
4	Tristate	Listen	0	0
5	Talk	Listen	1	0

FIGURE 4.8 I/O SWITCHING TIMING SCHEME

Note that although there is no longer any chance of a fight between tester and chip, the data input to the chip can't be guaranteed in the gray area around T0. This affects how you must implement your read timing. Data should be read (clocked in by the chip) before the onset of the gray area. Reading data during the gray area means garbage in and later garbage out causing the device to fail the test.

If you absolutely have to use single cycle I/O, then the strobe is not permitted to be in the gray area. You must also live with the consequences of tester/chip conflicts.

4.7.2 THOU SHALT NOT STEAL

When an I/O switch causes a mask register or direction register to be redefined, a condition known as "cycle stealing" occurs. Since it takes a finite amount of time to load the register, the tester may not be able to execute the functional test pattern and load the register in the same cycle. This will depend on test execution speed (.TPERIOD). When this is the case, an extra cycle is created to allow time for register loads and this is known as the stolen cycle. The table below shows the .TPERIOD below which cycle stealing will occur for the specified conditions.

TYPE OF TESTER	MINIMUM .TPERIOD BELOW WHICH CYCLE STEAL OCCURS (nS)	
	SINGLE REGISTER LOAD	DOUBLE REGISTER LOAD
SENTRY7	200	400
SENTRY20	100	200

Figure 4.9 shows how this affects the timing of input data and strobing of output data.

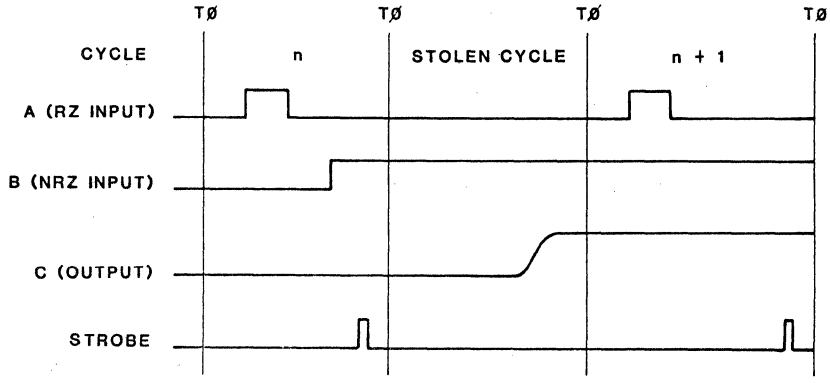


FIGURE 4.9 CYCLE STEAL TIMING CONDITIONS

There are some key points to note. As far as inputs are concerned, the stolen cycle just stretches things out. RZ/RO signals don't produce a pulse and NRZ inputs are extended and do not change state. The chip output will still produce data as expected, but the strobe does not occur in the stolen cycle.

This has two main effects.

a) Input data setup time is greater. For example, in Figure 4.9 if input A in cycle n+1 is clocking data B (which changed in cycle n), the setup time is extended by the period value.

b) Output C is not strobed until normal <strobe_time> plus the period value. This allows extra time for the output to propagate and therefore does not verify correct propagation delay.

This may or may not cause undertesting. If input A and output C occur in other parts of the test pattern when cycle stealing does not occur, then they will be properly tested at those times. However, if cycle stealing is always present when a particular input or output is being tested, setup time and/or output propagation delay is not properly tested. The least of all problems caused by cycle stealing is an increase in functional test time due to the extra cycles. However, this is usually a small percentage of overall test time.

You should be aware of whether cycle stealing will occur in your test pattern when it's executed on the tester. The functional test pattern file (.FTP) produced by ZyTEST contains this data. If register load statements (LSET) appear in this file and you are operating with a .TPERIOD smaller than those shown in the table, cycle steal will occur. Two LSET statements appearing as consecutive statements in the .FTP file means that both mask and direction registers are loaded. This is the double register load referred to in the table.

NOTE: I/O switching need not be present to cause cycle stealing, since mask register loads alone can cause it when .TPERIOD is small enough.

4.8 SOME SPECIFICS ABOUT TIMING

Time can be your enemy or time can be on your side. You decide. Proper choice of timing is the most important aspect of your test scheme, and it goes all the way back to system timing. If your system timing is such that it does not lend itself to testing, then a disaster is lurking and waiting to happen.

4.8.1 MARGIN MAKES THE DIFFERENCE

When designing your logic, don't work too close to the edge of your timing requirements. If a path delay cannot be allowed to exceed "X" nS, don't settle for a solution which produces a worst case simulation result of "X-1" nS. Buy some insurance. A good rule of thumb is a margin of 10%-15% (with an absolute minimum value of 5nS) faster than your requirement. That helps everyone.

Anyone can make a mistake so it's important to guard against them. If a cell doesn't work as advertised (heaven forbid!), or you miscalculate and your system really needs "X-3" and you allowed "X-1", the end result is the same - failure.

The lesson here is clear. Providing adequate timing margin during the design phase insures against mistakes, tester tolerances, and all kinds of unexpected situations. Margin makes the difference between success and failure.

4.8.2 .TPERIOD SELECTION

Choosing the tester period is probably the easiest of all timing choices. Most devices have a master clock and this becomes the natural choice for the tester clock period. When this isn't the case, just remember a few simple guidelines.

The tester (and test language) cannot produce more than two data changes within a test period. This will immediately limit your range of choice. A further limit is the maximum operating rate of the tester, currently 10MHz for Sentry 7 and 20MHz for Sentry 20. However, don't push it, the lower the better.

In addition, using NRZ mode (limiting changes to one per period) is usually preferable, unless this results in too small a tester period.

4.8.3 CHOOSING FORMATTING MODES

The best rule here is "keep it simple". The more straightforward your timing scheme is, the more chance that everyone (including you) understands it and that it will work.

To this end, NRZ mode timing is the best (simplest) choice for all signals except clocks. Use NRZ T0 mode whenever possible and minimize the number of different phases (.TGENs) in use. Remember, they are in short supply and using less of them also aids understanding.

When deciding on RZ or RO mode signals, when two data changes within a period are required (eg: clocks), two factors influence the decision.

a) Proximity of edges to T0. If it is absolutely essential to have a rising edge exactly aligned with T0, then RO mode must be used. However, inactive levels of the signal (if any) will be "1".

b) Inactive level. If it is absolutely essential to have an inactive level of "0", then RZ mode must be used. However, rising edges cannot be closer to T0 than 10nS. (Remember our minimum T1 requirement).

For example, assume it is required to produce a 50% duty cycle master clock whose rising edge is concurrent with T0. Figure 4.10 shows the possibilities.

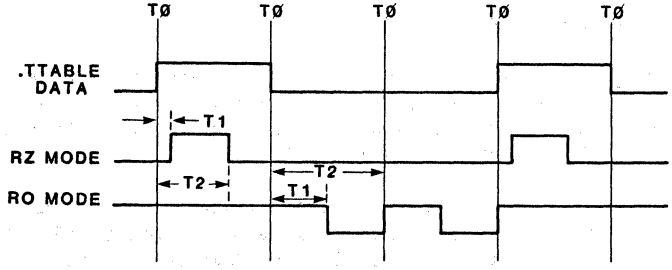


FIGURE 4.10 MASTER CLOCK EXAMPLE

For tester timing generators (and hence .TGEN statements), there is a hardware minimum timing value of 10nS. T1 represents the pulse leading edge and T2 the trailing edge, both with respect to T0. Thus it can be seen that an RZ mode signal has a 10nS minimum delay from T0 whereas the RO mode signal does not. Inactive levels are as shown.

Although it's usual (and easier to understand) to align the master clock signal with T0, it is not a requirement and may not always be possible. The position of the tester strobe has some limitations (see section 4.8.5) and it may be necessary to have a master clock which is not aligned with T0 in order to overcome them.

As a final word on formatting, for positive clocked logic, RZ mode is the logical best choice. That way, we don't have to "think upside down" to understand the timing.

4.8.4 DRIVING THE TESTER

It's important to consider the load capacitance presented by the tester in your simulation. This can easily be included in the net list using the ZyPSIM CAP primitive. However, the impact on timing may not be accommodated quite so easily.

The device performance should be such that your timing requirements can be met while driving the 50pF tester load. If your system load is small and device speed is high, it may not be possible to meet your propagation delay specifications when driving a 50pF load. However, every attempt should be made to over design the outputs so that normal performance is obtained under worst case conditions and a 50pF load.

The alternative solution is to design a test which runs at a lower speed and allows outputs to propagate during a tester period while driving 50pF. For system verification, a full speed simulation driving the reduced system load can be performed. ZyMOS will guarantee system speed performance (even though lower speed testing is performed) provided that the only difference between system speed verification and test simulation netlists are the values of .TPERIOD, .TSTROBE, and output loading capacitors.

Regardless of the method chosen, the netlist used in generating the simulation output for file transfer must include the tester loading. This is achieved by including a CAP primitive connected to each output or I/O pad cell. For example:

```
*
*       PAD CELLS
*
P1  PBE1  IN1  OUT1
P2  PBE1  IN2  OUT2
.
.
.
*
*       TESTER LOAD CAPACITANCE
*
C1  CAP   OUT1:50
C2  CAP   OUT2:50
```

4.8.5 STROBE POSITION

Probably the most common mistake made in choosing strobe position stems from a conceptual misunderstanding. It is not the tester's task to Measure dynamic AC characteristics, they are guaranteed by design and proper choice of output pad cells. The tester does a spot check only, at a particular point in the cycle - the strobe time.

Do not attempt to measure propagation delay to a high degree of accuracy on the tester. If an output is not as fast as you thought it might be, the tester says fail and a good device may be lost. Test it to a reasonable specification, provide good margins during design, and everybody wins.

4.8.5.1 Restrictions

Due to hardware limitations, there are some specific restrictions regarding choice of strobe position. Maximum strobe time is specified in terms of the tester period defined in the .TPERIOD statement.

a) If the signal is a dedicated output, (or an I/O that does not use single cycle I/O switching), then <strobe_time> must be less than or equal to <tester_period> - 20nS

b) When single cycle I/O switching is used, the strobe cannot be placed within the "gray area" of chip/tester conflict. This is the same for both SENTRY7 and SENTRY20, but it does differ between two types of test heads. The gray area is up to +/- 50nS for one type of test head and +/- 15nS for the other type. Choosing a particular head type impacts test cost and will therefore affect device pricing.

   The value for <strobe_time> is restricted to less than or equal to <tester_period> - 60nS for low speed heads, and less than or equal to <tester_period> - 25nS for high speed heads.

4.9 A FEW WORDS ABOUT ANALOG TESTING

4.9.1 TEST PHILOSOPHY

Testing analog functions requires following the same basic rules that have been previously outlined. However, analog testing is further complicated by the fact that the analog sub-system is usually embedded in a much larger digital system. As a consequence, it's usually not practical to test the entire analog sub-system in its normal operating mode. The recommended approach is to test each individual cell used on the chip according to its own measurable specifications.

When dealing with a chip having only a few analog cells, this is a fairly straightforward task. It requires that a few package pins be allocated in order to gain access to the inputs and outputs of the cell. If external components are being used, these pins may already be available. Each cell in the chip can now be tested for individual function and performance. Testing more complex analog functions is a bit more challenging and requires some creative thinking.

Since the number of cells and functions contained in a complex analog system can be significant, and the number of pins is usually limited, hard wiring the inputs and outputs of all cells to pins is usually not feasible. Therefore, it is imperative that a testing philosophy be developed as the design progresses from initial concept. Designing testability into the chip is not a new concept (see section 4.5), however, it cannot be overstated. Often the success of a particular chip (when transferred to production) will depend on how well the test issues have been addressed.

4.9.2 USING TEST MODES

We have seen that the basic approach in designing for testability is to include test circuitry that can be used to change pin functionality. Allocating a pin or logic combination to define a test mode greatly simplifies the task. Testing analog cells may require the use of analog switches and tri-state digital outputs in order to allow the outputs of analog cells to be examined at these pins. The same technique can also be applied to the inputs of cells needing to be forced.

Also, reconfiguring the system to allow a particular type of test to be performed is not a bad idea. For example, using an existing counter or memory, the digital codes for a D to A converter can be created. Since these codes are known and predictable, the output of the converter can be measured and verified. This avoids the need to supply 8 to 12 bits of digital information externally and minimizes pin limitation problems. A similar result could also be accomplished by building a serial input port and serial to parallel converter into the chip. It's certainly apparent from these examples that a little forethought during the design of a chip goes a long way. The hardest part in using this philosophy is trusting in the fact that total chip performance can be predicted by the individual component performance. If this were not true, the design of systems using off-the-shelf components could never have succeeded.

4.9.3 TESTER LIMITATIONS

The tester operates in the analog domain very simply, it can force voltage and measure current or vice-versa. Therefore, care must be taken to avoid testing cells which are internally AC coupled to the test pins. It is recommended that AC coupling always be done external to the chip. Other tester limitations that should be remembered include the 50pF load presented by the tester to the chip and forcing voltages of not less than 10mVp-p.

4.9.4 TEST TIME

At this time it's appropriate to say a few words concerning test times. Testing analog functions generally takes more tester time because of the number of calculations and measurement points required for a specific function. Another reason is that multiple tests are often required for a single cell. An example is Op-Amp testing which requires different types of measurements for offset voltage, open loop gain, common mode rejection ratio, etc. However, the single cell testing concept will still simplify testing and reduce test times, because the specifications are measured directly and not implied by sorting the results of more complicated measurements such as system transfer functions. From the standpoint of test development time, it takes much less time to build a test program from blocks of pre-defined tests than to create a program that is unique and designed for one chip.

4.9.5 WHERE'S ZyMOS

The "test by cell" approach makes analog testing of mixed analog/digital ZyP devices possible. ZyMOS supports this by providing pre-defined tests for all major analog cells. In addition, each analog cell datasheet defines the parameters that are production tested and guaranteed. (Look for an * in the conditions column.)

Of course, this method is valid only if all inputs and outputs are available for test. If all ports are not available, a reduced set of specifications will be tested. When all ports of a cell are inaccessible, the cell must then be tested as part of the system which is usually difficult and time consuming.

Since the testing of a chip containing analog circuitry is a more difficult task than testing purely digital ZyP designs, it is suggested new designs be carefully scrutinized for testability early in the design cycle. ZyP Design Support engineers can offer assistance by suggesting approaches which could result in improved testability.

5 THE TEST LANGUAGE IN ACTION

5.1 OVERVIEW

This section presents an example of a representative design to show how all the test language features are combined to produce a tester compatible output. The subject of test strategy development is discussed in general as well as in the context of the example described.

5.2 A METHOD IN ALL THIS MADNESS

When developing an integrated circuit, you have seen that it's extremely important to have a game plan which covers all aspects at the start of the design phase. A methodology which includes test pattern generation will not only ensure a testable device, it will also help to produce a "clean" design.

Now that you are aware of the combined capabilities and limitations of the tester and test language, it's appropriate to consider a strategy which addresses the testing issues. Some of the key things to consider up front are as follows.

5.2.1 BE A GOOD SCOUT

Be prepared! Define your test requirements. Knowing what you want is half the battle in any task. Before designing device logic, you need to know what functions it has to perform and how fast. It's not much fun (or use) trying to do it without this information. The same applies to test.

Before doing extensive simulations, define in general terms what you want to test and which parameters are important to verify. Detailed tests of each function can then follow. Also, try to anticipate invalid conditions. For example, a computer operating system has to filter all kinds of garbage and try to return useful messages to the user. That's not a bad philosophy to apply to logic design. Expect the unexpected!

Above all, make sure that the simulations for test development are performed under worst case conditions and include estimated interconnect and tester load capacitance.

The following sections outline a few major points which you should take the time to address.

5.2.2 DURING LOGIC DESIGN

a) Can the device be adequately initialized? If your simulations show "X" states which spread like a disease, the answer is no. You must be able to initialize the chip, preferably with a direct reset. If the overall system designer has not provided a system wide reset, ask him to come up with an initialization scheme for your chip. He'll give you a reset signal.

b) Is there adequate access to all parts of the logic from the outside world? For example, have you buried a RAM or ROM without providing address and data pin visibility.

c) Are there some parts of the logic which require forever to test? If you can answer yes to this question or b) above, you should consider changing your logic and/or adding test mode logic.

d) Are your outputs "on the edge of a cliff"? In other words, will a slight performance degradation cause failure? If so, it's time to provide some margin.

e) Make sure that the outputs are capable of driving the tester while meeting full speed timing requirements.

5.2.3 DURING TEST DEVELOPMENT

Assuming that the first hurdle is passed and you have fundamentally sound and testable logic, the next step is to develop the test pattern and timing scheme.

Remember that in addition to the primary goal of verifying good devices and weeding out bad ones, we're trying to minimize test vectors, minimize test time, and avoid timing problems.

a) Can your proposed timing scheme be implemented using the test language? If not, it's probably a case of marginal or bad timing design, or at best a need to test differently from normal system operation.

b) Does your device have I/O switching? Remember the things that have been discussed and avoid all the associated pitfalls.

c) Use the formatting modes best suited to each signal and minimize timing generator usage.

d) Choose a strobe position which provides a reliable test and is not in danger of failing potentially good parts.

5.3 AN EXAMPLE DEVICE

This example uses the ZyMOS "standard" demonstration circuit which appears in numerous other documents. It's a convenient device to use, since it does not have I/O switching and therefore simplifies things. Nevertheless, it allows many of the advantages of using the ZyPSIM test language to be illustrated.

The logic diagram for one half of the counter is shown below in Figure 5.1.

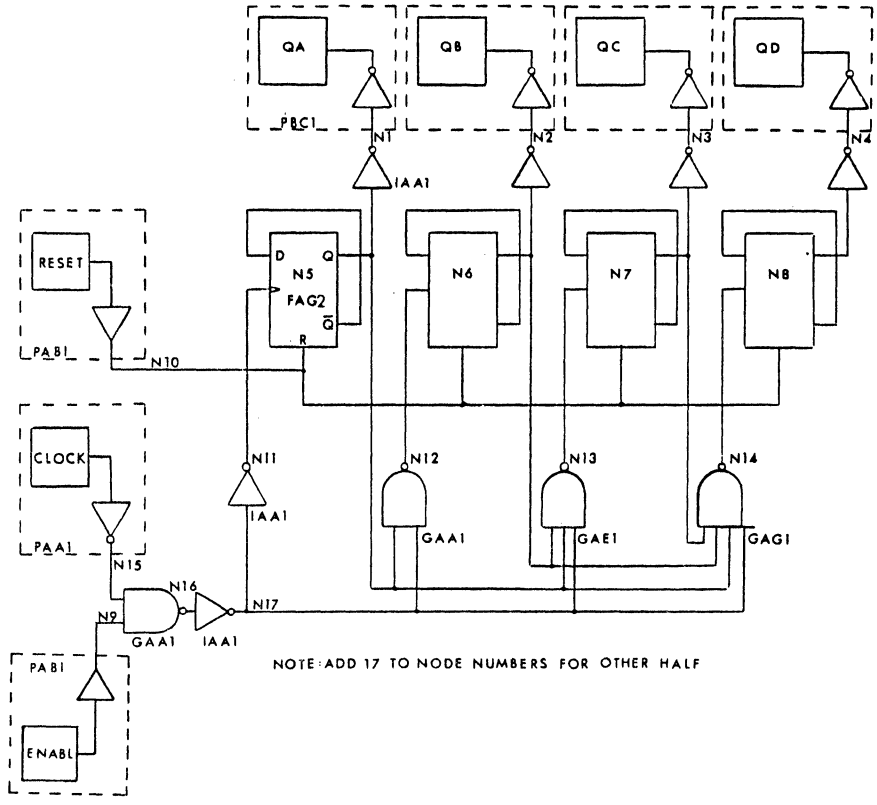


FIGURE 5.1 EXAMPLE DEVICE - DUAL 4 BIT SYNCHRONOUS COUNTER

5.3.1 DESCRIPTION

The circuit emulates the functions of a standard CMOS 4520 which is a dual 4 bit synchronous counter. Each half of the logic is identical and contains a 4 bit counter with clock, enable and reset inputs and buffered Q outputs. The clock is positive edge triggered and the enable is active high, ie: a logic "1" enables the counter. The enable input can also be used as a negative edge clock input if the clock input is held low. The netlist that follows includes the necessary test language statements and run commands to completely define a tester compatible functional test pattern.

```
*ZYPSIM Zy50000 CHMOS DEMO USING THE TEST LANGUAGE
*
* NOTE: THE SIMULATION USES WORST CASE COMMERCIAL ENVIRONMENTAL
* CONDITIONS AND INCLUDES ESTIMATED INTERCONNECT CAPACITANCE.
* A TOGGLE ANALYSIS OUTPUT REPORT IS ALSO REQUESTED.
*
.GUARDBAND VDD=4.5,5.5 TEMP=0,70 CONDITION=NOM
.INPUT ZYP>ZYPSIM>ZY5LIB>LAYOUTCAP
.TOGGLE
*
*        NETWORK DESCRIPTION
*
*          PIN ASSIGNMENT
*
.PADS CLK1:1 EN1:2  QA1:3  QB1:4  QC1:5  QD1:6  RST1:7  VSS:8
+     CLK2:9 EN2:10 QA2:11 QB2:12 QC2:13 QD2:14 RST2:15 VDD:16
*
*
*          PADS
*
CLOCK1        PAA1 CLK1 N15
ENABLE1       PAB1 EN1  N9
QA1           PBC1 N1   QA1
QB1           PBC1 N2   QB1
QC1           PBC1 N3   QC1
QD1           PBC1 N4   QD1
RESET1        PAB1 RST1 N10
VSS           PDD1 VSS  GROUND
CLOCK2        PAA1 CLK2 N32
ENABLE2       PAB1 EN2  N26
QA2           PBC1 N18  QA2
QB2           PBC1 N19  QB2
QC2           PBC1 N20  QC2
QD2           PBC1 N21  QD2
RESET2        PAB1 RST2 N27
VDD           PDC1 VDD  POWER
*
*          INVERTERS
*
N1  IAA3 N5  N1
N2  IAA3 N6  N2
```

```
N3  IAA3 N7  N3
N4  IAA3 N8  N4
N18 IAA3 N22 N18
N19 IAA3 N23 N19
N20 IAA3 N24 N20
N21 IAA3 N25 N21
N11 IAA3 N17 N11
N17 IAA3 N16 N17
N28 IAA3 N34 N28
N34 IAA3 N33 N34
*
*          NAND GATES
*
N16 GAA1 N15 N9  N16
N33 GAA1 N32 N26 N33
N12 GAA1 N5  N17 N12
N29 GAA1 N22 N34 N29
*
N13 GAE1 N5  N6  N17 N13
N30 GAE1 N22 N23 N34 N30
*
N14 GAG1 N5  N6  N7  N17 N14
N31 GAG1 N22 N23 N24 N34 N31
*
*          FLIP FLOPS
*
N5  FAG1 N11 N10 N5  NN5
N6  FAG1 N12 N10 N6  NN6
N7  FAG1 N13 N10 N7  NN7
N8  FAG1 N14 N10 N8  NN8
N22 FAG1 N28 N27 N22 NN22
N23 FAG1 N29 N27 N23 NN23
N24 FAG1 N30 N27 N24 NN24
N25 FAG1 N31 N27 N25 NN25
*
* 50 PF LOAD CAPACITORS (SENTRY LOADING)
*
CL1 CAP IN=QA1:50
CL2 CAP IN=QB1:50
CL3 CAP IN=QC1:50
CL4 CAP IN=QD1:50
CL5 CAP IN=QA2:50
CL6 CAP IN=QB2:50
CL7 CAP IN=QC2:50
CL8 CAP IN=QD2:50
*
*                    PATTERN DEFINITION
*
.TESTER SENTRY7                    % DEFINE TESTER
*
.TPERIOD 250                       % 4MHz DEVICE
*
* CLOCK TIMING AND DATA
*
```

```
.TGEN RZ 10 135 CLK1 CLK2
*
.TTABLE CLK1 CLK2
   18     1    1                  % CHECK COUNTER AND POSITIVE CLOCK
   15     1    1                  % PREPARE FOR INHIBIT/RESET TEST
    3     0    0
*
* RESET TIMING AND DATA
*
.TGEN NRZ 50 RST1 RST2
*
.TTABLE RST1,1 RST2,1
    1     1    1                  % START WITH A RESET
    1     0    0                  % REMOVE RESET IN CYCLE 2
   33     0    0
    1     1    1                  % END WITH A RESET
*
* ENABLE TIMING AND DATA
*
.TGEN NRZ 15  EN1  EN2
*
.TTABLE EN1,1  EN2,1
   31     1    1
    2     0    0                  % CHECK INHIBIT
    1     1    1
    2     0    0                  % CHECK NEGATIVE CLOCK
*
* MASK DATA
*
.TTABLE MASK
    1     0                       % MASK DURING INITIAL RESET
   17     1
   13     0                       % REPETITIVE DATA MASK
    5     1
*
*                    RUN COMMANDS
*
*
.TSTROBE 225   QA1,MASK  QB1,MASK  QC1,MASK  QD1,MASK
+              QA2,MASK  QB2,MASK  QC2,MASK  QD2,MASK
*
.TPRINT  CLK1  EN1  QA1  QB1  QC1  QD1  RST1  VSS
+        CLK2  EN2  QA2  QB2  QC2  QD2  RST2  VDD
*
*
.MAXTIME=9000                     % 36 VECTORS * TPERIOD
.XREF
.GO
.END
```

5.3.2 TEST DEFINITION

Since this device does not have problems due to a lack of pin visibility or bidirectional pins, we can start right away on the functional test. We will need to check the following functions:

a) Counter operation for all states.

b) Reset operation.

c) Positive and negative clock operation.

d) Counter inhibit.

5.3.3 INPUT PATTERN

Since the device contains two identical counters, both halves will be tested in parallel. The functional test will be as follows:

a) Initialize the chip via the reset input. Apply 16 clock pulses through the clock input to check the positive clock and all states of the counter.

   In order to properly check the reset function, the counter should be at state 15 (all "1"s). During this test we can check the inhibit and negative clock features of the enable input as follows.

   NOTE: Some stages of the counter may have powered up in the "0" state so the initialization reset pulse is not a complete check.

b) Apply 15 more clock pulses through the normal clock input, with the enable high for 14 of them. This will check the inhibit function of the enable input and the counter should now be at count 14.

c) Apply 1 negative clock via the enable input, which checks this feature and also brings the counter to count 15.

d) Finally, apply a reset pulse and look for all "0"s, this checks the reset function properly.

5.3.4 MASKING

The outputs need to be masked only during the initialization period. However, masking can also be applied to reduce vector count. Although we're not in trouble on this device with either test time or vector count, it's good practice to follow the rules and mask repetitive data.

In this case, we can mask during the time the counter is being clocked to reach count 15 for the reset test, since the counter has already been tested. In fact, we should only mask for 13 of the 15 cycles, so that count 14 can be verified prior to and during inhibit.

5.3.5 TIMING SCHEME

5.3.5.1 Simulation Period

The circuit is designed to operate at 4MHz so a period of 250nS will be used. This should always be the first timing value chosen, since everything else is based on it.

5.3.5.2 Clock Input

For the clock signal, we need a positive going pulse with an inactive level of "0", which requires a formatting mode of RZ. Timing for <T1> and <T2> for a 50% duty cycle will be 10nS and 135nS (remember 10nS minimum value).

5.3.5.3 Enable Input

For the enable signal, we need to have several cycles of "1"s and "0"s so a formatting mode of NRZ is required. Furthermore, since we will need to generate a clock pulse (to check the negative clock feature of the enable input), the timing of the enable input should be close to that of the clock input. This requires a NRZ TG mode format and the value chosen for <T1> is 15nS to ensure a clean switch from enable to inhibit, while maintaining a value close to the 10nS of the clock.

The values need to be kept close since the same strobe will be checking output response, regardless of the input which caused it.

5.3.5.4 Reset Input

Reset is a similar situation to the enable and requires a NRZ TG mode format. Again, since output response with respect to reset is checked by the same strobe, <T1> must be properly chosen. This time, however, we need to consider the difference in propagation delay between clock and reset. The reset to output propagation delay is 40nS faster than clock to output, which means that the <T1> value for reset is 50nS.

5.3.5.5 Strobe Position

The strobe position is chosen after determining the worst case propagation delay for the device environmental conditions. During simulation, estimated capacitance values are included as well as tester load capacitance (50pF) on all outputs. The ZyPSIM predicted value is 200nS, adding 10nS margin gives 210nS. The latest clock (enable) occurs at 15nS so 225nS is a good number for <strobe_time>.

The complete input pattern and timing scheme is shown in Figure 5.2.

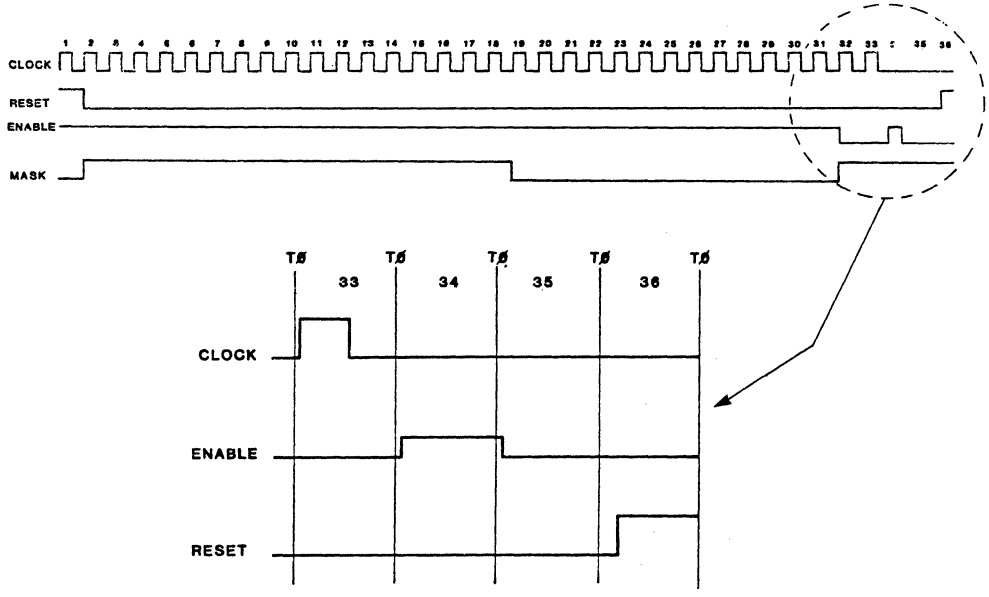


FIGURE 5.2 FUNCTIONAL TEST INPUT PATTERN

5.3.6 OUTPUT

```
1  EXAMPLE NETWORK #1 USING THE ZyPSIM TEST LANGUAGE

       CEQQQQRVCEQQQQRV
       LNABCDSSLNABCDSD
       K11111TSK22222TD
       1     1 2     2


       CYCLE

       00000 dDmmmmDmdDmmmmDm
       00001 DDmmmmDmDDmmmmDm
       00002 DD0000dmDD0000dm
       00003 DD1000dmDD1000dm
       00004 DD0100dmDD0100dm
       00005 DD1100dmDD1100dm
       00006 DD0010dmDD0010dm
       00007 DD1010dmDD1010dm
       00008 DD0110dmDD0110dm
       00009 DD1110dmDD1110dm
       00010 DD0001dmDD0001dm
       00011 DD1001dmDD1001dm
       00012 DD0101dmDD0101dm
       00013 DD1101dmDD1101dm
       00014 DD0011dmDD0011dm
       00015 DD1011dmDD1011dm
       00016 DD0111dmDD0111dm
       00017 DD1111dmDD1111dm
       00018 DD0000dmDD0000dm
       00019 DDMmmmdmDDMmmmdm
       00020 DDmMmmdmDDmMmmdm
       00021 DDMMmmdmDDMMmmdm
       00022 DDmmMmdmDDmmMmdm
       00023 DDMmMmdmDDMmMmdm
       00024 DDmMMmdmDDmMMmdm
       00025 DDMMMmdmDDMMMmdm
       00026 DDmmmMdmDDmmmMdm
       00027 DDMmmMdmDDMmmMdm
       00028 DDmMmMdmDDmMmMdm
       00029 DDMMmMdmDDMMmMdm
       00030 DDmmMMdmDDmmMMdm
       00031 DDMmMMdmDDMmMMdm
       00032 Dd0111dmDd0111dm
       00033 Dd0111dmDd0111dm
       00034 dD0111dmdD0111dm
       00035 dd1111dmdd1111dm
       00036 dd0000Dmdd0000Dm
```

6 COMMAND SYNTAX REFERENCE

6.1 .TESTER

Defines the type of tester to be used for device testing.

Syntax:

.TESTER <tester_type>

Valid values for <tester_type> are:

SENTRY7
SENTRY20

Example:

.TESTER SENTRY20

The default when .TESTER is omitted will be SENTRY7.

Refer to section 3.3.1 for more details about the .TESTER command and Appendix A for tester dependent parameters.

6.2 .TPERIOD

Specifies the absolute value of the tester period (and simulation cycle) in nS.

Syntax:

.TPERIOD <period>

Example:

.TPERIOD 400

will produce a test rate of 2.5MHz.

Refer to section 3.3.2 for more details about the .TPERIOD command and sections 4.7 and 4.8 for information regarding .TPERIOD selection.

6.3 .TGEN

Defines the timing and formatting of a signal defined in a .TTABLE.

Syntax:

.TGEN <mode> <T1> [<T2>] <N1>[,<dir1>] <N2>[,<dir2>]..<Nn>[,<dirn>]

<mode> specifies the formatting mode. Valid modes are:

NRZ, RZ, RO, IONRZ, IORZ, and IORO.

<T1> specifies the delay from T0 for NRZ mode signals.

<T1> and <T2> specify leading and trailing edges of a pulse (from T0) for RZ and RO mode signals.

<N1> through <Nn> define the signals controlled by the .TGEN.

<dir1> through <dirn> define the direction control signals, required when an IO mode is specified.

Examples:

```
.TGEN NRZ   50 ABC XYZ
.TGEN RO   100 150 LOW
.TGEN IORZ  50 250 HIGH,HIGH_DIR
```

NRZ signals ABC and XYZ occur 50nS after T0.

RO mode signal LOW is a 50nS negative going pulse occurring 100nS after T0.

Bidirectional RZ mode signal HIGH is a 200nS positive going pulse occurring 50nS after T0 when in input mode. Signal HIGH_DIR controls the direction (1=input).

Refer to section 3.3.4 for more details about the .TGEN command and section 4.8.3 for information regarding the choice of formatting modes.

6.4 .TTABLE

A .TTABLE allows cycle based input data to be defined in a table format. Using a .TTABLE is the only means of defining tester compatible simulation input data.

Syntax:

.TTABLE <node 1>[,<iv 1>] <node 2>[,<iv 2>] ... <node n>[,<iv n>]

where <node 1> through <node n> are the signals to be driven and <iv 1> through <iv n> are optional initial values for NRZ mode signals.

A .TTABLE statement must be followed by table data as follows:

<count> <data 1> <data 2> ... <data n>

<count> specifies the number of times that the associated data will be applied.

<data 1> through <data n> are "0" or "1" corresponding to the nodes specified in the .TTABLE.

Data statements may be defined in a subroutine contained within a .TTABLE. Subroutines are defined as follows:

```
(<label_name
<count>   <data 1> <data 2> ... <data n>
.
.etc
.
<count>   <data 1> <data 2> ... <data n>
)[<*n>]
```

(<label_name> defines a routine called <label_name>. The name must immediately follow the left parenthesis without spaces. Names may contain up to 16 characters.

)<*n> terminates the routine.

n specifies the number of times to execute the routine. If n is 0, the routine is defined but not executed. If n is omitted, the routine is repeated as many times as necessary until .MAXTIME is reached. If <*n> is omitted altogether, a default of *1 (one execution) will be used.

A previously defined routine is executed by placing a "call" within a .TTABLE as follows:

```
(<label_name>)*n
```

Subroutines may be nested to any level.

Examples:

```
.TTABLE    ABC    XYZ,1
   20       1      0
(SUB1
   10       1      1
  100       0      0
    5       1      0
)*5
(SUB2
   23       1      1
   11       0      1
(SUB3
    5       1      1
    2       0      1
    3       1      0
)*2
    2       0      0
)*3
(SUB1)*10
(SUB2)*20
```

Signal XYZ is initialized to 1 (implying it was defined as NRZ in its associated .TGEN).

Subroutine SUB1 is executed 5 times when defined and later called 10 times.

Subroutine SUB2 contains a further subroutine SUB3. SUB2 is executed 3 times when defined and later called 20 times.

Refer to section 3.3.3 for further information about the .TTABLE command.

6.5 .TPRINT

Defines the output signals which are to be printed to the .TPN, .TPS or .TPF files. Output is directed to filename.TPN, filename.TPS or filename.TPF as selected by the ZyPSIM .GUARDBAND statement.

Syntax:

.TPRINT <node 1> <node 2> ... <node n>

<node 1> through <node n> are the nodenames to be printed.

Only nodes which appear in .TGEN or .TSTROBE (both for I/O signals) are permitted. (Exceptions are the special keywords VDDn, VSSn, and UNUSEDn).

All signals which are device pins should appear in the .TPRINT statement in device pin order. Internal nodes should not be included.

Only one .TPRINT statement is allowed.

Examples:

```
.TPRINT            N1  N2  N3

.TPRINT   CLK RST UNUSED1 VSS1 QA QB QC QD VSS2 CARRY VDD
```

In the first example the default extension (.TPN) and default (.NET) filename, eg: CHIP.NET, will cause the output for nodes N1, N2, and N3 to be printed to file CHIP.TPN.

In the second example device pin #3 is unused, as signified by the keyword UNUSEDx. As required for final netlist coding the VSS and VDD pins are also included in this statement.

Refer to section 3.3.6 for more details about the .TPRINT command.

6.6 .TSTROBE

Defines the time at which an output specified in a .TPRINT statement will be printed.

Syntax:

.TSTROBE <strobe_time> <N 1>[,<m 1>] <N 2>[,<m 2>] .. <N n>[,<m n>]

<strobe_time> is specified in nS with respect to T0 and defines the time at which output signals are sampled and printed to the .TPN, .TPS or .TPF files.

<N 1> through <N n> define the nodes to which <strobe_time> applies.

<m 1> through <m n> represent optional mask control signals. When present, a mask signal value of "0" means the output is masked, "1" means it will be tested. The default (when masking data is not supplied) is "1".

Examples:

```
.TSTROBE  200    ABC     XYZ
.TSTROBE  100    ABC,MASK
```

In the first statement, signals ABC and XYZ are strobed 200nS after T0 and are always tested. Since masking signals are not supplied, a default of "1" is used for the mask signal.

In the second case, the strobe occurs 100nS after T0 and signal ABC will be tested whenever MASK is "1".

Refer to section 3.3.5 for more details about the .TSTROBE command and sections 4.7 and 4.8 for usage information.

6.7 .TPOC

Used to minimize the printing of output data by printing only when signals in the .TPRINT nodelist change state.

Syntax:

.TPOC

Arguments are not required or permitted.

Refer to section 3.3.7 for more details about the .TPOC command.

APPENDICES

A TIMING LIMITS

A.1 TEST AND SIMULATION RATE (.TPERIOD)

	SENTRY7	SENTRY20
MIN TPERIOD	100 nS	50 nS
MAX FREQUENCY	10 MHz	20 MHz

A.2 TIMING GENERATORS (.TGEN)

MIN <T1>	10nS
MAX <T2>	2 * .TPERIOD - 40nS
MIN <T2> - <T1>	10nS

A.3 STROBE TIMING (.TSTROBE)

OUTPUT SIGNAL ONLY	.TPERIOD - 20 nS
I/O SIGNAL W/O SINGLE CYCLE I/O	.TPERIOD - 20 nS
I/O SIGNAL WITH SINGLE CYCLE I/O	.TPERIOD - 25 nS *1
I/O SIGNAL WITH SINGLE CYCLE I/O	.TPERIOD - 60 nS *2

*1 When using high speed test head.
*2 When using low speed test head.

B GLOSSARY

Argument	The part of a command or statement that defines a a piece of information which is variable. Examples are filenames, nodenames and timing values.
Command	In ZyPSIM, any line starting with a "." in column 1.
Cycle Stealing	A condition which occurs when mask or direction registers are loaded and the rate of test is high. Causes increased memory requirements and possible undertesting.
Data Formatter	The part of the tester which changes raw data into a specific type of pulse.
Direction Register	A tester register which is used to indicate whether a particular device pin is an input or an output.
DUT	Device under test. The specific device being tested.
Mask Register	A tester register which defines whether a particular pin will be tested or not.
Pin Electronics	Tester hardware specifically associated with each pin, such as drive buffers and comparators.
Primitive	A "primitive" ZyPSIM element (such as AND, OR, INV etc.). Primitives may contain capacitance and/or timing data and are normally used to model ZyP library cells.

Single Cycle I/O	A bidirectional pin (I/O) which is switched from input to output (or vice-versa) without using an intermediate tri-state condition (dead cycle).
Statement	A collection of arguments which, together with a command, provide ZyPSIM with specific information regarding simulation execution.
Strobe	A tester generated pulse which is used to sample the output of a DUT versus expected data comparison.
TO	Pronounced Tee-zero. Indicates the starting point of each tester cycle.
Test Head	Located remotely from the tester to interface to the DUT. Contains the pin electronics.
Timing Generator	The part of the tester which generates timing information used to modify raw data before application to the DUT.
Vector	A string of data used to define a test or simulation pattern.

6

6

SPECIFICATION & TEST USER'S GUIDE

Doc: 20-010-010 Rev: C | Issued: August 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

Table of Contents

7

List of Illustrations

1 INTRODUCTION

1.1 SCOPE

This user's guide provides reference information and background data for ZyP device testing. Information is provided on tester hardware and the ZyP Device Specification.

1.2 OVERVIEW

Testing is an integral and important part of integrated circuit design. The ZyP Design System includes extensive facilities to aid in test program definition, pattern generation, and device specification.

ZyP testing begins with ZyPSIM. ZyPSIM provides for test pattern generation using an integral test language. Understanding the relationship between the ZyPSIM test language and the tester can be enhanced substantially by reading Sections 2 and 4 of this manual.

In addition to the ZyPSIM test language, ZyPSIM assists design for test with the following features:

a. Node toggle analysis.

b. Accurate circuit timing analysis.

c. Pattern generation for worst case and best case operating corners.

ZyP testing is further assisted with ZyTEST. ZyTEST performs pattern compression using techniques similar to those described in Section 4 of this manual. It also provides a comparison utility between ZyPSIM best case/worst case pattern files to aid in identifying potential test pattern problem vectors.

ZyP addresses automatic specification generation with ZySPEC. ZySPEC processes the ZyPSIM network file, a ZyPEE generated test file, and cell data bases to create a comprehensive hardcopy ZyP device specification. The ZySPEC generated specification is substantially similar to the specification described in Sections 5 and 6 of this manual.

1.3 RELATED DOCUMENTS

To aid in understanding ZyP test philosophy, resources, and utilities, as well as how testing is integrated into the ZyP system, the following documents are recommended.

20-010-009	ZyPSIM User's Guide.
20-010-209	ZyPSIM Reference Manual
20-010-309	ZyPSIM Test Language
20-010-310	ZySPEC User's Guide
20-010-020	ZyTEST User's Guide.

2 ABOUT THE TESTER

For many years the semiconductor industry's "standard" tester has been the SENTRY series, manufactured by Fairchild. Although the design is now well over 10 years old, it is still the most widely used tester for general purpose LSI and VLSI testing.

In this section the characteristics, capabilities and limitations of the test hardware and software will be described. Time spent in understanding the tester attributes will enable the user to avoid creating difficult to test or untestable circuit features.

2.1 GENERAL DESCRIPTION

SENTRY is a versatile test system, designed to operate in many modes to meet a variety of testing needs. The hardware is modular and customizable. A broad range of software is also available, to allow test program development and execution as well as extensive device characterization and data analysis.

In simple terms, the test system can best be described by the following components. They are shown in block form in Figure 2-1.

Computer with program and data memory.
Reference voltage supplies.
Device power supplies.
Precision measurement unit.
High speed controller.
High speed local memory and sequence processor.
Test station pin electronics.
Performance board.

The computer controls all of the functions involved in testing except the application of changing logic levels to the inputs of the device under test (DUT) and the checking of the resulting output levels. These functions operate in real time and are controlled by the Sequence Processor Module (SPM) with its own high speed local memory. DC testing is performed by the Precision Measurement Unit (PMU). The pin electronics module is the final link from tester to DUT via the performance board.

Each section of the tester will be discussed and those aspects which must be considered by the user are summarized.

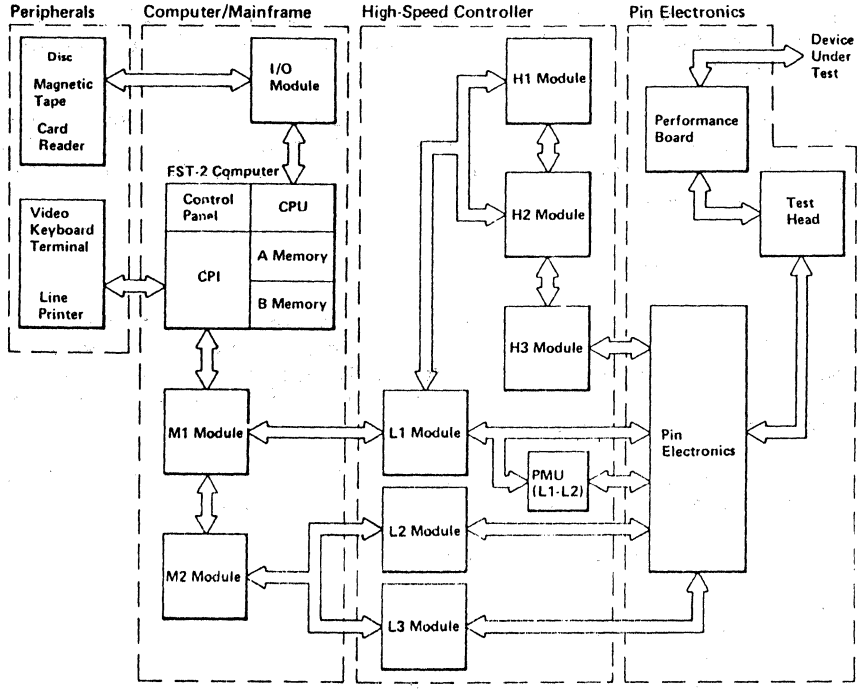


FIGURE 2-1. TESTER BLOCK DIAGRAM

2.1.1 COMPUTER

For the ZyP user, this portion of the test system is fairly transparent and consequently needs little explanation. It is a 24 bit word machine with CPU, memory and interface. Test programs are written using a high level language called FACTOR. The computer supports all normal peripherals and interfaces including terminals, tape, disk, RS232 and IEEE-488.

ZyMOS has also interfaced its ZyP System to the SENTRY test system computers to transfer processed ZyPSIM simulation output data in the form of the functional test pattern. Section 4 discusses ZyPSIM to SENTRY translation.
The computer has a memory cycle time of 1.75 uS, too slow to allow real time testing. Functional testing thus requires a higher speed portion of the test system, referred to as local or pattern memory.

2.1.2 REFERENCE POWER SUPPLIES

The reference voltage supplies (RVS) provide the one and zero logic levels to the drivers and comparators. The test program defines these values, which are in turn derived from the test specification supplied by the user. The functional test pattern ones and zeroes are applied as discrete voltage levels (VIL and VIH) by the drivers. VOL and VOH are the comparator reference levels against which the output will be checked.

Two sets of input reference pairs are available, allowing different data and clock input levels if required. Only one pair of output reference levels is available on the standard test head. See appendix 7.1 for specifications.

2.1.3 DEVICE POWER SUPPLIES

Three device power supplies (DPS) are available. These supplies provide programmable voltages and currents which may be used as power and bias for the DUT. Both magnitude and polarity are under software control. See appendix 7.2 for specifications.

2.1.4 PRECISION MEASUREMENT UNIT

The precision measurement unit (PMU) is an instrument that, under program control, can be connected to an individual pin of the DUT to make a quantitative voltage or current measurement at that point. This unit is used for DC parametric or DC GO/NO-GO testing. Parametric measurements are made by forcing voltage on a pin and measuring the resultant current. Alternatively, current can be forced and voltage measured. The ranges of voltages and currents available are shown in the PMU specification in appendix 7.3.

2.1.5 HIGH SPEED CONTROLLER

In addition to defining the voltage levels of the functional pattern, the timing must also be defined. This function is performed by timing generators within the controller. Eight timing generators are available, 6 for input signals and two for output strobe signals.

The functional pattern from the simulation is a snapshot of the DUT state at the desired output strobe time, when all data is valid.

NOTE: This also includes input data changes that occur during the clock cycle in which the strobe occurs. The ability to specify 6 different input timing relationships allows reconstitution of device timing from the "snapshot" provided by the logic simulation. The techniques for producing a simulation output consistent with the timing features of the tester are described in Section 4. The timing capabilities of the tester are shown in appendix 7.4.

2.1.6 HIGH SPEED REGISTERS

There are several high speed registers in the high speed controller. They control formatting and timing of local memory data input and output of the DUT. These registers provide selection on a per pin basis.

2.1.6.1 DA/DB Registers

The DA/DB registers are I/O definition registers that provide the ability to change the input/output definition of any DUT pin on the fly at the programmed test rate. For devices which use bidirectional I/O switching, the DA register can define the data direction when the I/O pins are inputs, with DB defining the pin data direction when the I/O pins are outputs. Programming a "1" into a register bit position defines the pin corresponding to that channel as an input (ie: driven by the tester), "0" defines an output. By enabling the appropriate DA or DB register during test execution, the device can accept data or be tested as an output (by also enabling the appropriate MA or MB register).

However, this allows for only 2 sets of data direction control. If several configurations are required, the DA/DB registers must be redefined during the test execution. See the Sequence Processor Module in this section and also Section 4 for more information.

2.1.6.2 MA/MB Registers

The MA/MB registers are mask definition registers that allow the changing of care/don't care conditions on any DUT pin on the fly. A care condition enables a defined output pin (defined by DA or DB) to be compared to the expected state stored in local memory. A don't care condition disables a comparison of a DUT output with any expected value. These registers allow all outputs to be masked while the DUT is in an undefined state and then, once the device has been initialized, to be switched on the fly to the specified input/output definition.

Programming a "1" into a MA or MB register bit, will cause the pin for the corresponding channel to be enabled for comparison (care condition). A logic "0" will disable all comparisons for that pin (don't care condition). Devices which do not require complex masking can define MA as all zeroes and MB as ones in the output channel bit positions. In this way, enabling MA at the start of test prior to initialization will disable all comparisons. Enabling MB when outputs are valid will cause comparisons to be made for the remainder of the test or until MA is enabled again. This is the simplest care/don't care control and requires only the selection of the MA or MB register, not the redefinition of the register contents. Register definition is discussed under REGISTER MODIFICATION in this section.

2.1.6.3 Data Formatting

The functional test data in local memory is unformatted. Data format hardware is used to reconstruct the complex timing needed to test. This simply means that the data is constant for each clock period. The data formatting capabilities of the tester are shown in Figure 2-2.

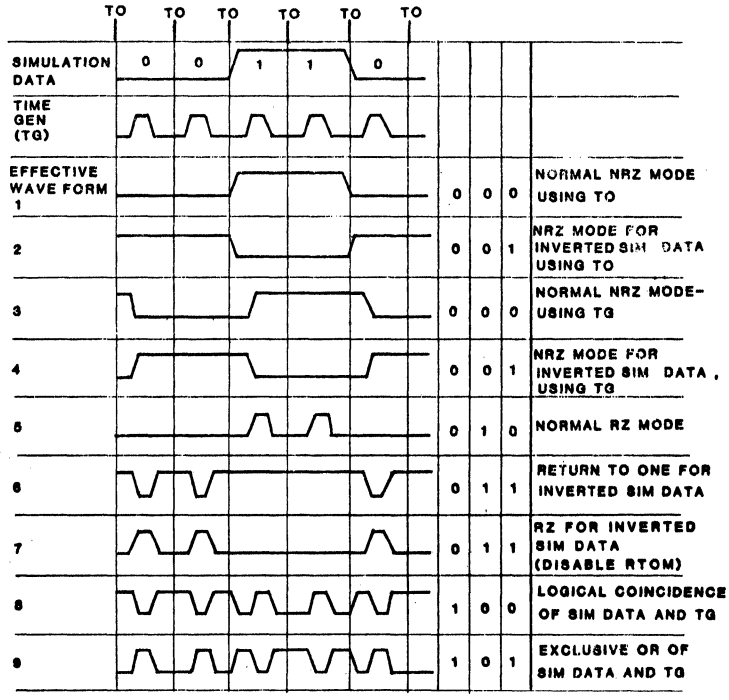


FIGURE 2-2. DATA FORMATTING WAVEFORMS

The many possibilties of data format coupled with the timing generator capabilities provide an extremely powerful means of input data generation.

2.1.7 SEQUENCE PROCESSOR MODULE (SPM)

Functional test patterns derived from the logic simulation are stored in high speed local memory. These test patterns can be applied to the DUT at clock rates up to 10MHz. The memory size allocated to the test pattern is 60 bits wide by 4096 words long. This allows patterns to be applied to a maximum of 60 pins simultaneously. The SPM contains additional storage of microcode (see Figure 2-3) which allows efficient re-use and modification of data already in local memory. In addition, on the Sentry Series 20, Model 120, the clock rate can be up to 20 MHz and 64K vectors in length.

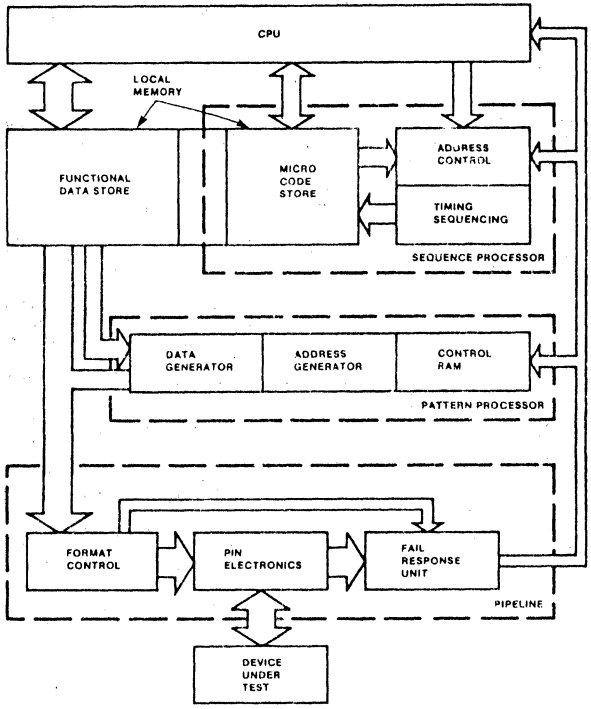


FIGURE 2-3. SEQUENCE AND PATTERN PROCESSOR

Normally, functional testing proceeds by sequentially applying test patterns (vectors) from each local memory location. However, the test execution sequence may be altered under software control by the SPM microcode as follows.

2.1.7.1 Clock Burst

A clock burst is basically a loop confined to one pattern at one memory location. The pattern may be repeated a specified number of times whenever the DUT requires repetitive data. Each functional pattern may be repeated up to 4097 times.

2.1.7.2 Subroutine Call

Any group of patterns in local memory may be defined as a subroutine. Additionally, a repeat or loop count may be assigned to the subroutine. Together with clock bursts, this provides an extremely powerful pattern compression tool. For example, assume that 3 test patterns each one applied to the DUT 1000 times and the overall group repeated 20 times. This would apply a total of 60,000 test vectors to the DUT. Using "straight line code", this would require 60,000 memory locations. However, local memory is limited to 4096 locations. By using the clock burst feature with a count of 1000 on the 3 test vectors, and defining the group of 3 as a subroutine with a repeat count of 20, only 3 local memory locations are required. Subroutines may be nested up to 16 levels deep to provide additional pattern compression.

2.1.7.3 Register Modification

Register load commands are performed by the sequence processor. The MA/MB and DA/DB registers previously described, are of particular importance here. The sequence processor can select either the A or B register on the fly, which allows the care/don't care and input/output definitions to be changed as the test progresses. However, when more than 2 sets of such conditions are required, the registers can also be redefined on the fly. This is subject to cycle steal which is explained in Section 4. The user must supply the control information for selection and definition of these registers.

2.1.8 TEST STATION PIN ELECTRONICS

The test station contains a test head assembly with test head pin electronics that form the final link between the SENTRY and the DUT. The pin electronics circuitry allows each pin to be programmed to any of the following types.

Data input pin.
Input/Output pin
Detector output pin.

Several types of data are received by the pin electronics (see Figure 2-4). The actual data sent to the DUT is fully software controllable. The functional test pattern digital data is applied to the DUT at the levels and timing defined in the program. The pin control logic defines which pins are inputs (driven), which pins are outputs (compared to expected data) and which pins are power.

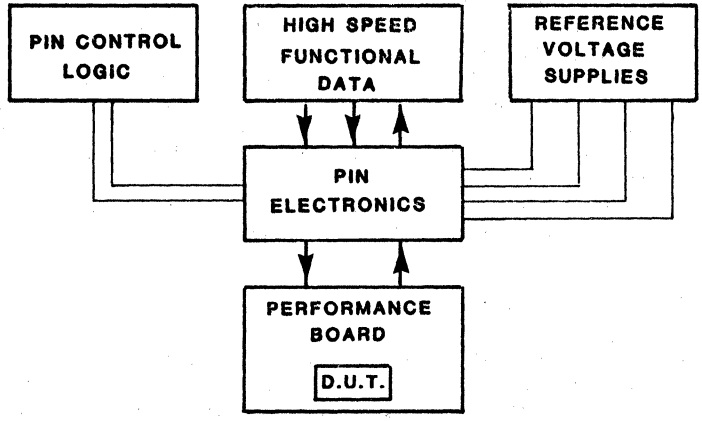


FIGURE 2-4. PIN ELECTRONICS FUNCTIONAL BLOCK DIAGRAM

2.1.9 INTERFACE HARDWARE

Interface hardware provides the means of connecting the device to be tested to the tester. The device may be in wafer form or be an assembled (packaged) part depending on the stage of test. Two main pieces of hardware are used, both of which are specific to the device under test.

2.1.9.1 Performance Board

The physical interface between the pin electronics and the DUT is via a performance board. This is a specially designed printed circuit board, which connects the pin electronics to a test socket for the DUT, while minimizing capacitance and noise problems. For testing of packaged parts, devices are inserted directly into the socket on the performance board.

This board is of significance to the user because it also allows the connection of external circuitry to the DUT. This will consists of load networks for the DUT outputs and relays for pin isolation and power supply connections. The load networks are specified in the ZyP Device Specification and more details can be found in that section.

2.1.9.2 Probe Card

When devices are tested at the wafer stage (wafer sort), a piece of hardware called a probe card is used. The probe card consists of a set of probes, mapped to the device bonding pad layout, mounted on a small printed circuit board. The probe card is mounted on a wafer sorter and is normally connected to the performance board via a special interface to minimize capacitance.

2.2 SUMMARY OF FEATURES AND LIMITATIONS

From the description of the components of the tester, it can be seen that there are several factors which will have a significant effect on the test program. The following is a summary of the key items which should be considered.

2.2.1 FUNCTIONAL TESTING

2.2.1.1 Pin Count

The pin electronics module has 60 channels on the standard test head enabling simultaneous testing of 60 pins. For devices with more than 60 pins the Sentry Series 20 Model 120 is available.

2.2.1.2 Pattern Length

The high speed local memory has a capacity of 4096 vectors. The functional test pattern can normally be compressed to meet this constraint. However, local memory can be loaded from disk during test execution when the pattern exceeds 4096 vectors.

It is important to realize that the local memory limit does not restrict the number of vectors which can be applied to the DUT. Clock bursts and subroutines allow infinite patterns, limited only by test time.

2.2.2 PARAMETRIC TESTING

Measurement of voltages and currents are subject to the accuracy limits defined in the appendices. The measurement time is dependent on the absolute value being measured. When measurements are to be made at a particular device state, the device must be pre-conditioned to that state and must be capable of maintaining the state for the required time. A typical value of 5mS to complete a parametric test should be allowed when PMU currents of >10uA are used. Check the PMU specification in appendix 7.3 for measurement time information.

2.2.3 POWER SUPPLIES

Up to three power supplies may be used to power the DUT. Standard connection through relays mounted on the performance board provide a maximum of 1 Amp per supply.

2.2.4 REFERENCE LEVELS

Two pairs of levels are available for inputs. Only one pair is available for outputs on the standard test head. Two output reference level pairs are available on a special high speed test head. Dual reference capability is required when TTL and CMOS I/O levels are used together on a single circuit design.

2.2.5 DEVICE TIMING

Input signals may have up to 6 different timing values. For example, Pin 1 input may start at time T, Pin 2 at time T + 50nS, Pin 3 at T + 100nS etc. Outputs may be strobed at two different points in the output cycle. For applications requiring more timing reference edges, an extended timing module (ETM) is available.

3 ZyP TESTING

3.1 OVERVIEW

Because of the nature of the ZyP system, (low development cost and fast turnaround time) the approach to testing ZyP circuits is unique. In order to introduce the testing ground rules to the ZyP user, a comparison of test considerations for conventional versus ZyP designs will be presented.

3.2 CONVENTIONAL FULL CUSTOM DESIGN

A typical full custom design for a medium size and complexity circuit takes from 9 months to 1 year to complete. From the time that the logic design is completed, the test engineer rarely has less than 6 months to develop a test program. Additionally, when the design engineer is producing the logic, consultation between test and design engineers ensures that potential test problems are eliminated. In many cases a breadboard is available, which allows the test program to be debugged prior to the availability of silicon.

3.2.1 TEST PHILOSOPHY

A conventional full custom design is almost without exception initiated due to a large volume requirement. Because of this, the test program is designed from the outset with the needs of high volume production testing in mind. Some of the considerations of this requirement are as follows.

3.2.1.1 Test Time

Test machine and operator time are extremely expensive. The test program must be written to take advantage of the tester features which offer maximum potential for minimum test execution time. Once the program is operational, considerable effort is made to further reduce the test time by refining the program.

3.2.1.2 Production Capability

To ensure that the device specifications can be met under the extremes of environmental conditions defined in the test specification, a technique known as guardbanding is used. This is a tightening of the test limits used under nominal environmental conditions, in order to guarantee performance at the extremes. This technique also reduces the machine dependency of the test. Devices may be tested on several different testers and the limits must reflect the limitations of the tester measurement accuracy.

Three types of testing are carried out for a production device: wafer sort, final test and Quality Assurance testing. The test program must be capable of handling all three types under software control and must have the test specification parameters modified by guardbanding to cater for all three.

3.2.1.3 Characterization

When first silicon is received, the product engineer must determine the sensitivities of the various device functions and parameters with respect to the variations of process parameters and test conditions. For example, a device may have a VIH specification of 2.0 volts and may pass the test program. But by how much? If the device were to fail with an input voltage of 1.95 volts, the chances of high yield in volume production would be slim.

To allow the product engineer to extensively characterize and evaluate the device, the test engineer must build a characterization facility into the test program.

3.3 ZyP FULL CUSTOM DESIGN

The time available to ZyMOS for test program development, is the time between logic network file transfer and prototype parts availability. This usually ranges from 8 to 16 weeks. The test engineer may have no knowledge of the circuit details or functionality of the part and normally no breadboard will be available to aid in test program debug. In the same time frame, the test hardware (performance board and probe card) must also be designed and manufactured.

3.3.1 TEST PHILOSOPHY

Since the production future of the part is largely unknown, an entirely different approach to test program design must be taken. The development time is not available to allow the type of program normally associated with a standard full custom design to be generated. However, the program must guarantee that:

a. The device performs functionally according to the logic simulation.

b. The device integrity is ensured by standard parametric tests and by the possible addition of some customer specified parametric tests.

c. Provisions are made to allow expansion to a full production test program should the device go into high production.

3.3.1.1 Test Time

At the prototype stage, test time is not nearly as important as in volume production. However, a test method which is intrinsically long and has no potential for reduction is not desirable. A long test time which enables fast program development is only permissible if it can be seen that significant improvements can be made given more development time.

3.3.1.2 Production Capability

Normally, prototype parts will only be tested at nominal temperature with specification limit adjustments to ensure operation under worst case conditions. Guardbanding and wafer sort/final test/Q.A. testing facilities are unlikely to be comprehensive due to time pressures. These features can be added when it is known that the device is to go into production.

3.3.1.3 Characterization

Prototype parts will be tested to basic functional and parametric test conditions. Minimal characterization will be done at this stage. Characterization is a time consuming process and will only be performed when it is known that the device is headed for high volume and such action would be justified.

3.4 GROUND RULES FOR ZyP TESTING

3.4.1 OVERVIEW

Having now discussed test philosophy and the tester itself, the requirements which must be met by a ZyP test program will be presented. This is first covered generally to establish a basis for the technical details presented in later sections.

The test program is a joint effort between ZyMOS and the customer who is the user of the ZyP system. The user must provide data required to generate the test program. This usually consists of two major items, the logic simulation output file(s) and the ZyP Device Specification. The logic simulation file(s) are the results of the network verification produced by the logic designer (user). The ZyP Device Specification allows a formal specification to be easily supplied by the user. The specification provides ZyMOS with all required data in a format suitable for fast test program development. Refer to the ZySPEC User's Guide, Document Number 20-010-310 for specification generation.

ZyMOS will generate the test program which includes: a functional test pattern derived from the logic simulation, standard parametric tests, and user defined parametric tests all of which are specified in the ZyP Device Specification.

3.4.2 FORMAL REQUIREMENTS

Throughout the ZyP Design System, every attempt has been made to place few, if any, restrictions on the user. However, at the transfer point from customer to ZyMOS, a procedure has been developed to formalize the transfer. The main transfer document is no more than a standard form called the ZyP FILE SUBMITTAL, a copy of which can be found in the ZyP Design System Introduction, Document Number 20-010-001. The ZyP Device Specification must also be part of the transfer. These two documents, with any specified attachments, are required before ZyMOS can proceed to generate prototype devices.

3.4.3 ZyMOS REQUIREMENTS

The summary of features and requirements described in Section 2 should be used as a guide line to what is technically feasible regarding testing. However, the following limits are normally applied.

a. 4K (4096) test vectors.

b. 5 second test time.

c. 4 user defined parametric tests.

Applications which exceed these limits are considered non standard and may require a specific quotation.

4 LOGIC SIMULATION FOR FUNCTIONAL TEST PATTERN GENERATION

4.1 OVERVIEW

The ZyPSIM simulation output file, which is used to verify the logic design by the user, is also used by ZyMOS as the functional test pattern to verify the actual device. Due to the dual purpose aspect of the simulation, there are a number of requirements which must be met in order that the simulation file can be converted into a SENTRY compatible test pattern. The requirements fall into two major categories:

a. The pattern memory limitations of the tester necessitate pattern compression techniques which place several timing requirements on the simulation.

b. The translation from ZyPSIM to SENTRY format is carried out by the ZyTEST translator software module. The ZyPSIM output must be formatted to be compatible with the input requirements of the translator.

These requirements along with the general subject of design for testability will be covered in this section. It is extremely important that the user produces a simulation that, not only verifies the logic, but also verifies that the device has no defects.

4.2 PATTERN COMPRESSION

Functional test patterns which fully exercise device logic can be anywhere from several thousand to hundreds of thousands of test vectors long. However, as previously mentioned, the high speed local memory is limited to 4096 locations. Pattern compression is therefore a very important part of the test program generation. The actual compression is carried out by the ZyTEST translator using the ZyPSIM output "straight line code" as the basic pattern.

To make the compression possible, the user must follow some basic rules when setting up the simulation conditions. To illustrate why this is necessary, consider the example circuit of Figure 4-1. Although this is a simple trivial circuit, it demonstrates the major problems (and solutions) related to translation and compression of functional test patterns.

4.3 TIMING REGENERATION

4.3.1 DATA FORMATTING

A timing generator can take raw data from the simulator and cause a transition to the appropriate level at a specified time according to various formats. Some of the more common formats (shown earlier in

Figure 2-2) will now be described. Reference to Figure 2-2 should be made in order to fully understand the importance of each formatting mode. The following section details the method of defining the required timing in conjunction with data formatting.

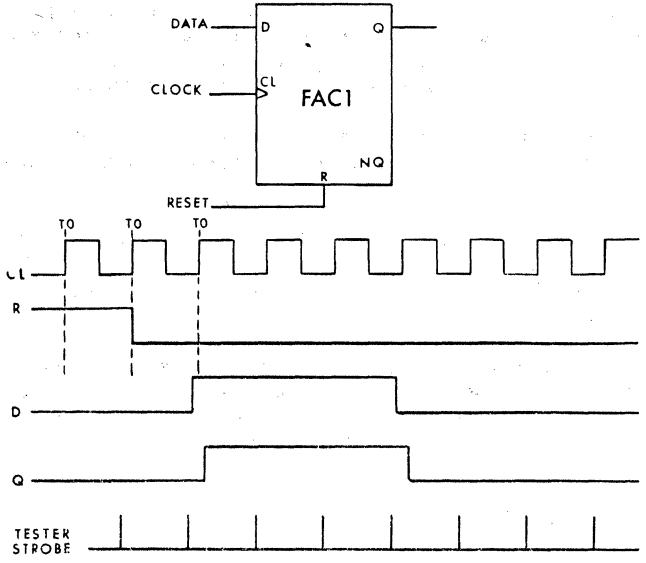


FIGURE 4-1 CIRCUIT TO ILLUSTRATE TEST PATTERN COMPRESSION

4.3.1.1 Normal NRZ Mode Using T0

This is the simplest form of data formatting, with essentially no formatting. The simulation data will be applied to the part by the tester directly. All transitions occur at tester T0 times. Refer to Figure 2-2, EFFECTIVE WAVEFORM 1. It is advantageous to utilize this mode whenever possible, since no TG is required. Remember, TG's are limited to 6 sets of timing for inputs.

4.3.1.2 Normal NRZ Format Using TG

As an example, consider a signal specified as NRZ format with a TG delay (from T0) of 50nS. A simulation state change from 1 cycle to the next would hold true for the whole cycle, eg: "0" in cycle 1, "1" in cycle 2. The signal as applied to the part would go to a "1" 50nS after the T0 edge in the cycle containing the "1". The signal would remain at "1" until a "0" transition occurs in the simulation. This would cause a "0" output transition 50nS after the T0 edge in the cycle containing the "0". This waveform is represented in Figure 2-2, effective wave form number 4.

4.3.1.3 Normal RZ Mode

RZ mode implies the use of a timing generator. This mode is used for clock pulse reconstitution. It allows a positive going pulse to be generated within a T0 cycle, even though the simulation data is constant for that cycle. In this mode, delay from T0 to the start of the pulse and also the pulse width must be specified. Whenever the simulation output contains a "1", a pulse with the specified timing will be generated for that cycle. The output will go to "1" after tdelay, remain at "1" for twidth, then return to "0". A "0" in the simulation will produce a "0" for the complete cycle. An example of this mode can be found in Figure 2-2, effective waveform number 5.

4.3.1.4 RO Mode

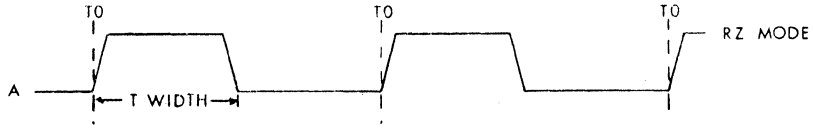
RO mode is similar to RZ mode except that the logic levels are inverted. It is used to generate negative going pulses. Whenever the simulation data contains a "0", the output will go to "0" after tdelay, remain at "0" for twidth, then return to "1". A "1" in the simulation will produce a "1" for the complete cycle. The RO mode is invalid for the "I/O" operation.

4.3.2 TIMING DEFINITION

Timing definition is no more than a summary of the the timing relationships (used in the simulation) in timing diagram format. If it is considered in this manner, the following procedure can be used to logically relate the tester timing to simulation/system timing.

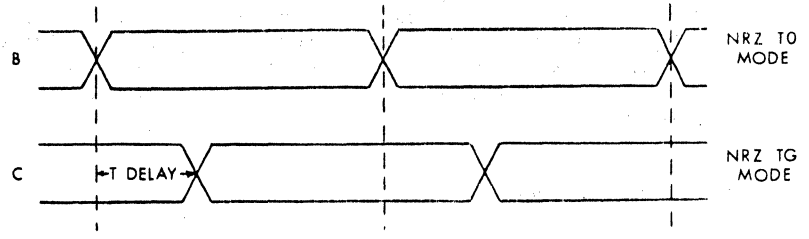
4.3.2.1 Master Clock (Step 1)

Define the start of a T0 period as the rising edge of the master clock signal. This will normally be the basic reference for all timing in the logic simulation and should be carried through to the tester timing.



4.3.2.2 Input Data (Step 2)

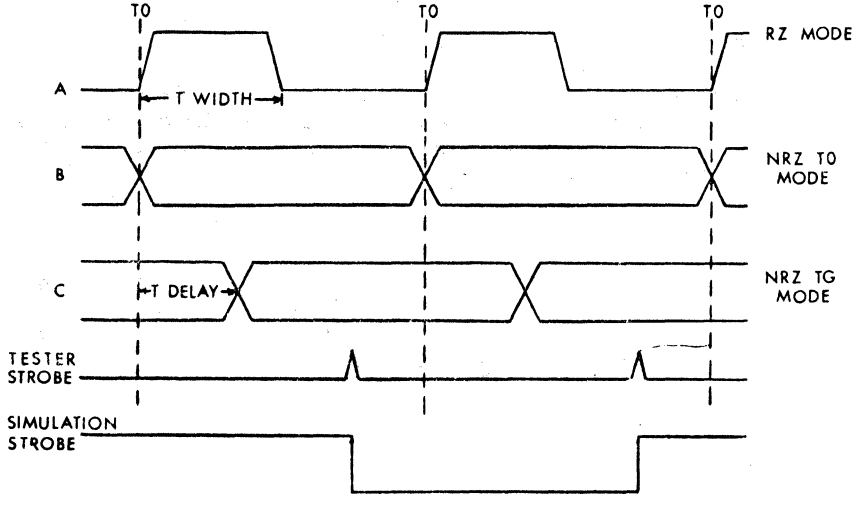
Define the timing of each input signal, using the .TGEN statement. There are six (6) different timing generators available.



NOTE: Signal B uses NRZ TO mode. If for example this were a data input being sampled/clocked by A, equal setup and hold times are applied.

4.3.2.3 Output Strobe (Step 3)

Set .TSTROBE to occur at a point when output data has maximum settling time.



4.3.2.4 Timing Diagram Reference Points

In the preceding diagrams, the T0 transition times have been exaggerated to show data transitions. Tester rise and fall times are extremely fast and transitions are concurrent with T0.

The ZyP Device Specification is used to specify the timing information and signal data formatting, in the form of tables and timing diagrams. This is discussed further in section 5.

4.4 SENTRY TRANSLATION

For the example pattern, consisting of 9 test vectors, the translated SENTRY pattern is shown in Figure 4-2. and now contains only 4 test vectors. A more complex example using register control is presented in the next section.


```
REM ... NEW BLOCK BEGINNING AT CYCLE     1
SET FC 2*1100;
SET F 0;
SET FC 3 0111;
SET FC 3 0100;
```

FIGURE 4-2 EXAMPLE CIRCUIT PATTERN AFTER SENTRY TRANSLATION

For further details regarding the ZyTEST translator, refer to the ZyTEST User's Guide.

4.5 SIMULATION GUARDBANDING

4.5.1 OVERVIEW

The ZyPSIM standard cell models contain timing data derived from worst case slow process parameters. However, the operating conditions for the models are nominal, ie 5 volts 27 degrees C. Simulation is normally performed at the nominal operating frequency of the device, resulting in the worst case slow process performance for nominal operating conditions.

To allow for other conditions, the simulation must be performed using the .GUARDBAND statement. (See the ZyPSIM User's Guide, document number 20-010-009 and the ZyPSIM Reference Manual, document number 20-010-209 for details). Most users will realize that when voltage is low and temperature is high, device speed will be reduced. What may not be so obvious, is that this has a major impact on testing. Futhermore, when the process is fast, test problems are just as likely to occur.

In this section, the subject of simulation guardbanding is covered, to ensure that problems do not occur due to either fast or slow speed operation. An example demonstrating these problems can be found in the ZyTEST User's Guide (document number 20-010-210) ZyTEST assists in the identification of potential test problems resulting from operating condition variations.

4.5.2 THE TESTING PROBLEM

Before proceeding to the examples, the test problem caused by device speed variations should be defined. The problem is known as "cycle slipping" and is often neglected, because at the system level it may not be a problem. In many cases, the exact clock cycle at which an output occurs is not critical in the application. However, the SENTRY cares! The device output is compared on a clock for clock basis with the expected data stored in memory. If the output from the device were to slip from one clock cycle to another, the device would fail the test program.

4.5.2.1 High Speed Device Failure (Fast Process)

This is the condition which is most likely to go undetected. The circuit of Figure 4-3 is used to demonstrate the problem. To allow for worst case fast process parameters, the .GUARDBAND fast condition must be used. This results in a higher device speed, and the models will be proportionally faster. A hand calculation method for determining the speed variation due to different operating conditions can be found in the appropriate design manual.

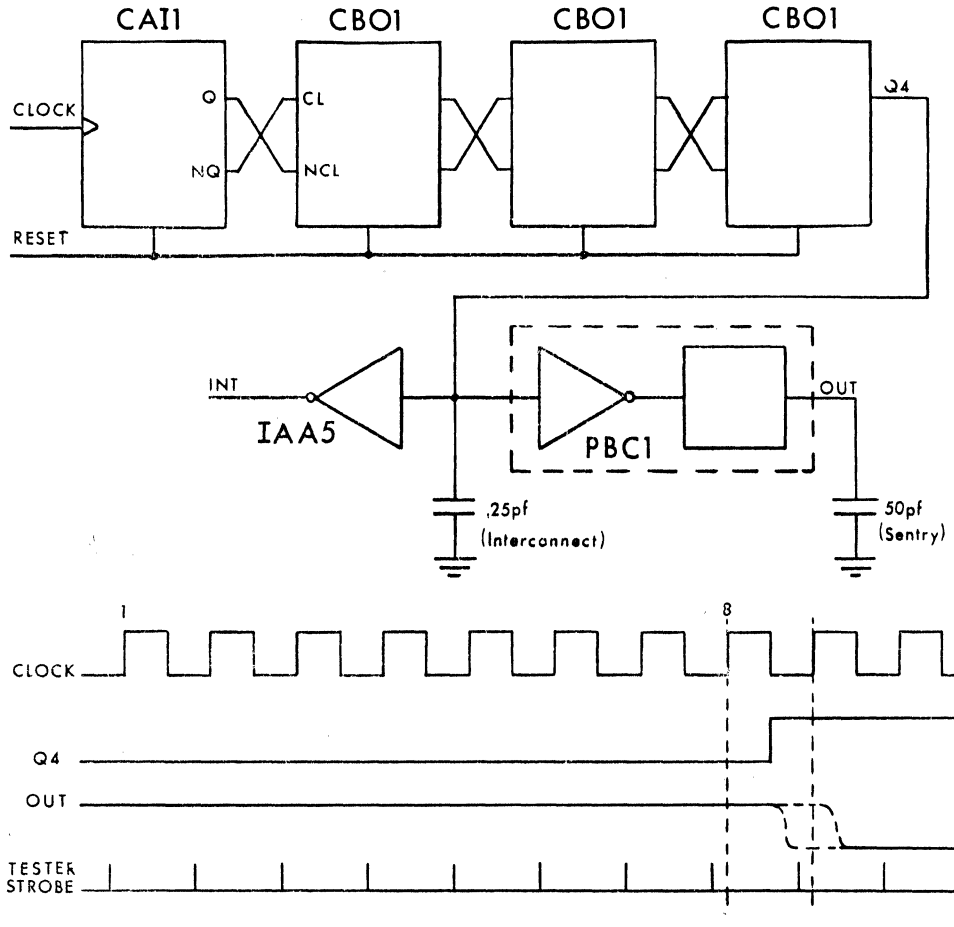


FIGURE 4-3 EXAMPLE CIRCUIT FOR CYCLE SLIPPING PROBLEM

The example circuit of Figure 4-3 uses a four stage ripple counter which is often a major cause of delay for this type of problem. Interconnect capacitance at Q4 output compounds the problem. Additional delay is caused by the output pad driver, loaded with a 50pF capacitor to allow for the SENTRY capacitance. Allowance should always be made for the SENTRY interface capacitance. The value varies somewhat and 50pF represents a good approximation.

The problem with this circuit is that cycle slip has already occurred under nominal conditions. The propagation delay through the ripple counter and output driver is such that the output occurs in the cycle following the cycle causing the output.

The correct procedure to resolve this problem is to design the logic such that cycle slip does not occur. In this circuit, the delay path is through 5 stages (4 counters + pad cell). Synchronization to clock edges could be employed to minimize delay (from clock) build up.

An alternative is to use the mask register to disable testing for the period of uncertainty, in this case 1 cycle. To control the mask register would require the generation of a ZyPSIM "window" signal during the uncertainty period. Although not as preferable as a design change, this is often an acceptable solution for some applications.

4.5.2.2 Failure at Extremes (Low VDD, High Temp)

Variations in circuit operation at the worst case slow operating conditions results in the more common failure mode. For this situation the additional delay resulting from low VDD and high temperature causes the output to slip past the strobe such that it does not appear until the next cycle.

4.5.3 GUARDBANDING SUMMARY

Simulation guardbanding is not limited to detection of cycle slip problems only. All types of design problems such as glitches, race hazards and marginal timing can be detected using this technique. For example, ripple counters with non synchronous decodes, which seem to operate perfectly, can develop all kinds of glitch problems when subjected to the rigors of guarbanded simulations.

Guardbanding is an essential requirement of design for test. Simulations must be performed with interconnect capacitance to ensure that performance is not affected by layout. These steps are required so the device can be guaranteed to function according to the test pattern derived from the simulation. The following procedures should be used to perform the above tasks.

4.5.3.1 Guardband Limits

To determine guardbanding factors the following procedure can be used.

a. Determine the extremes of temperature and voltage. Use the information presented in the appropriate technology design guide (Zy40000 or Zy50000) to obtain the delay derating factors for the required conditions. For example, consider a Zy40000 device that operates over the temperature range 0 to 70 degrees and a voltage range of 4.5 to 5.5 volts.

   The slow speed conditions are:

```
TEMPERATURE  = 70          DELAY INCREASE = 1.16
VOLTAGE      =  4.5        DELAY INCREASE = 1.2
                      TOTAL DELAY INCREASE = 1.4
```

   The high speed conditions are:

```
TEMPERATURE =  0          DELAY REDUCTION =  .88
VOLTAGE     =  5.5        DELAY REDUCTION =  .88
                    TOTAL DELAY REDUCTION =  .77
```

   In addition, the worst case fast process parameters result in a delay reduction to 0.25 of the worst case slow value. The overall reduction is therefore 0.2.

b. The outputs of the two guardband simulations should be compared against the original simulation to determine if problems exist. The simulation comparison feature of ZyTEST can be used to perform the comparison and report on any differences.

4.5.3.2 Interconnect Capacitance

It is necessary to consider the capacitance of the interconnect since this can significantly affect device performance. Ultimately, accurate values can be obtained from the completed layout. ZyMOS will always provide a capacitance file to the user as a result of a final routing. However, it is desirable to have this data as early as possible in the design cycle in order to detect potential problems. For this purpose, an estimate of the interconnect capacitance prior to routing can be obtained from ZyPSIM.

An algorithm based on fanout and emperical data is used to produce a value for interconnect capacitance on each node. It must be emphasized that some nodes may have considerable variation from the estimated value when final data is known. For example, a node which drives five inputs will always produce more capacitance than a node driving one input. In practice the five inputs may be close together and also close to the driving node, whereas the single driving node and input may be on opposite sides of the chip. However, signals such as master reset which usually run all over the chip will be more accurate. For information on using the capacitance estimation feature refer to the ZyPSIM User's Guide in the Routing Capacitance section.

4.5.3.3 ZyPSIM Capacitance Summary

A detailed summary of the capacitance on each node is available from ZyPSIM. This is obtained by including the command .XREF in the net list. A complete node by node cross reference will be produced in filename.XRE where filename is the name of the .NET file. Fanout, timing, and capacitance data will be contained in the .XRE file.

The total capacitance on each node will be shown as well as a breakdown of each type: cell input, estimated interconnect and CAP primatives. CAP primatives include those entered directly or C* artwork generated values.

4.6 INITIALIZATION

4.6.1 OVERVIEW

In generating the functional test pattern it is essential that the device can be fully initialized. Furthermore, this must be achievable by the application of signals (from the tester) to the device pins. Since the test pattern is derived from the simulation, it is the simulation which must contain the initialization procedure. Logic simulators traditionally allow the use of an initialization statement to set/reset the logic level of any node. However, ZyPSIM does not permit this to ensure that initialization problems are addressed during the design phase. This feature requires the user to apply a sequence of signals which are able to guarantee the logical state of the device.

To illustrate how the simulator can predict this condition the circuit of Figure 4.4 is used.

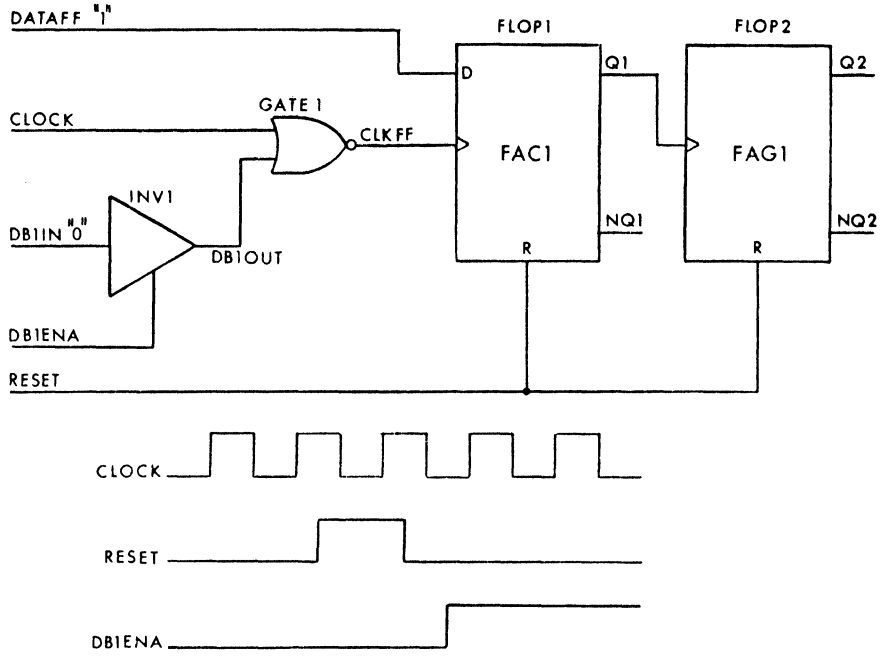


FIGURE 4-4 INITIALIZATION EXAMPLE CIRCUIT

4.6.2 EXAMPLE CIRCUIT DESCRIPTION

FLOP1 is a D type flip flop which is clocked by a gated signal CLKFF. The gating signal is DB1OUT which is provided by the data buss signal DB1IN when the tri-state buffer INV1 is enabled by DB1ENA. When FLOP1 receives a clock causing Q1 to make a positive transition, the (toggle) flip flop FLOP2 will be toggled. The net list is shown in Figure 4.5.

```
INITIALIZATION EXAMPLE
*
*
*
INV1    IAE1    DB1ENA    DB1IN    DB1OUT
GATE1   GAB1    DB1OUT    CLOCK    CLKFF
FLOP1   FAC1    DATAFF    CLKFF    RESET    Q1    NQ1
FLOP2   FAG1              Q1       RESET    Q2    NQ2
*
*
*
.TGEN     RO      100      200     CLOCK
.TGEN     NRZ      50               RESET
.TGEN     NRZ      50               DBIENA
.TGEN     NRZ       0               DBIIN
.TGEN     NRZ       0               DATAFF
.TPERIOD  200
.TTABLE    CLOCK    RESET    DBIENA    DBIIN    DATAFF
   1         0        0         0         0         1
   1         0        1         0         0         1
   3         0        0         1         0         1
*
*
*
.POC
.PRINT DB1ENA/DB1OUT/CLOCK/CLKFF/RESET/Q1/NQ1/Q2/NQ2
.MAXTIME=1000
.GO
.END
```

FIGURE 4-5 NET LIST FOR CIRCUIT OF FIGURE 4-4

4.6.3 SIMULATION INPUT SIGNALS

The master CLOCK is a 5MHz continuous signal which is inverted and applied to FLOP1 when enabled by DB1OUT. The RESET signal is applied at time 250 until time 450 to initialize the circuit. DB1IN data buss is switched to DB1OUT at time 550 by DB1ENA. The DATAFF data signal for FLOP1 is tied to logic "1" and DB1IN is tied to logic "0" in order to cause a clock to the toggle stage FLOP2 when GATE1 is first enabled.

4.6.4 SIMULATION OUTPUT ANALYSIS

As expected, the flip flop outputs are undetermined at time zero. Also as expected, DB1OUT is disabled and is in a high impedance state. When the reset signal is applied, both flip flops are reset after some propagation delay. However, note that when reset is removed both flip flops become undetermined again. This is caused by signal CLKFF which is still undetermined, since the high impedance state is propagated through GATE1 as an undetermined condition.

TIME	DB1ENAT	DB1OUT	CLKFF	CLKFFT	RESET	Q1	NQ1	Q2	NQ2
0	0	Z	1	0	0	X	X	X	X
100	0	Z	0	0	0	X	X	X	X
103	0	Z	0	X	0	X	X	X	X
200	0	Z	1	X	0	X	X	X	X
202	0	Z	1	0	0	X	X	X	X
250	0	Z	1	0	1	X	X	X	X
257	0	Z	1	0	1	X	X	X	1
258	0	Z	1	0	1	X	1	X	1
259	0	Z	1	0	1	X	1	0	1
261	0	Z	1	0	1	0	1	0	1
300	0	Z	0	0	1	0	1	0	1
303	0	Z	0	X	1	0	1	0	1
400	0	Z	1	X	1	0	1	0	1
402	0	Z	1	0	1	0	1	0	1
450	0	Z	1	0	0	0	1	0	1
500	0	Z	0	0	0	0	1	0	1
503	0	Z	0	X	0	0	1	0	1
507	0	Z	0	X	0	0	X	0	1
517	0	Z	0	X	0	X	X	0	1
521	0	Z	0	X	0	X	X	0	X
527	0	Z	0	X	0	X	X	X	X
550	1	Z	0	X	0	X	X	X	X
554	1	0	0	X	0	X	X	X	X
557	1	0	0	1	0	X	X	X	X
600	1	0	1	1	0	X	X	X	X
602	1	0	1	0	0	X	X	X	X
700	1	0	0	0	0	X	X	X	X
703	1	0	0	1	0	X	X	X	X
711	1	0	0	1	0	X	1	X	X
714	1	0	0	1	0	0	1	X	X
800	1	0	1	1	0	0	1	X	X
802	1	0	1	0	0	0	1	X	X
900	1	0	0	0	0	0	1	X	X
903	1	0	0	1	0	0	1	X	X

FIGURE 4-6 SIMULATION OUTPUT FOR NET LIST OF FIGURE 4-5

ZyPSIM will always produce an undetermined output for flip flops when the clock is undetermined and the D input is not at the same state as Q. It could be argued that this is incorrect but a little analysis will show otherwise. The "Z" condition of DB1OUT is actually a floating node and the possibility of spurious clocks being applied to CLKFF is extremely high. Under these conditions, the flip flop output is a legitimate unknown. If a clock occurred, the output would change state (DATAFF is "1"), whereas if a clock does not occur the flip flop state will be unchanged.

The enable signal DB1ENA is not applied until time 550 which allows the clock to be undetermined after the reset signal is removed. This causes both flip flops to be undetermined. When the clock becomes determined (after time 550), FLOP1 receives a clock and Q becomes a valid "1". However, the toggle flop FLOP2 can never recover unless a reset signal is applied. To solve this problem, the input signals should be such that the "Z" condition does not occur after reset is removed. A longer reset pulse or earlier application of DB1ENA would be an effective solution.

4.6.5 SIMULATION OUTPUT WITH CORRECTED INITIALIZATION

Figure 4.7 shows the simulation output after modification of the initialization procedure. As can be seen, the clock state is now determined before removal of reset. This was accomplished by enabling signal DB1ENA at the earlier time of 350.

TIME	DB1E N	DB1O UT	CLOCK	CLEAR F	RESET F	Q1	NQ1	Q2	NQ2
0	0	Z	1	0	0	X	X	X	X
100	0	Z	0	0	0	X	X	X	X
103	0	Z	0	X	0	X	X	X	X
200	0	Z	1	X	0	X	X	X	X
202	0	Z	1	0	0	X	X	X	X
250	0	Z	1	0	1	X	X	X	X
257	0	Z	1	0	1	X	X	X	1
258	0	Z	1	0	1	X	1	X	1
259	0	Z	1	0	1	X	1	0	1
261	0	Z	1	0	1	0	1	0	1
300	0	Z	0	0	1	0	1	0	1
303	0	Z	0	X	1	0	1	0	1
350	1	Z	0	X	1	0	1	0	1
354	1	0	0	X	1	0	1	0	1
357	1	0	0	1	1	0	1	0	1
400	1	0	1	1	1	0	1	0	1
402	1	0	1	0	1	0	1	0	1
450	1	0	1	0	0	0	1	0	1
500	1	0	0	0	0	0	1	0	1
503	1	0	0	1	0	0	1	0	1
507	1	0	0	1	0	0	0	0	1
515	1	0	0	1	0	1	0	0	1
519	1	0	0	1	0	1	0	0	0
522	1	0	0	1	0	1	0	1	0
600	1	0	1	1	0	1	0	1	0
602	1	0	1	0	0	1	0	1	0
700	1	0	0	0	0	1	0	1	0
703	1	0	0	1	0	1	0	1	0
800	1	0	1	1	0	1	0	1	0
802	1	0	1	0	0	1	0	1	0
900	1	0	0	0	0	1	0	1	0
903	1	0	0	1	0	1	0	1	0
1000	1	0	0	1	0	1	0	1	0

FIGURE 4-7 SIMULATION OUTPUT WITH CORRECTED RESET

5 THE ZyP DEVICE SPECIFICATION

5.1 SCOPE

This section describes the components of the standard ZyP Device Specification. It is intended for reference only; ZySPEC should be used for actual device specification generation.

5.2 OVERVIEW

The ZyP Device Specification has two main functions:

a. To define requirements.

b. To provide in a single document, a complete specification for the device. This includes Test, Assembly, Packaging and Q.A. requirements.

The ZyP Device Specification is a formal specification document required for all ZyP devices. ZyP device specifications have been integrated into the ZyP System with the ZySPEC software utility. However, the following sections are intended to explain the components of the ZyP specification and provide examples. Occasionally, specific items in the following discussion may differ from those presented in ZySPEC. An example may be the list of available packages. When differences occur, always use ZySPEC as the most up to date source of specification data.

5.3 GENERAL INFORMATION

5.3.1 DEVICE IDENTIFICATION

A ZyP device can be identified in two ways; by a customers designation and by a ZyMOS assigned part number.

5.3.2 RELEVANT PERSONNEL

The first section at the top of the front page identifies technical and purchasing individuals who can be contacted to discuss device issues.

5.3.3 REFERENCED DOCUMENTS

Any documents referenced in the Device Specification must be physically attached. Attached documents are valid only for the section referenced. For example, if the Q.A section referred to Doc # ABC123 and that document covers electrical device limits, the ZyP Device Specification device limits are still effective unless that section also specifically refers to DOC # ABC123.

When such documents are referenced they will be considered part of the specification, subject to acceptance.

5.3.4 ADDITIONAL DOCUMENTATION

Although not part of the formal specification, a ZyP format logic schematic and a device functional description are desired. These are of assistance to ZyMOS in understanding the functional test method and also to aid in layout during artwork generation.

5.4 PINOUT INFORMATION

Section 1 describes the basic information for each pin of the device in the form of a table. It is a general section which refers to other sections containing the detailed information. Each column of the table is covered in detail.

5.4.1 SIGNAL NAME

The names assigned should describe the pin function in any manner convenient to the user. However, names must be consistent with any other occurrences such as logic schematic, net list, device description etc.

Because of character limitations or length considerations, names cannot always be entered via software to agree with customer identification. The name in the netlist will be the name on the specification.

5.4.2 SIGNAL TYPE

This column should be marked I, O, or I/O to describe whether the pin is an input, output, or bi-directional input/output pin.

NOTE: Pins should also be marked TO if they have tri-state outputs, even though they do not have input capability.

5.4.3 TIMING REFERENCE AND FORMAT TYPE (TREF FMT)

This column is applicable only to input pins. Formats can be defined as any type in Figure 2.2 but will typically only be NRZ or RZ. Timing references are defined in Table 5 of the specification.

5.4.4 STROBE REFERENCE

The .TSTROBE statement is reflected here and defined in Table 5 of the specification.

5.4.5 INPUT AND OUTPUT LEVELS (VIL VIH VOL VOH)

This column reflects the input and/or output references to be used for the device. The references are defined in Table 2 of the specification.

5.4.6 FUNCTIONAL VOLTAGE PARAMETERS

This table defines the voltage references used in Table 1, the PIN OUT information. These levels are generated to match the ZyPSIM simulation of the device.

5.4.7 OUTPUT LOADING

Table 3 is a list of the device pins and the load references that will be used to test each output. These load references are defined in Section 6 of the specification, standard loads.

5.5 TEST CONDITIONS

The .GUARDBAND statement is reflected in this table. These are for functional operation only.

5.6 FUNCTIONAL TEST

Section 3 of the Device Specification defines the simulation output files which will be used to generate the functional test pattern.

5.6.1 REQUIREMENTS

To generate a functional test the chip.TPN file is required. Refer to the ZyPSIM Test Language User's Guide, document number 20-010-309, for more information.

5.6.2 TEST PATTERN INTEGRITY

It is important to realize that the responsibility for test pattern integrity is with the customer. ZyMOS assures that the device will agree with the simulation. The customer must ensure that the simulation exercises the device logic to guarantee performance in the system. If the simulation cannot detect a failure mode which cause problems in the system, the test program will not detect it either.

5.6.3 AC CHARACTERISTICS

The dynamic performance of output signals such as propagation delay and risetime are not explicitly measured. Although the tester is capable of this type of measurement, many iterations of the functional pattern are required and the test time is prohibitive in a production environment.

However, this does not mean that dynamic performance cannot be tested. Refer to the section concerned with test hardware for a detailed description of how this is achieved.

5.7 TIMING DEFINITIONS

Section 4 of the Device Specification reflects the .TGEN statements of the chip.NET file and is used to define the timing references from Table 1 of the specification.

5.8 PARAMETRIC TESTING

Table 5 describes the conditions and limits for parametric testing. ZyMOS will always include the indicated tests as standard. These are tests considered necessary to ensure the basic integrity of the device.

If the user requires additional tests they can be requested using the expanded Table 5 which is located in Section 5 of this guide.

5.8.1 GENERAL INFORMATION

Before describing the parametric table and the standard/user parametrics, some general background and global information will be described. It is important that the user understand the limitations involved.

5.8.1.1 Determining Parametric Test Limits

Parametric testing is performed by the Precision Measurement Unit (PMU) of the tester. Measurements are made by forcing a value to a pin and determining the response. The PMU has several ranges available dependent on the absolute value to be measured. The chosen range determines the resolution and accuracy with which the measurement can be made. The PMU specifications are shown in appendix 7.3.

This must be considered by the user when specifying a parametric test. For example, suppose it is required to measure a current of nominal value 2uA. As can be seen from the current measuring range specification, range 1 is the lowest range that could be used (0 to 102uA) and this has a resolution of 100nA with an accuracy dependent on the measured value of +/- .15% +/- 200nA. Obviously a specification of 2uA +/- 50nA is not acceptable since the basic resolution on this range is 100nA. Assuming that 2uA +/- 100nA is the specification, devices which passed this test could in fact measure anywhere between 1.797 and 2.203uA when the accuracy of the measurement is considered.

The pass limits should therefore be carefully considered in regard to the absolute value and the PMU specification. ZyMOS will always select the range which offers the best accuracy for the test. The user must select the limits which take into account the accuracy of the tester.

5.8.1.2 Keyletters

To define the condition of a pin, keyletters are used in conjunction with a pin table (see Section 5.7.5). The definitions of the keyletters is as follows:

* G (GROUND) The pin will be connected to the VSS supply of the device.

* O (OPEN) The pin will be disconnected from all tester hardware and will be floating.

* T (TEST) The pin will be connected to the appropriate voltage/current source defined in the table 5.

* N (NORMAL FUNCTIONAL CONDITION) The pin will be driven from the test pattern with the normal voltage levels defined for the functional test.

5.8.1.3 Default Values

For the standard tests, ZyMOS has supplied the force (applied) and pass limit (measured) values appropriate to the technology in the form of default values, which have been chosen such that device integrity is ensured values. Do not specify any values for these tests unless there is a good reason for the change. An exception to this would be supply current for low power designs.

5.8.2 PARAMETER TABLE DEFINITION

5.8.2.1 Test Type

This is simply a name to indicate the general type of test being performed, eg: breakdown, leakage, etc. Names for user defined tests will probably be more specific and can refer to a single signal or to groups of signals as well as the test type.

5.8.2.2 Vector Number

Some tests require that the device is preconditioned (set to a specific logic state) before the test can be performed. For this type of test, the vector number in the functional test pattern at which to perform the test should be entered in this column of the table.

For example, if the leakage of an open drain output was to be a parametric test, the device would first have to be clocked to a state which caused the output to be in the off condition. The lowest vector number at which this occurs in the functional pattern should be used for the test.

5.8.2.3 Power

The next two columns allow the VSS and VDD pins to be defined individually for each test. They will usually be at their normal levels but can be specified appropriate to the test being performed.

5.8.2.4 Pin Table

This column is used to refer to a pin table which defines the state of each pin using the keyletters already described. Any number of tests may reference the same pin table if the conditions are the same. Pin tables for the standard ZyMOS tests are not required. See the pin table section (5.7.5) for details.

5.8.2.5 Force Value

This column specifies the value of voltage or current which will be supplied to the pin(s) under test by the PMU. One or two values may be supplied (corresponding to the low and high pass limits) dependent on the type of test. All pins which are labeled "T" in the pin table specified for the test will receive the value(s).

5.8.2.6 Pass Limit

The pass limit column allows two values to be specified. They are the low and high values which are the acceptable results of the test when the voltage/current specified in the force value column is applied.

5.8.3 STANDARD TESTS

These tests are performed before the functional tests.

5.8.3.1 Continuity and Shorts

This is always the first test performed and is designed to detect gross failures due to open or short circuits. Current is forced into each pin and the voltage developed is measured. The limit values are such that the presence of the bonding wire (or wafer probe) connectivity is assured. Also pin to pin or pin to ground shorts are detected.

5.8.3.2 Input Breakdown

The input breakdown test detects the presence of the input protection network. Without the protection network, the device is susceptible to static destruction caused by handling etc.

5.8.3.3 Input Leakage

This test forces the specified values (normally VDD and ground) to all input pins and measures the resultant leakage currents flowing into and out of the device. The test verifies the integrity of the gate inputs.

5.8.4 USER DEFINED TESTS

Dependent on the type of device, further parametric testing may be required. For example, special inputs where the standard tests are not possible, tri-state output leakage, and analog functions. Parametric testing is not used, however, to measure DC drive capability of output pins. This is measured dynamically under functional test conditions. See Section 5.9 for more information on this subject.

5.8.4.1 Tri-State Testing

The high impedance off state of tri-state outputs is tested parametrically. It is possible in some cases to guarantee this during functional test but this has potential problems which make parametric testing the preferred method.

Outputs are connected to load networks on the performance board. When the device is switched to the off state, the output voltage will depend on the load network. If this is a resistor to VDD/2 (typical for a symmetrical drive test), the output voltage will go to VDD/2. This is not compatible with the VOH and VOL output levels. Even if the resistor were returned to VDD or ground, ensuring a compatible level, the capacitance (at least 50pF) present at the output will form a RC time constant with the load resistor. For all but the slowest (clock speed) devices, this will almost certainly exceed the clock period. Thus the output level will not have reached a valid state at strobe time.

During functional test the normal method of treating tri-state outputs is to use the mask register to disable comparisons when the output is in the high impedance state. Subsequently (after functional testing is completed), a parametric test is performed to measure the leakage in the high impedance state. The user must therefore provide the mask register control information during the functional test and the vector number at which the output is tri-stated for the parametric test.

5.8.5 PIN TABLE

The pin table is a universal table which allows flexibility in defining the conditions for a parametric test. Some tests can be easily defined whereas others such as analog functions require the use of external networks to perform the test. The table is designed to allow data, diagrams and comments to be mixed on one sheet to fully define the conditions for the test. Keyletters are also used to simplify the definition.

As an example, a CMOS device with tri-state outputs is presented. The partial parametric table is shown in Figure 5.1.

TEST	VEC NUM	POWER		PIN TABLE #	FORCE VALUE		PASS LIMIT		
		VSS	VDD		Act	Def		Act	Def
TRISTATE TEST #1	123	G	5.0V	1	0.0V 5.0V		L H	-5uA 5uA	
TRISTATE TEST #2	456	G	5.0V	1	0.0V 5.0V		L H	-20uA 20uA	

FIGURE 5-1 EXAMPLE PARAMETRIC TABLE

5.8.5.1 Tri-State Parametric and Pin Table Example

Two of the outputs, pins 3 and 4, can be placed in the off state at vector number 123. The other two outputs, pins 7 and 8, can be placed in the off state at vector number 456. For pins 3 and 4 the force value of 0v checks that a maximum of 5uA flows out of the pin under test (-5uA limit). The force value of 5v checks that a maximum of 5uA flows into the pin under test (+5uA limit). Pins 7 and 8 have higher limits due to larger output transistor sizes.

The fact that two separate tests have been defined (because of the two vector numbers and two sets of limits) does not mean that two pin tables must be defined. The pin table column shows pin table #1 referenced for both tests. The partial pin table of Figure 5.2 shows how this can be achieved.

REF PIN #	ACT PIN #	TSTR CHAN	PIN CONDITION	REMARKS AND/OR EXTERNAL NETWORKS
1			G	
2			N	
3			T/O	NOTE: PINS 3 AND 4 ARE
4			T/O	TESTED BY TRISTATE TEST #1
5			N	
6			N	
7			T/O	NOTE: PINS 7 AND 8 ARE
8			T/O	TESTED BY TRISTATE TEST #2
9			N	
10			O	

FIGURE 5-2 EXAMPLE PIN TABLE

The four output pins are shown along with several other pins. All pins which affect the ability of the functional pattern to place the output pins into the off state at the specified vector number should be shown as "N". These pins will then receive the normal functional pattern. Other pins are shown as "G" or "O" as appropriate. Inputs not required for this test would be shown as "G" (grounded) or connected to a bias voltage (such as VDD). Outputs not being tested would be left "O" (open).

For the pins to be tested "T" is used, meaning that the specified force values will be applied to the pin when the test is performed. Actually, since the PMU can only test 1 pin at a time, the symbol "T/O" is used. This indicates that when the PMU is not testing that pin, it is left open circuit. "T" can be used in conjunction with other symbols/voltages whenever multi pin testing is performed. Alternatively, a comment could be added to the pin table such as: "pins are open circuit when not tested". Since there are two groups of pins, comments are used to define which group corresponds to which test. Pins 3 and 4 are tested at vector number 123 and pins 7 and 8 are tested at vector number 456.

5.8.5.2 Example Parametric Test Flow

Since the PMU can only test 1 pin at a time, the test flow for this example would be as follows.

a. The device is exercised by the pattern until vector 123 is reached to set the outputs of pins 3 and 4 to the high impedance state.

b. The PMU is connected to pin 3, 0v is forced and the current supplied by the PMU is measured. This must be less than the -5uA limit in order to pass. Next, 5v is forced and the +5uA limit is checked. The PMU is then connected to pin 4 and the above tests are repeated for that pin.

c. The device is clocked again until state 456 is reached.

d. The tests in 2) are performed on pins 7 and 8.

5.9 GENERAL REQUIREMENTS

Sections 6, 7 and 8 of the device specification define ZyMOS standards for device limits, packaging and Q.A. For commercial applications, the user need only provide the package type to fully define the specification. If the default standards are not adequate for the application, the user may reference other documents to be included as part of the specification subject to acceptance by ZyMOS.

5.9.1 STANDARD TEST HARDWARE

This section of the Device Specification shows the load networks use to insure IOL and IOH of the output cells.

5.9.2 DEVICE LIMITS

This section of the Device Specification covers the absolute maximum ratings of the device. The CMOS and NMOS versions of the device specification differ slightly in this area, the (absolute) maximum voltage ratings are higher for NMOS. More important, however, the operating temperature range for NMOS devices is subject to a maximum junction temperature limitation of 150 degrees C. The junction temperature for CMOS is approximately equal to the ambient temperature so no restrictions other than the ambient operating range apply.

5.9.3 BURN-IN OR LIFE TEST DIAGRAM

Section 7 defines the burn-in pin list. This is only used if burn in is required by the user.

5.9.4 PACKAGING

For standard ZyP designs, dual in line or leadless chip carrier packaging is offered. A matrix of packages available versus pin type is shown in Section 7 of the Device Specification. Checking the box for the required package defines the package outline spec. The assembly flow specification can be selected from those offered. Both specifications can be found in the packaging section of the ZyP Specification and Test User's Manual.

5.9.5 QUALITY ASSURANCE

Section 8 of the Device Specification allows the Q.A requirements for the device to be defined. This can be a ZyMOS standard procedure or an acceptable customer defined procedure. The standard ZyMOS Q.A procedures are shown in the Q.A section of the ZyP Specification and Test User's Manual.

6 ADDITIONAL TEST SPECIFICATION FORMS

6.1 OVERVIEW

This section presents the forms necessary for adding special tests and their additional cost to the project.

6.2 EXAMPLE

In the example Table 5, a device that is battery operated has a requirement for low operating current. The dynamic current measurement requires the use of some portion of functional test to cycle upon during the IDD measurement.

In the case where a loop has been defined, the vector number has no meaning. For the static measurement the vector number applies to the cycle number where the device is in a fully static condition. This includes all internal busses driven to a known level and not in a high impedance state.

In the case of standard tests for particular cells, calling out the cell name and giving a pin state table that puts the device in a state that will not interfere with the cell is the most straight forward way to go. Cells like pull ups, pull downs, osc cells, I/O leakage, and microprocessor cells have standard tests available now. The Newsletter and ZyMOS applications can be helpful as to currently available standard tests.

After filling out the table, it should be submitted for development quote as soon as possible.

6.3 PACKAGING

Table 6A below shows the standard package types available versus Silicon Gate CMOS Library Grid Counts. For Silicon Gate CHMOS designs, multiply the grid count number shown in Table 6A by 14.

Table 6B is provided for selection of prototype and production package types.

NO. of lds	DIE SIZE		DUAL-IN-LINE		CHIP CARRIERS	
			PLASTIC	CERAMIC	PLASTIC (leaded)	CERAMIC (leadless)
8	GRIDS	MIN	200	600	N/A	N/A
		MAX	1200	2400	N/A	N/A
14	GRIDS	MIN	100	1200	N/A	N/A
		MAX	1800	3400	N/A	N/A
16	GRIDS	MIN	100	1200	N/A	N/A
		MAX	2400	3400	N/A	N/A
18	GRIDS	MIN	600	1000	N/A	100
		MAX	3000	3400	N/A	900
20	GRIDS	MIN	700	1000	N/A	N/A
		MAX	2400	3400	N/A	N/A
22	GRIDS	MIN	1500	2400	N/A	N/A
		MAX	3000	5000	N/A	N/A
24	GRIDS	MIN	800	3200	N/A	N/A
		MAX	4200	6000	N/A	N/A

TABLE 6A. PACKAGE TYPES AND GRID COUNTS

NO. of lds	DIE SIZE		DUAL-IN-LINE		CHIP CARRIERS	
			PLASTIC	CERAMIC	PLASTIC (leaded)	CERAMIC (leadless)
28	GRIDS	MIN	1100	3200	2400	200
		MAX	4200	6000	4000	4000
36	GRIDS	MIN	N/A	N/A	N/A	1000
		MAX	N/A	N/A	N/A	3500
40	GRIDS	MIN	1000	3200	N/A	2800
		MAX	6500	10000	N/A	5500
44	GRIDS	MIN	N/A	N/A	N/A	4200
		MAX	N/A	N/A	N/A	7000
48	GRIDS	MIN	N/A	N/A	N/A	1100
		MAX	N/A	N/A	N/A	7500
64	GRIDS	MIN	5200	6500	N/A	N/A
		MAX	9000	16000	N/A	N/A
68	GRIDS	MIN	N/A	N/A	2800	6500
		MAX	N/A	N/A	5500	18000
84	GRIDS	MIN	N/A	N/A	N/A	N/A
		MAX	N/A	N/A	N/A	N/A

TABLE 6A. PACKAGE TYPES AND GRID COUNTS (continued)

LEAD COUNT	DUAL-IN-LINE		CHIP CARRIERS	
	PLASTIC	CERAMIC	PLASTIC	CERAMIC
8			N/A	N/A
14			N/A	N/A
16			N/A	N/A
18			N/A	
20			N/A	N/A
22			N/A	N/A
24			N/A	N/A
28				
36	N/A	N/A	N/A	
40			N/A	
44	N/A	N/A	N/A	
48	N/A		N/A	
64			N/A	N/A
68	N/A	N/A		
84	N/A	N/A	N/A	N/A

TABLE 6B. PACKAGE TYPES AVAILABLE

APPENDIX A

A.1 SENTRY TESTER SPECIFICATIONS

The tester specifications should be reviewed before major design work is started. At the very least it should be done before undertaking the completion of the ZyP Device Specification. As outlined in this document, the tester attributes affect the testability of all devices.

Range	Voltage Range	Max Load	Least Significant Digit	Accuracy (% of value)
2	LOW 0 to ±10.23 V*	±1 A	10 mV	0.1% ±1 bit
3	HIGH 0 to ±40.92 V*	±1 A	40 mV	0.1% ±1 bit
Regulation	No-load to full-load: ±20 mV**			
Time Response	Settling to 1% in 1 ms Overshoot 1% of step value change into resistive load			

FIGURE A-1 DEVICE POWER SUPPLY SPECIFICATIONS

Range	Forcing Range	Resolution	Accuracy (% of programmed value)
2	0 to +6 V or -10.23 V	10 mV	0.1% ±10 mV
3	0 to +6 V or -16.00 V	40 mV	0.1% +40 mV
System Repeatability/ Noise	±10 mV max (allowed in internal node check)		

FIGURE A-2 REFERENCE POWER SUPPLY SPECIFICATIONS

Force Voltage/Measure Current

Range	Voltage Forcing Range	Resolution	Rise Time* (10-90%) (all measuring ranges)	Accuracy (% of programmed value)	Overshoot* (% of programmed value) (all measuring ranges)
1	0 to ±1.023 V	1 mV	100 μs	±.3% ±4 mV	<15%
2	0 to ±10.23 V	10 mV	200 μs	±.15% ±10 mV	<5%
3	0 to ±40.92 V	40 mV	400 μs	±.15% ±40 mV	<1%
4	0 to -102.30 V	100 mV	600 μs	±.15% ±100 mV	<1%

Required Measurement Delay** (in ms)
Forcing Ranges 1/2/3/4

Range	Current Measuring Range	Least Significant Bit	Accuracy*** (% of measured value)	Typical Value	5 MHz Max Value	10 MHz Max Value
0	0 to ±1.023 μA	1 nA	±.5% ±10 nA	4/5/20/40	10/15/50/100	10/15/50/120
1	0 to ±0.1023 mA	0.1 μA	±.15% ±200 nA	0/0/0.5/1	0/0/1/2	1/1/1.5/3
2	0 to ±10.23 mA	10 μA	±.15% ±20 μA	0/0/0.5/1.5	0/0/1.5/3	0/0/1.5/3
3	0 to ±102.3 mA	100 μA	±.2% ±200 μA	0/0/0.5/1.5	0/0/1.5/3	0/0/1.5/3

Voltage compliance for all ranges: -93.0 V

Force Current/Measure Voltage

Range	Current Forcing Range	Least Significant Bit	Rise Time* (10-90%) (all measuring ranges) 1(1 V)	2(10 V)	3(40 V)	4(100 V)	Accuracy*** (% of programmed value)	Overshoot* (% of measuring ranges) 1/2/3/4
0	0 to ±1.023 μA	1 nA	700 μs	5 ms	20 ms	45 ms	±.5% ±100 nA	20/5/3/3
1	0 to ±0.1023 mA	0.1 μA	110 μs	250 μs	500 μs	800 μs	±.15% ±200 nA	20/5/3/3
2	0 to ±10.23 mA	10 μA	110 μs	250 μs	500 μs	800 μs	±.15% ±20 μA	20/5/3/3
3	0 to ±102.3 mA	100 μA	110 μs	250 μs	500 μs	800 μs	±.15% ±200 μA	20/5/3/3

Voltage compliance for all ranges: -93.0 V

Required Measurement Delay** (in ms)
Forcing Ranges 0/1/2/3

Range	Voltage Measuring Range	Least Significant Bit	Accuracy*** (% of measured value)	Typical Value	5 MHz Max Value	10 MHz Max Value
1	0 to ±1.023 V	1 mV	±.3% ±4 mV	3/0/0/0	8/0.5/0.5/0.5	8/0.5/0.5/0.5
2	0 to ±10.23 V	10 mV	±.15% ±20 mV	20/0/0/0	35/1/1/1	48/1/1/1
3	0 to ±40.92 V	40 mV	±.15% ±40 mV	80/2/1/0	130/6/2/2	190/6/2/2
4	0 to -102.30 V	100 mV	±.15% ±100 mV	190/5/2/2	270/13/5/5	480/13/5/5

FIGURE A-3 PRECISION MEASUREMENT UNIT SPECIFICATIONS

Test Rate Period Pulse	Range	Full Scale	Programming Resolution
	0	40 μs	10 ns
	1	400 μs	100 ns
	2	4 ms	1 μs
	3	40 ms	10 μs

Pulse Width or Delay	Range	Full Scale	Programming Resolution
	0	10 μs	0.16 ns
	1	100 μs	100 ns
	2	1 ms	1 μs
	3	10 ms	10 μs

Conditions for above:
Pulse width ≤ period
Pulse delay ≤ period
Pulse delay + width ≤ 2periods

Accuracy of Programmed Value Maximum: 0.5 ns

Note

The above times exclude the use of I/O switching.

FIGURE A-4 TIMING GENERATOR SPECIFICATIONS

ZyTEST USER'S GUIDE

Doc: 20-010-210 Rev: C | Issued: September 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

Table of Contents

8

1 INTRODUCTION

1.1 SCOPE

This document describes a collection of application software modules available under the general name of ZyTEST. As the name implies, ZyTEST is the ZyP systems link to test program generation from the design verification features of ZyPSIM.

The ZyTEST user's guide is intended to describe the operation of the various programs and how to interpret the results. The technical background of each feature can be found in the ZyP Specification and Test User's Guide and we recommend that both documents be used together.

1.2 OVERVIEW

ZyTEST has two primary functions to support test program generation:

a. Verification of the functional test pattern to guarantee device performance under extremes of process and environmental conditions.

b. Conversion of the ZyPSIM logic simulation output into a SENTRY compatible functional test pattern.

The section on general operation describes those features which are common to all programs. The features and operating details of each program are individually described in the following sections.

1.3 RELATED DOCUMENTS

The following documents should be used in conjuction with this user's guide to fully understand the concepts which relate to logic simulation and test program generation.

20-010-009	ZyPSIM User's Guide.
20-010-309	ZyPSIM Test Language
20-010-010	ZyP Specification and Test User's Guide.
20-010-310	ZySPEC User's Guide

1.4 DOCUMENT CONVENTIONS

In the following instructions and sample terminal sessions showing program operation, user input is shown underlined and the computer's response is not underlined. User input will always end with a Carriage Return <CR> but this will not be shown. For example:

TYPEME

In this example the user responded with "TYPEME<CR>" which is indicated by TYPEME.

2 GENERAL OPERATION

2.1 OPERATION

ZyTEST may be invoked by typing ZYTEST. A menu of available options will be displayed, each option being assigned a number. At this point the number of the desired option should be entered which will run the appropriate program. Required data will be prompted for.

2.2 FILENAMES

Throughout this document filenames are referred to which may have two components: prefix and extension. When a filename has two components, the "." (period) is used as a separator. The component preceeding the "." is known as the prefix and following the "." is the extension. The prefix will be the common name and the extension will differentiate the file types for any given option.

Prefixes are referred to as "xxxx" which represents any number of valid filename characters (but no periods). Extensions will be described in each section as they apply to the selected option.

It is extremely important that the filename prefix be the same for the different files that ZyTEST will use. For example, when working on a circuit named CHIP1, the COMPARE option will require two of the following files:

CHIP1.TPS

CHIP1.TPN

CHIP1.TPF

3 COMPARE OPTION

3.1 PURPOSE

The compare option is used to verify that the performance of the logic network is independent of process parameter variations and environmental conditions specified and simulated by the user. For detailed information refer to the simulation guardbanding section of the ZyP Specification and Test User's Guide and ZyPSIM User's Guide.

3.2 METHOD

The ZyPSIM simulation output generated using a .GUARDBAND command with CONDITION=SLOW is compared to the output generated with CONDITION=FAST. Thus the following two situations are compared:

a. Low VDD, high temperature and slow processing parameters (CONDITION=SLOW).

b. High VDD, low temperature and fast processing parameters (CONDITION=FAST).

The comparison takes place on a pin for pin, cycle by cycle basis considering mask and direction information. The logic level in each bit (pin) position and each vector (cycle) in both files must be the same. The title line of the .NET file (which becomes the header lines of the output files) may be different for the fast and slow simulations because the header lines are ignored during the comparison.

Detected differences should be examined and may be identified as being caused by a logic network problem or they may be legitimate cycle slips. When discrepancies are known, understood and acceptable, they can be dealt with by masking the output pin in question during the strobe times that discrepancies occur. This will disable testing for that pin during periods of output uncertainty. The COMPARE feature can thus be used as an aid in generating the masking information that may be needed.

3.3 FILE REQUIREMENTS

To run a ZyTEST comparison (in preparation for generating a test pattern) two files are required:

3.3.1 .TPS

xxxx.TPS is the file generated as a result of the .TPRINT Test Language command when the .GUARDBAND command has CONDITION=SLOW. This is the "worst case slow" file.

3.3.2 .TPF

xxxx.TPF is the file generated as a result of the .TPRINT Test Language command when the .GUARDBAND command has CONDITION=FAST. This is the "worst case fast" file.

The xxxx.TPN file (generated when CONDITION=NOM or if there is no .GUARDBAND command) is optional. It may be compared with either the .TPS or .TPF files if desired. This file is required, though, for the SENTRY option discussed later.

3.4 RUNNING THE COMPARISON PROGRAM

3.4.1 GENERAL PROCEDURE

To run the ZyTEST comparison program type ZYTEST. The first question you will be asked is whether you want to 1) compare output files, or 2) generate a SENTRY test program (or you can quit). Choose the comparison option by typing 1. Continue to respond to the prompts. The full structure of the menu is shown in the ZYTEST MENU TREE near the end of this document.

Differences are indicated individually on a cycle and pin basis. The lines in each file which are different are displayed with their respective cycle numbers. Also, the pin(s) containing the differences are pointed out by an exclamation mark.

An output file is also created which is a listing of all detected differences. The filename will be xxxx.DIF (or COMPARE.DIF if you are working with arbitrary filenames). For the files mentioned earlier the output filename would be CHIP1.DIF.

NOTE: The above discussion assumes that you are attached to the directory in which CHIP1.TPS, et cetera, reside. This is usually the most convenient situation, but you do not actually need to be there in order to compare the files. Just give the full pathname, except for the extension, when asked for the prefix. ZyTEST will compare the files and put the results in your current working directory.

3.4.2 SAMPLE NETLIST

In the sample terminal sessions that will come later we will use the output files from the following netlist.

```
* ZYTEST DEMONSTRATION FILE       DEMO.NET
.GUARDBAND  VDD=4.5,5.5  TEMP=0,70  CONDITION=SLOW
*GUARDBAND  VDD=4.5,5.5  TEMP=0,70  CONDITION=FAST
*GUARDBAND  VDD=4.5,5.5  TEMP=0,70  CONDITION=NOM
.PADS   PAD1:1 PAD2:2 VDD:3 VSS:4
PAD1  PAA1  PAD1  N1     % INVERTING  INPUT PAD
PAD2  PBA1  N2    PAD2   % INVERTING OUTPUT PAD
VDD   PDC1  VDD   PWR    % VDD PAD
VSS   PDD1  VSS   GND    % VSS PAD
N2    IAA4  N1    N2     % INV BETWEEN PADS
C2    CAP   PAD2:50      % 50 pF TESTER LOAD ON OUTPUT PAD
.TPERIOD  200            % 5 MHz CLOCK RATE
.TGEN NRZ   1  PAD1      % PAD1 SIGNAL IS NON-RETURN-TO-ZERO
.TTABLE        PAD1 MSK
1              0    1
1              0    1    % NEED PAD2=1 AT THIS STROBE
1              1    1
1              1    1    % NEED PAD2=0 AT THIS STROBE
1              0    1
1              0    1    % NEED PAD2=1 AT THIS STROBE
.TSTROBE  180  PAD2,MSK  % STROBE PAD2 AT 180 nS IF MSK=1
.TPRINT PAD1   PAD2   VDD   VSS
.MAXTIME=1200            % SIMULATE FOR SIX TO PERIODS
.GO
.END
```

If you would like to duplicate the sample session results simply type in this netlist and file it as DEMO.NET. Then run ZyPSIM by typing ZYPSIM DEMO ZY4

By changing the appropriate *'s of the GUARDBAND lines to .'s you can generate the fast and slow files that you'll need.

You will get files that contain the following test cycle information:

```
     fast              slow
  DEMO.TPF          DEMO.TPS

     PPVV              PPVV
     AADS              AADS
     DDDS              DDDS
     12                12

CYCLE             CYCLE

00000 dmmm        00000 dmmm
00001 d1mm        00001 d1mm
00002 d1mm        00002 d1mm
00003 D0mm        00003 D1mm
00004 D0mm        00004 D0mm
00005 d1mm        00005 d0mm
00006 d1mm        00006 d1mm
```

3.4.3 SAMPLE TERMINAL SESSION

Shown below is a sample terminal session using the files generated from the netlist shown above.

```
ZYTEST

                        ZyTEST
                Test Program Generation Utility
                Version 5.1.4     Release Date: 09/19/84
            A proprietary product of ZyMOS Corporation
        477 North Mathilda, Sunnyvale, California, 94086, USA


    Please select one of the following options:

    1)  Compare ZyPSIM output files (COMPARE).

    2)  Generate a Sentry compatible test program (SENTRY).

    Enter Option (1 or 2) (<CR>=QUIT)>: 1


                        COMPARE
                File Comparison Program
                Version 5.1.7     Release Date: 09/18/84
            A proprietary product of ZyMOS Corporation
        477 North Mathilda, Sunnyvale, California, 94086, USA

  Do you want to compare :

     1) Normal test language files, e.g.  chip.TPS, chip.TPF, chip.TPN
     2) Arbitrary files

  Enter selection (1 or 2).  (<CR>:Quit) > 1

  Enter file prefix (everything up to the period).  (<CR>:Quit)> DEMO

  Please select one of the following 3 comparison choices :

      1.  Fast    verses Slow
      2.  Nominal verses Slow
      3.  Nominal verses Fast

  Enter selection (1, 2, or 3).  (<CR>:Quit)> 1
```

```
Comparing DEMO.TPF and DEMO.TPS ...

   COMPARE Version 5.1.7  (09/19/84)        13:49:19      09/20/84

                              Comparison of

                          DEMO.TPF   vs   DEMO.TPS

            PPVV
            AADS
            DDDS
            12

        CYCLE

        00003 D0mm
        00003 D1mm
               !

        00005 d1mm
        00005 d0mm
               !

The total number of legal differences is       2
```

8

```
                  ----- end of sample terminal session -----
```

ZyTEST has found two discrepancies between the slow file (DEMO.TPS) and the fast file (DEMO.TPF). They occured at cycles 3 and 5 and this information will be contained in the file DEMO.DIF as follows:

```
   COMPARE Version 5.1.7 (09/19/84)        13:49:19      09/20/84

                              Comparison of

                          DEMO.TPF   vs   DEMO.TPS

            PPVV
            AADS
            DDDS
            12

        CYCLE

        00003 D0mm
        00003 D1mm
               !

        00005 d1mm
        00005 d0mm
               !

The total number of legal differences is       2
```

Thus for this circuit there were cycle slips at cycles 3 and 5. Assuming that this is acceptable we can mask the PAD2 output during these cycles by changing the .TTABLE in DEMO.NET, i.e.:

```
.TTABLE        PAD1 MSK
 1              0    1
 1              0    1    % NEED PAD2=1 AT THIS STROBE
 1              1    0    % DON'T CARE ABOUT PAD2 HERE
 1              1    1    % NEED PAD2=0 AT THIS STROBE
 1              0    0    % DON'T CARE ABOUT PAD2 HERE
 1              0    1    % NEED PAD2=1 AT THIS STROBE
```

With this new .TTABLE the resulting test cycle information will be:

```
  fast              slow
DEMO.TPF          DEMO.TPS

    PPVV              PPVV
    AADS              AADS
    DDDS              DDDS
    12                12

CYCLE             CYCLE

00000 dmmm        00000 dmmm
00001 d1mm        00001 d1mm
00002 d1mm        00002 d1mm
00003 Dmmm        00003 DMmm
00004 DOmm        00004 DOmm
00005 dMmm        00005 dmmm
00006 d1mm        00006 d1mm
```

Now running the ZyTEST comparison program to compare the slow and fast simulations will produce the message:

The files are the same

Of course the files are not identical but when considering only the inputs and unmasked outputs they are the same, and that's what is important in order to generate a test program.

4 SENTRY OPTION

4.1 PURPOSE

The SENTRY option is used to convert the final verified ZyPSIM output into a SENTRY compatible functional test program. In addition to the task of data conversion, pattern compression is also performed. Compression techniques are applied to reduce the data so that it can reside in the SENTRY's high speed local memory (which has a capacity of 4K vectors). Therefore the SENTRY option is run by users to see if the test pattern they have designed can be compressed to less than 4K vectors, and thus avoid the need for multiple memory loads with their added complexity.

4.2 METHOD

ZyTEST uses both HORIZONTAL and VERTICAL compression techniques to reduce the SENTRY memory requirements. More information on how this is done can be found in the later section SOME DETAILS OF OPERATION.

4.3 FILE REQUIREMENTS

The ZyTEST SENTRY option requires only one file -- the xxxx.TPN file.

4.4 RUNNING THE SENTRY PROGRAM

4.4.1 GENERAL PROCEDURE

To run the ZyTEST SENTRY program type ZYTEST to get the initial menu that we saw before when discussing the comparison program. Now type 2 for the SENTRY option. You will be prompted for additional information. The full menu structure is shown in the ZYTEST MENU TREE near the end of this document. The SENTRY option will also accept full pathname inputs as described for the COMPARE option.

4.4.2 SAMPLE TERMINAL SESSION

Below is a sample terminal session using the final DEMO.TPN file produced from DEMO.NET.

```
ZYTEST

                        ZyTEST
              Test Program Generation Utility
          Version 5.1.4     Release Date: 09/19/84
        A proprietary product of ZyMOS Corporation
    477 North Mathilda, Sunnyvale, California, 94086, USA

Please select one of the following options:

1)  Compare ZyPSIM output files (COMPARE).

2)  Generate a Sentry compatible test program (SENTRY).

Enter Option (1 or 2) (<CR>=QUIT)>: 2
```

```
                              SENTRY
                    SENTRY TESTER CONVERSION PROGRAM
                  Version 5.1.5     Release Date: 09/15/84
               A proprietary product of ZyMOS Corporation
             477 North Mathilda, Sunnyvale, California, 94086, USA

Enter file prefix (everything up to the period). (<CR>:Quit)> DEMO

Do you want to try straight line compression ? Y

Beginning readin
Finished readin

Total input  vectors  =     7
Total output vectors  =     9

*****  You made it in one memory load.  *****

Do you want to try subroutine compression ? Y

Beginning Subs
Finished Subs
Finished Shlsrt
Finished Weedo
Finished Garbij

Total vectors after subroutining =     7

First pass created    0 subroutines.
Finished Pass2
Finished Pass3

*** MESSAGE :    10 vectors written to output file.

Finished Vert

For test pattern use file DEMO.FTS

*****  All Done.  *****

          ----- end of sample terminal session -----
```

Note that in this case the number of output vectors was greater than the number of input vectors for both straight line and subroutine compression. This is because the absolute number of input vectors was small so that overhead statements were a significant factor. The SENTRY test pattern using straight line compression is in the xxxx.FTP file and the pattern using subroutine compression is in the xxxx.FTS file.

Note that subroutine compression is a time consuming and computation intensive task for the computer. The use of subroutine compression is optional and is normally used only when the total vector count from straignt line compression exceeds 4096.

4.5 SOME DETAILS OF OPERATION

Two techniques are employed in reducing the pattern data: HORIZONTAL and VERTICAL compression. Both make use of features offered by the SENTRY tester for this purpose and are described separately.

4.5.1 VERTICAL COMPRESSION

Vertical compression attempts to reduce the total number of vectors required to define the functional test pattern. Vertical compression can potentially enable very long test patterns to be reduced to less than 4K vectors. Two methods are used in order to achieve vertical compression: 'straight line' and subroutine compression. Both take advantage of the SENTRY register modification statements described in the ZyP Specification and Test User's Guide.

4.5.1.1 STRAIGHT LINE COMPRESSION

When the output from the simulation contains consecutive occurences of the same vector, the CLOCK BURST or repeat count statement can be utilized. First, consider the case when consecutive patterns do not repeat:

```
10   0 1 0 1
11   1 0 1 1
12   0 1 0 1
```

would convert to the SENTRY statements:

```
SET F 0101;
SET F 1011;
SET F 0101;
```

SET F is the statement which defines a test vector in the SENTRY.

Note that in this case no compression occurs, a separate statement and hence memory location is required for each vector. This is due to the fact that although the vectors (patterns) for cycles #10 and #12 are identical, they are not consecutive. Now consider the following pattern:

```
10   0 1 0 1
11   0 1 0 1
12   0 1 0 1
```

this would convert to:

```
SET FC 3 0101;
```

In this case, three consecutive repeats occur in the ZyPSIM pattern and the SENTRY SET FC (functional count) statement with a count of 3 can be used. This requires only one high speed memory location.

The mask and direction information contained in the ZyPSIM output file (the .TPN file) is taken into account by ZyTEST so that patterns that are not strictly identical may still be "the same" as far as the SENTRY is concerned. For example the following vectors would be seen as "the same" and would be compressed.

```
10    0 m 0 1
11    0 M 0 1
12    0 w 0 1
```

4.5.1.2 SUBROUTINES

During subroutine compression ZyTEST looks for repeating blocks of patterns that can be defined as subroutines in the SENTRY local memory. These subroutines may be called with repeat counts which may further reduce the memory space needed. Subroutine compression usually takes more computer time than straight line compression so it is not normally used unless straight line compression cannot reduce the vector count enough.

4.5.2 HORIZONTAL COMPRESSION

Horizontal compression is used to reduce the size of the ASCII string which defines the test vector. When the test program is compiled, reduced string size results in faster compile times. Several SENTRY features are utilized to reduce string size as follows:

4.5.2.1 VECTOR DATA CHANGES

When defining a vector, only the data which is different from the previous vector need be defined. For example:

```
0 0 1 1
1 0 1 1
```

would be written as:

```
SET F 0011;
SET F 1;
```

4.5.2.2 REPEAT COUNT

Vectors containing repeating "1's" and/or "0's" will be reduced to a single "1" or "0" with a repeat count if this reduces the overall string size. For example:

```
0 0 0 0 0 0 1 1
```

would be written as:

```
SET F (6:0)11;
```

However, the pattern:

```
0 0 0 0 1 1
```

would be written as:

```
SET F 000011;
```

In this case using (4:0) requires 5 characters whereas defining 0000 requires only 4.

4.5.2.3 COLUMN ADDRESSING

When data changes occur in the right hand columns, column addressing is used to avoid writing unchanged data to reach the changed column. For example:

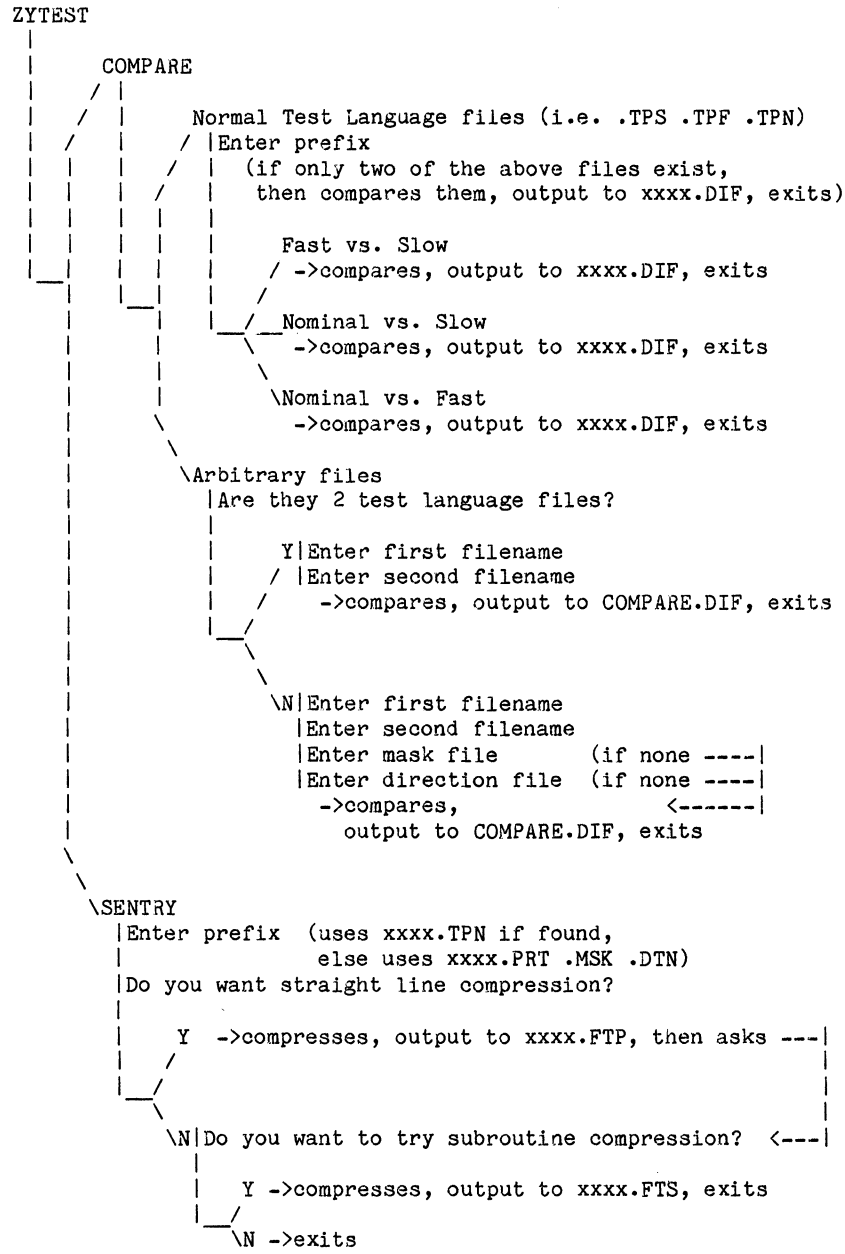
```
0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 0 1 0 0
```

would be written as:

```
SET F 000011110000;
SET F [10]1;
```

Using [10] defines that the next column referenced is column 10.

5 ZYTEST MENU TREE



8

8

ZySPEC USER'S GUIDE

Doc: 20-010-310 Rev: B | Issued: March 1985

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: B
Table of Contents	Rev: B
Page 1-1	Rev: B
Pages 2-1 through 2-3	Rev: B
Pages 3-1 through 3-7	Rev: B

TABLE OF CONTENTS

9

1 INTRODUCTION

1.1 SCOPE

This user's guide describes operation and use of the ZyP application program ZySPEC.

1.2 OVERVIEW

ZySPEC is an automatic specification generation program integrated into the ZyP Design System. ZySPEC receives device information from the ZyPSIM network file and terminal input. This data is in turn processed with library specific data such as input levels, output levels, and maximum ratings to produce a specific ZyP device specification.

ZySPEC has two modes of operation, user data input and specification runoff. Users are only required to run ZySPEC in the user data input mode. This data is picked up, in turn, and internally processed by the manufacturer into a final device specification document. The final device specification is returned to the user for approval.

1.3 RELATED DOCUMENTS

ZySPEC is an integral part of the ZyP Design System. To aid in understanding the relationship between ZySPEC and the ZyP system, as well as to aid in successful execution of ZySPEC, the following documents are recommended reading.

20-010-009	ZyPSIM User's Guide
20-010-209	ZyPSIM Reference Guide
20-010-309	ZyPSIM Test Language
20-010-010	ZyP Specification and Test User's Guide
20-010-016	ZyPEE User's Guide

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, the following conventions are used.

a. All user inputs to the computer are shown bold and underlined.

b. <CR> indicates the single typed key RETURN.

2 SUMMARY OF OPERATION

2.1 PREPARING TO RUN ZySPEC

ZySPEC has been designed to run with minimal user interaction. However, a certain amount of preparation is required before ZySPEC will execute successfully.

First, the user must have a ZyPSIM network file which compiles without error. This file must include the .GUARDBAND statement and test pattern definition using the ZyPSIM test language.

Second, the user must have completed ZyPEE verification of the .NET file. ZyPEE generates a .TST file used by ZySPEC. ZySPEC also uses the ZyPEE generated .GOT file for reference.

Finally, ZySPEC needs the .TPN test pattern file. This file is generated by running ZyPSIM with the parameter CONDITION=NOM selected in the .GUARDBAND statement.

2.2 OPERATION

ZySPEC is invoked by typing ZySPEC. ZySPEC is menu driven and all required input is prompted from the terminal.

9

2.2.1 MODE SELECTION

ZySPEC operates in two modes;

1. User data input or modification
2. Specification runoff

Mode 1 prompts for all necessary input data and Mode 2 generates the final specification source file. Normally, Mode 2 is run only by ZyMOS and is not run by the user.

2.2.2 MODE 1

Mode 1 is the data entry mode. When selected, ZySPEC prompts for network file and library. Proper responses are the ZyPSIM .NET file and appropriate library selection as indicated in the library prompt.

When run initially, ZySPEC enters the data entry mode. Otherwise, with a rerun of ZySPEC from an existing .SPC file, ZySPEC will go to the main menu.

Data entry is divided into two sections, customer information and packaging information. Customer information is requested first. Provide names, addresses and phone numbers as indicated. Packaging information involves many options. All available options are indicated in menu form. Provide package type, pin count and marking information as requested.

At the completion of data entry, ZySPEC displays the main menu as follows:

1. Edit Customer Information
2. Edit Packaging Information
3. Create a file for transfer to ZyMOS
4. Exit (save changes)
5. Quit

Editing proceeds in a straight forward fashion. However, to complete execution of ZySPEC, a file for transfer must be created. To do so, select menu option 3. When selected, ZySPEC will prompt for names of .TST, .NET, .TPN and .GOT files. Upon entering, ZySPEC will create a .SPC file and terminate execution.

Restarting ZySPEC in Mode 1 will cause the main menu to be displayed.

2.2.3 MODE 2

Mode 2 is the specification source file generation mode. The source file is generated by processing the .SPC file generated in Mode 1. Normally, Mode 2 is only run by ZyMOS.

2.3 FILES

2.3.1 INPUT FILES

To run properly, ZySPEC requires the following input files:

* chip.NET - Network file including .GUARDBAND statement and test language.
* chip.TPN - ZyPSIM generated pattern file using .GUARDBAND statement with CONDITION=NOM.
* chip.GOT - ZyPEE error output file.
* chip.TST - ZyPEE generated file (condensed .NET file).

3 OPERATION

ZySPEC operates in two different modes. One of these modes creates the specification and can only be run by a licensee, a design center, or by ZyMOS internally. This mode enables ZyMOS to create the device specification.

The other mode is for use by all users. In the user mode ZySPEC is used to gather the necessary customer data for a smooth continuation of the project when ZyMOS begins to prepare the design for silicon.

3.1 CUSTOMER INFORMATION DATA ENTRY

ZySPEC is invoked by typing ZYSPEC. There are no arguments used. When invoked, ZySPEC will ask the user to select the mode of operation. Customer users should always select option 1 as shown here.

NOTE: ZySPEC keeps all the information it gathers in a file called chip.SPC, where chip is the user defined file name. ZySPEC will prompt the user to enter the name of the chip file next. If this file exists then ZySPEC will read this file and use all the data from the file that has been input before and go to the main menu as shown in paragraph 3.3. An example of a new file created under the name DEMO_TEST.SPC is used here to show user inputs. The DEMO_TEST file is the DEMO_TEST.NET file used in most of the ZyP Documentation.


```
ZYSPEC<CR>

                        ZySPEC
                ZyMOS device specification program
              Version 5.2.1    Release Date: 02/15/85
          A proprietary product of ZyMOS Corporation
      477 North Mathilda, Sunnyvale, California, 94086, USA


   1    User data input or modification
   2    Specification source file generation

Please select 1 or 2. (<CR>:Quit)> 1<CR>

Enter file prefix (everything up to the  period). (<CR>:Quit)> DEMO_TEST<CR>
```

```
Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY5<CR>

  Please answer the following questions

Customer part designation > MY.PART<CR>
ZyMOS part number         > 51256<CR>
Company Name              > MY.CO<CR>
Street Address            > MY.STREED<CR>
City                      > MY.CITY<CR>
State                     > CA<CR>
Zip Code                  > 99999<CR>
Technical Contact         > ME<CR>
Phone                     > (408) 555-7777<CR>
Purchasing Contact        > ANOTHER PERSON<CR>
Phone                     > (408) 555-7778<CR>

  ZyMOS Part Number: 51256
            Device: MY.PART
          Customer: MY.CO
                    MY.STREED
                    MY.CITY
                    CA   99999
          Contacts: ME
                    (408) 555-7777
                    ANOTHER PERSON
                    (408) 555-7778

Enter E to edit or <CR> to proceed > E<CR>)
```

NOTE: We have answered E to the above example to review all our entries in the following example. Street is misspelled so we will make this change. A <CR> after any entry leaves that entry unchanged.

```
Please answer the following questions

Customer part designation > MY.PART <CR>
ZyMOS part number         > 51256 <CR>
Company Name              > MY.CO <CR>
Street Address            > MY.STREED MY.STREET<CR>
City                      > MY.CITY <CR>
State                     > CA <CR>
Zip Code                  > 99999 <CR>
Technical Contact         > ME <CR>
Phone                     > (408) 555-7777 <CR>
Purchasing Contact        > ANOTHER PERSON <CR>
Phone                     > (408) 555-7778 <CR>

  ZyMOS Part Number: 51256
            Device: MY.PART
          Customer: MY.CO
                    MY.STREET
                    MY.CITY
                    CA   99999
          Contacts: ME
                    (408) 555-7777
                    ANOTHER PERSON
                    (408) 555-7778

Enter E to edit or <CR> to proceed > <CR>
```

NOTE: After completing the above customer information entering <CR> will cause ZySPEC to prompt for packaging information as shown in paragraph 3.2. The following example shows completing the packaging section. Paragraph 3.4 shows how to run ZySPEC and edit existing information or add information left out during the initial run.

3.2 PACKAGE INFORMATION DATA ENTRY

The next step is to enter the packaging and assembly information. This is done again by a question and answer session.

NOTE: After selecting the package configuration and material, ZySPEC displays the number of pins available for the configuration of package, material, and library you are using. The example shown is a typical example and may not show the current available packages and assembly flow.

And now we have some questions on Packaging

```
1    CERAMIC DIP
2    PLASTIC DIP
3    CERAMIC LCC
4    PLASTIC LCC
5    CERDIP

Please select option (1-05).  > 1<CR>
```

```
 1    CERAMIC DIP    8 PIN
 2                  14 PIN
 3                  16 PIN
 4                  18 PIN
 5                  20 PIN
 6                  22 PIN
 7                  24 PIN
 8                  28 PIN
 9                  40 PIN
10                  64 PIN

Please select option (1-10).  > 3<CR>

 1    50-900-01 MEDICALLY IMPLANTABLE DEVICES
 2    50-900-02 ZyREL B HERMETIC
 3    50-900-03 ZyREL C HERMETIC
 4    50-900-04 COMMERCIAL HERMETIC

Please select option (1-04).  > 2<CR>
```

NOTE: If any discrepancy exist to package size or pin capacity ZySPEC will give the following warning message.

> Warning: There exists a potential problem in fitting your device in the selected package. Please contact ZyMOS for additional information.

```
Marking (11 characters max)  > MY.PART<CR>

   Packaging information:

                Style: DIP
                 Type: Ceramic
                 Pins: 16

              Marking: MY.PART
        Assembly flow: 50-900-02 ZyREL B HERMETIC


Enter E to edit or <CR> to proceed >  <CR>
```

NOTE: Since the complete packaging information is displayed we can check it for accuracy and then go on to the next step. A <CR> indicates that no editing will be done to the customer and packaging information. ZySPEC will automatically go to the Main Menu.

3.3 THE MAIN MENU

Once the user has entered all the customer and packaging data that ZyMOS requires, ZySPEC enters the main menu. The main menu looks like this.

```
1) Edit Customer Information
2) Edit Packaging Information
3) Create a file for transfer to ZyMOS
4) Exit (save changes )
5) Quit
```

NOTE: At this point you have the options of editing customer information, editing packaging information, creating a file for transfer to ZyMOS, exiting and saving the file, or quit. Since we have entered all customer and packaging information and all files are complete, we can create the file for transfer to ZyMOS. To create the file for transfer we will select 3 from the menu. Paragraph 3.4 shows how to reenter a previously created file and make changes.

```
Please select option (1-5). > 3<CR>

    The following files are needed to create a .SPC file for transfer to ZyMOS.
   DEMO_TEST.TST
   DEMO_TEST.NET
   DEMO_TEST.TPN
   DEMO_TEST.GOT

Type C to continue or <CR> to return to the menu > E<CR>


[CONCAT Rev 19.0]

OK,   INS  **A1 DEMO_TEST.TST
OK,   INS  **A2 DEMO_TEST.NET
OK,   INS  **A3 DEMO_TEST.TPN
OK,   INS  **A4 DEMO_TEST.GOT
OK,   QUIT
OK,
```

NOTE: After ZySPEC creates the file for transfer to ZyMOS it takes you back to the operating system. As indicated in the above display you will see an OK, QUIT and then another OK,. This indicates the transfer file is complete and you are back to the operating system. If the .TST, .NET, .TPN or the .GOT files are not found, ZySPEC will output the following message and default back to the operating system.

```
 These file(s) were not found and the transfer file .SPC file was not created.
DEMO_TEST.TST
DEMO_TEST.NET
DEMO_TEST.TPN
DEMO_TEST.GOT
```

3.4 EDITING INFORMATION ALREADY ENTERED

To edit information already saved in a .SPC file run ZySPEC and give the name of the file that you wish to edit when prompted for the filename. ZySPEC will then read this information from the file and present the main menu. After the edit session the user should use main menu item 4 to save the changes.

ZYSPEC

ZySPEC
ZyMOS device specification program
Version 5.2.1 Release Date: 02/15/85
A proprietary product of ZyMOS Corporation
477 North Mathilda, Sunnyvale, California, 94086, USA

1 User data input or modification
2 Specification source file generation

Please select 1 or 2. (<CR>:Quit>1<CR>

Enter file prefix (everything up to the period). (<CR>:Quit) >DEMO_TEST<CR>

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY5<CR>

What would you like to do??

1) Edit Customer Information
2) Edit Packaging Information
3) Create a file for transfer to ZyMOS
4) Exit (save changes)
5) Quit

NOTE: At this point you have the Main menu displayed and have the option of editing customer information, editing packaging information or creating a file for transfer to ZyMOS as described in paragraph 3.3. If you are only editing customer information or packaging information you must EXIT (save changes) when you are finished.

Please select option (1-5). >4<CR>

OK,

ZyPEE USER'S GUIDE

Doc: 20-010-016 Rev: B | Issued: September 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: B
Table of Contents	Rev: B
Pages 1-1 through 1-2	Rev: B
Pages 2-1 through 2-2	Rev: B
Pages 3-1 through 3-3	Rev: B
Pages 4-1 through 4-7	Rev: B
Page 5-1	Rev: B

Table of Contents

1 INTRODUCTION

1.1 OVERVIEW

This user's guide describes the operating principles and usage of the ZyPSIM network error eliminator, ZyPEE. ZyPEE provides the ZyP system user with the experience gained from many custom I.C. designs, as an aid to eliminating potential design errors when converting system logic to silicon using the ZyP standard cell libraries. ZyPEE is a software module which processes ZyPSIM net and output files to detect various types of design errors. Detected errors are explained, each occurrence is flagged, and possible solutions are described.

1.2 SCOPE

The program is intended for use when networks are well developed and satisfactory results have been obtained from ZyPSIM. However, it can be used at any time, if the network has been compiled by ZyPSIM with no errors.

ZyPEE supplements the design verification features of ZyPSIM with "engineering common sense" and many years (and I.C.'s) worth of experience. It should be realized that ZyPEE will not catch every design "gotcha" possible. In fact, the next problem it fails to catch will be incorporated into the next revision of the program. A successful pass through ZyPEE does not guarantee a flawless design. However, it does guarantee that known common design problems will not be repeated. As such, ZyPEE should be considered as a powerful addition to the range of ZyP I.C. design tools.

1.3 RELATED DOCUMENTS

Design verification, test program generation and overall design for successful production are greatly interrelated. It is essential that the following documents be used in conjunction with this user guide to fully understand the concepts which relate to logic simulation and test program generation.

20-010-107	Zy40000 Series CMOS Cell Library, Volumes 1 through 3.
20-010-109	ZyPSIM User's Manual.
20-010-309	ZyPSIM Test Language.
20-010-110	ZyP Specification and Test User's Manual.
20-010-117	Zy50000 Series CHMOS Cell Library, Volumes 1 through 3.
20-010-310	ZySPEC User's Guide.

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, user input is shown underlined. For example:

TYPEME

Where illustrations of commands are shown, lower case text is used to indicate additional input. Items shown enclosed in angle brackets < > are user defined. For example:

COMMAND <filename>

Items enclosed in braces {} are optional. For example:

COMMAND <filename> {<-option>}

2 PRINCIPLES OF OPERATION

2.1 BACKGROUND

ZyPSIM is the ZyP system's primary means of verifying logic networks. However, it provides a "black and white" interpretation of circuit operation and areas of doubt cannot be taken into consideration. Typically, logic configurations which are commonplace in board level or breadboard designs can fail disastrously when implemented as an MOS circuit.

To fill this void, ZyPEE was developed to add the "gray area" of interpretation to aid in generating error free system logic which can be successfully converted to silicon. The basic method is well suited to future expansion and every effort will be made to keep the program as current as possible, with all known network problems detected.

2.2 METHOD

2.2.1 GOTCHAS

Potential design problems are defined as "gotchas" and each problem type is assigned a number: GOTCHA1, GOTCHA2 etc. ZyPEE operates on the same net list used by ZyPSIM and scans for each gotcha in turn. Detected errors are displayed in three phases of output:

a) A brief definition of the offending gotcha is presented.

b) The signal(s), cell(s) or macro(s) involved will be listed.

c) Possible solutions will be described.

NOTE: All of the above are written to the output file but only item b) is displayed at the terminal.

2.2.2 BASIC NETWORK INTEGRITY

In addition to specific gotchas, a general network check is performed and any anomalies are reported. An example of the type of discrepancy which can be detected is a missing signal in a user defined macro. It is possible to define a macro which has a signal name in its argument list that is not used within the macro. This is equivalent to an unused output and is not considered an error by ZyPSIM. ZyPEE will flag such items in its initial network read.

2.3 REQUIREMENTS

ZyPEE assumes that any net list to be operated on has been previously compiled (successfully) by ZyPSIM and is free of basic errors. This relieves ZyPEE of the task of duplicating the syntax checking functions of ZyPSIM. It is therefore essential that an error free ZyPSIM simulation be performed prior to using ZyPEE. Failure to observe this single requirement will almost certainly result in wasted time chasing errors which ZyPSIM would have easily caught.

A successful ZyPEE run is also a requirement prior to executing ZySPEC, a software utility used to create the final specification for the user's design.

3 RUNNING THE PROGRAM

Operation of ZyPEE is similar to ZyPSIM and is extremely straightforward. ZyPSIM .NET files can be processed unmodified by ZyPEE. Several control commands are also possible, but this requires that the ZyPEE commands are placed in the net list. However, they are such that normal ZyPSIM processing is not affected.

3.1 FILES

The input file requirement for ZyPEE is the normal ZyPSIM net list:

filename.NET

In addition to terminal output, all detected errors will be written to an output file named:

filename.GOT

A second output file will be generated if a ZyPSIM generated file suffixed .TPN, .TPS, or .TPF is present in the working directory and if no errors are found in ZyPEE. This output file is called:

filename.TST

The file will be used as an input file for ZySPEC, a program which creates a document containing the final specifications of the user's design. An example of each file is available in section 4 of this document.

3.2 EXECUTION

ZyPEE is invoked by typing:

ZYPEE {<filename>} {<lib>}

where {<filename>} is the filename prefix and {<lib>} is a character representing the desired library (Zy4=Zy40000 CMOS, Zy5=Zy50000 CHMOS).

Alternatively, typing ZYPEE alone will cause the program to prompt for a filename and library. This format is identical to ZyPSIM.

3.3 OUTPUT

After reading the net list, a summary of macro and cell usage will be displayed followed by any basic network anomalies. ZyPEE will then display at the terminal every gotcha as it is being checked along with all occurrences of potential problems. The .GOT output file will contain the gotcha error list, together with the problem definition and possible solutions.

3.4 ZyPEE COMMANDS

3.4.1 COMMAND FORMAT

All ZyPEE commands must be transparent to ZyPSIM so that normal simulations are unaffected. For this reason, ZyPEE commands begin with the ZyPSIM comment character, "*" asterisk. This is followed by the actual ZyPEE command which begins with a period.

It is essential that the "*" be the first character in the line (no leading blanks) and that the period follows immediately in column two. For example:

*.ZYPEE_COMMAND

3.4.2 NOLIST

The NOLIST command is provided to suppress terminal output of any detected errors. All output is still directed to the output file. This feature will be found useful when there are many errors and it is more convenient to browse through the output file using the editor. The NOLIST command format is:

*.NOLIST [G1 G2 ..... Gn]

where G1 through Gn represent optional (blank or comma separated) integers, without arguments; terminal listing for all errors will be suppressed. When arguments are present, terminal output for the specified gotchas will be suppressed. For example:

*.NOLIST 3 5

will suppress terminal display of errors for gotchas 3 and 5.

3.4.3 NOGOTCHA

The NOGOTCHA command disables the checking of the specified gotcha completely. Do not confuse NOLIST and NOGOTCHA, NOLIST suppresses terminal error output, while NOGOTCHA disables error checking.

A NOGOTCHA command must be supplied for each gotcha check to be disabled. The command format is:

*.NOGOTCHA<n>

For example, specifying:

*.NOGOTCHA1

will completely disable all checks for GOTCHA1.

NOTE 1: The *.NOGOTCHA command cannot be applied to GOTCHA7 because its results are essential for the .TST file used by ZySPEC. If *.NOGOTCHA7 is specified by the user the command will simply be ignored.

NOTE 2: The requirement for a specific command for each gotcha allows arguments (ie signal names, cell names or macro names) to be specified, applicable to each gotcha, so that specific items can be excluded from the check.

3.4.4 RUNTIME

The RUNTIME command causes the CPU and I/O time used to be displayed at the end of program execution. The command format is:

*.RUNTIME

The CPU and I/O time will be displayed in minutes, seconds and ticks. There are 330 ticks in one second on PRIME computer.

4 ZyPEE EXAMPLE

4.1 NETLIST

The partial network listing shown below contains two errors in order to demonstrate how ZyPEE operates. The first section contains an example of GOTCHA2, which is a potential clock skew problem that occurs when discrete flip flops are used as a shift register. Next, an analog macro has been defined which does not contain a correct left end interface cell, GOTCHA5.

Both of the above problems would not cause errors in ZyPSIM. However, GOTCHA3 can cause severe problems on silicon and GOTCHA5 cannot be routed without the correct end cell. Further details of these particular gotchas are shown in the output file for this example.

```
TEST.NET
.
.
.
*
*
*
*          EXAMPLE OF GOTCHA #2
*
FF1       FAC1 DIN   CLK   R   Q1   NQ1
FF2       FAC1 Q1    CLK   R   Q2   NQ2
*
*
*
*          EXAMPLE OF GOTCHA #5
*
.MACRO ANALOG1 IN NCTRL PCTRL OUT
SWITCH ASA1 IN NCTRL PCTRL OUT
REC    AJC2
.EOM
*
*
.
.
.
.END
```

FIGURE 4.1 ZyPEE EXAMPLE NET LIST

4.2 EXAMPLE TERMINAL SESSION

The following listing shows the invocation and terminal response when ZyPEE is run on a network containing the errors shown in FIGURE 4.1.

```
ZyPEE TEST Zy4

                        ZyPEE
                    Version 5.1.2 (09-04-84)
                ZyPSIM Network Error Eliminator
            A proprietary product of ZyMOS Corporation
        477 North Mathilda, Sunnyvale, California, 94086, USA

---- Reading file TEST.NET

     NETLIST REPORT:
     -----------------------------------------------
     The following control commands are erroneous or
     not found in the netlist:

---- (Warning) Missing .INPUT ZYP>ZYPSIM>Zy4LIB>LAYOUTCAP statement.

---- (Fatal) Missing .TPERIOD statement.

---- (Warning) Missing .PADS statement.

---- (Fatal) Missing .TPRINT statement.

     The control commands mentioned above are needed
     to run the final netlist in order to transfer
     your circuit to ZyMOS. However, if you are
     checking a partial netlist, the warnings or
     errors may be ignored temporarily.
     -----------------------------------------------

---- Expanding the network.

   1 user defined macros
   0 macro usages
   2 standard cell usages
```

```
---- Checking GOTCHA 1  [Input of mux driven by unbuffered storage
                         element]

---- Checking GOTCHA 2  [Clock skew due to non-telescoping shift elements]

GOTCHA 2:
  Bad connection(s):
    FF1.FAC1.Q1 to FF2.FAC1.Q1

---- Checking GOTCHA 3  [Storage element clock input driven by tri-state]

---- Checking GOTCHA 5  [Analog cell(s) not contained in a .MACRO
                         definition]

GOTCHA 5:
  Invalid left end cap cell (AJC2) exists in macro ANALOG1.

---- Checking GOTCHA 6  [Telescoping and Non-telescoping cells in same
                         .MACRO]

---- Checking GOTCHA 7  [Improper use of input or output pad connections]

GOTCHA 7:
  .PADS is not found, GOTCHA 7 is skipped.

---- Checking GOTCHA 8  [Improper use of Analog cell biasing, power or
                         ground]

---- Checking GOTCHA 9  [Correlation of .PADS and .TPRINT]

GOTCHA 9:
  .PADS is not found, GOTCHA 9 is skipped.

---- Checking GOTCHA 10 [Pad usage within a .MACRO definition]

---- Checking GOTCHA 11 [Coincidence of .MAXTIME value and strobe time]

---- Checking GOTCHA 12 [Improper use of "bare" input pads]

---- Checking GOTCHA 13 [Asynchronous feedback path created in a function]

**** All possible solutions to the problem(s) above
**** are stored in the file TEST.GOT
**** Please check them out.

**** Test commands are incompletely specified.
**** The file TEST.TST was not created.

**** END OF ZyPEE ****
```

4.3 EXAMPLE .GOT OUTPUT FILE

The text of this section is a listing of the TEST.GOT file from the example of paragraph 4.2.

```
ZyPEE (5.1.2) GOTCHAs for TEST.NET          13:22:35  09-04-84


NETLIST REPORT:
-----------------------------------------------
The following control commands are erroneous or
not found in the netlist:


---- (Warning) Missing .INPUT ZYP>ZYPSIM>SLIB>LAYOUTCAP statement.

---- (Fatal) Missing .TPERIOD statement.

---- (Warning) Missing .PADS statement.

---- (Fatal) Missing .TPRINT statement.


The control commands mentioned above are needed
to run the final netlist in order to transfer
your circuit to ZyMOS. However, if you are
checking a partial netlist, the warnings or
errors may be ignored temporarily.
-----------------------------------------------



---- Expanding the network.

   1 user defined macros
   0 macro usages
   2 standard cell usages
```

ZyPEE (5.1.2) GOTCHAs for TEST.NET 13:22:35 09-04-84

GOTCHA #2: PROBLEM DEFINITION (SEVERITY: WARNING)

Stand alone D type flip flops have been used to construct a shift register. The delay produced by inter-stage interconnect lines can cause clock skew between successive stages, resulting in incorrect operation. Discrete shift register construction is dangerous but not necessarily fatal.

Signal(s) affected:

1: FF1.FAC1.Q1 to FF2.FAC1.Q1

GOTCHA #2: POSSIBLE SOLUTIONS

1) Construct all shift registers as telescoping macros. This produces a controlled layout of clock lines common to all stages within the macro.

2) Document the clock lines of shift registers (which use stand alone flip flops) in the .COMMENTS file.

3) Design the register (using a second clock phase) such that up to a half clock period of clock skew can be tolerated.

ZyPEE (5.1.2) GOTCHAs for TEST.NET 13:22:35 09-04-84

GOTCHA #5: PROBLEM DEFINITION (SEVERITY: FATAL)

An analog cell is not described in a user macro definition, or, the macro does not start and end with an appropriate left or right end cell. Analog cells can be freely mixed within a digital circuit, only if the proper interface requirements are met. Definition within a macro and correct selection of the interfacing left and right end cells is mandatory for all analog cells.

Macro(s) affected:

Invalid left end cap cell (AJC2) exists in macro ANALOG1.

GOTCHA #5: POSSIBLE SOLUTIONS

Define the analog function within a macro and/or select the correct end cells from the "AJ" family. Refer to the ANALOG Design Guide (20-010-011) and the "AJ" data sheets for further information.

4.4 EXAMPLE .TST OUTPUT FILE

The partial listing shown below is an example of a .TST output file generated by ZyPEE. The information contained in the .TST file is used by ZySPEC to generate the test specifications for the chip.

An explanation of the column headings follows:

Pin#	Pin number defined in the .NET file
Pad	Pad standard cell name
Type	Pad cell function (i.e. IN, OUT, I/O)
Signal	Signal name for the Pad
Pull up/down	Designation of standard cell pull/up down where applicable
Tri-state	Tester cycle time where tri-state condition exists
Static	Tester cycle time where static current measurement can be made
PU/PD test	Tester cycle time for pull up/down test
Input_buf	Input buffer standard cell name

```
*CHIP.TST generated by ZyPEE (5.1.2) for ZySPEC          11:39:19  09-05-84
*
*Library: Zy40000
*
ZY4
*
*Pin# Pad  Type Signal    Pull up/down  Tri-state  Static  PU/PD test  Input buf
*
  1   PDC1 IN   VDD
  2   PAB1 IN   NCSKBDW
  3   PAA1 IN   NCSKBDR
  4   PCM1 I/O  D0IO                   T@00005                        IAL1
  5   PDA1 IN   NRESETIN  UP=TAF1                 D@00006   d@00001    IAS1
  6   PAD1 IN   R10
  7   PDA1 IN   A0IN      DOWN=TAE2               d@00001   d@00001    IAL1
  8   PDA1 IN   A1IN      DOWN=TAE2               d@00001   d@00001    IAL1
  9   PCM1 I/O  D1IO                   T@00005                        IAL1
 10   PCM1 I/O  D2IO                   T@00005                        IAL1
 11   PCM1 I/O  D3IO                   T@00005                        IAL1
 12   PCM1 I/O  D4IO                   T@00005                        IAL1
 13   PCM1 I/O  D5IO                   T@00005                        IAL1
 14   PCM1 I/O  D6IO                   T@00005                        IAL1
```

5 TYPES OF GOTCHA

The major areas which ZyPEE addresses are outlined in the following sections. Detailed information on each individual gotcha is available in the gotcha output file.

5.1 LAYOUT DEPENDENCY

Although ZyPSIM provides a means of artwork capacitance estimation, there are certain types of potential problems which cannot be detected by this method. In particular, clock skew related failures fall into this category.

5.2 BAD DESIGN PRACTICES

Certain logic configurations which can be implemented in breadboard logic, do not lend themselves to successful MOS integration. Race hazards such as asynchronous self resetting counters fall into this "bad design practice" category.

5.3 ARTWORK COMPATIBILITY

Since the netlist must support the artwork software as well as ZyPSIM, there are some general formatting and procedural requirements of the net list. ZyPEE is well suited to address this issue.

5.4 TEMPORARY PROBLEMS

When ZyP system problems are encountered, ZyPEE will be used (if possible) to make users aware of temporary solutions or alternative methods of achieving a given result. When permanent solutions are found, this type of gotcha check will be removed.

MAILBOX USER'S GUIDE

ZyMOS Corporation believes the information contained herein to be accurate and reliable, however ZyMOS Corporation makes no warranty for the use of this documents contents and assumes no responsibility for any errors which may appear in the document, nor does it make a commitment to update this information. ZyMOS retains the right to make changes to the information or specifications contained herein at any time, without notice.

Doc: 20-020-316 Rev: B

Issued: January 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

Table of Contents

1 INTRODUCTION

1.1 OVERVIEW

MAILBOX is a ZyMOS provided utility which runs on the PRIME and allows users to send and receive messages. It is particularly useful when used in conjunction with a LOGIN.COMI file to pickup mail when you login.

It can be used for general communication when the PRIMOS "MESSAGE" utility program (limited to a one line message) is not convenient, ie: when the message recipient is not logged on or the message is longer than one line. When using a login file to pickup mail, MAILBOX also provides a useful reminder function.

1.2 FEATURES

MAILBOX provides the basic functions of sending and receiving mail. In addition, there are many options which enhance the basic functions and allow users a great deal of flexibility. The following is a brief summary of MAILBOX features:

* Mail can be sent to a single user or multiple users whose names are entered at the terminal, or it can be sent to a group of users whose names are contained in a system file.

* Mail can consist of text entered at the terminal or the contents of a system file.

* Mail can be simply read and deleted, allowed to accumulate, or saved to a system file.

* Direct spooling of mail is possible to local or remote sites.

* Notification of receipt of mail to any logged on user is provided automatically by the MESSAGE utility.

* Multiple systems are supported over PRIMENET

* Status information can be obtained on the state of mail sent and also on the systems served by MAILBOX.

* Users can set several attributes for maximum convenience.

1.3 AVAILABILITY

MAILBOX operates by saving messages in system files on one particular system. The configuration of the ring network, linking the various systems, will determine which systems can utilize MAILBOX over PRIMENET.

Section 5 provides details of how to determine which systems are served by MAILBOX.

1.4 GENERAL OPERATION

MAILBOX is invoked using several options to specify the particular functions required. Options are specified by using option keywords which always begin with a "-" (dash) character. Some option keywords also require additional (option text) arguments. The general form of command line is as follows:

MAILBOX -keyword1 {option_text} -keyword2 {option_text} .... etc.

where -keyword is the keyword for the required function and option_text is data appropriate to that function.

The sections describing each function provide details on option keywords and any required option text. If option text is required but not supplied, the program will prompt for missing data.

1.5 HELP FEATURE

Operating information can be obtained by using:

MAILBOX HELP or MAILBOX -HELP

This will display all possible types of help available "on line" through the program. Currently, there are three help options available:

NEWS lists the latest revisions and coming updates.

USAGE displays a brief summary of functions and syntax.

FULL displays full operating information - this document.

They can be obtained by:

MAILBOX <help_type> or MAILBOX -HELP <help_type>

2 SENDING MAIL

2.1 OVERVIEW

Mail can be sent in various ways dependent on which options are supplied when MAILBOX is invoked. In normal use, mail will be sent to a single user by text entered directly at the terminal. Mail to more than one user simultaneously is easily achieved from names entered at the terminal or from a system file. The actual text of the message can also be contained in a system file.

2.2 TERMINAL ENTRY

To send mail which will be entered from the terminal to a single user, the following command is used.

MAILBOX -SEND user_name

where user_name is the login name of the user to whom the mail is directed. If user_name is omitted, MAILBOX will prompt for it.

The "message entry mode" will be entered and the ">" prompt will be displayed. At this point the text to be sent should be entered. To terminate text entry, enter a null line (carriage return). The text will be sent immediately after termination of entry.

NOTE: A line containing only blanks will be interpreted as a null line, ie: termination of text. For this reason, it is not possible to separate text within the message using blank lines. Use a period or other "low visibility" character for this purpose.

To send mail to more than one user, simply specify each name separated by blanks when invoking MAILBOX as follows:

MAILBOX -SEND user_name1 user_name2 ..... user_nameN

Alternatively, if the names are omitted on the command line, more than one name can be entered in response to the program's prompt for a user name.

2.2.1 MESSAGE LIMITS

When using terminal entry to create the message text, there is a limit of 100 lines for the message length. The message will be automatically terminated and transmitted if this limit is reached.

There is also a limit of 80 characters per line. This limit applies to messages created by terminal entry or by use of a system file.

2.2.2 CONFIRMATION OF ENTRY

It is possible to display the message prior to transmission by using the the check option as follows:

```
MAILBOX -SEND user_name -CHECK
```

If this is used, the text will be displayed after termination of entry and an "OK to send" option will be offered. If the message is not sent, the chance to save the text in a file (for editing) will be offered. If the text is not saved, the chance to re-enter the message entry mode will be offered.

It is not possible to edit text within the program. If you anticipate long messages which will require edits before transmission, use the message from file option described in the following section.

2.2.3 MAKING A COPY OF WHAT YOU SEND

To make a copy of message text entered at the terminal, the copy option is used as follows:

```
MAILBOX -COPY copy_filename -SEND user_name
```

Where copy_filename is the name of the file in which to save the message text. If copy_filename exists, it will be overwritten.

2.3 FILE ENTRY

The text of the message can be contained in a system file. This is useful when the message is long and may require editing, or you wish to send an existing file. The names of recipients may also be contained in a file.

This feature allows several files to be set up for different groups of people to whom you regularly send mail. These features are controlled using options LIST and MESSAGE.

2.3.1 MESSAGE TEXT FROM A FILE

Create the desired message in a system file prior to invoking MAILBOX. There is no limit to the number of lines in the file and blank lines do not terminate the message so they may be freely used.

Specify the filename when invoking MAILBOX, using the MESSAGE option as follows

```
MAILBOX -MESSAGE message_filename
```

where message_filename is a file containing the message text.

2.3.2 USER NAMES FROM A FILE

Create a file containing a list of user names. Specify only one user name per line and use only the first 6 columns. Any text after column 6 will be ignored so additional text can be used as comments. There is a limit of 30 names per file.

Specify the filename when invoking MAILBOX, using the LIST option as follows:

MAILBOX -LIST list_of_names_filename

where list_of_names_filename is a file containing the login names of the desired recipients. To send mail using text from a file to a specified group of users in a file, invoke MAILBOX as follows:

MAILBOX -MESSAGE message_filename -LIST list_of_names_filename

2.4 MIXED TERMINAL ENTRY AND FILE MODE

Any combination of terminal and file entry can be used. For example, to send mail from a file to a single terminal entered user name, use:

MAILBOX -SEND user_name -MESSAGE message_filename

To send a terminal entered message to names in a file, use:

MAILBOX -LIST list_of_user_names

2.5 BUSY USERS

Occasionally, mail may not be sent if a user's message file is in use. If this occurs and the message text was entered from the terminal, you will be asked if you wish to save the message in a file. The message can then be sent later using the MESSAGE option.

2.6 UNKNOWN USERS

When mail is sent to a new user (unknown to MAILBOX) for the first time, an "invalid user" message will be displayed followed by an option to add that user to the valid list. Users are not automatically added to prevent names with typos from entering the MAILBOX system.

If the add option is taken, the primary system name for the user will be requested, ie: the normal login system for the user. If an incorrect system is entered, the user will still receive mail, but will not receive notification of receipt of mail.

2.7 NOTIFICATION OF RECEIPT OF MAIL

When mail is sent, MAILBOX will send notification via the PRIMOS MESSAGE utility to each recipient. This will be effective only for those users who are logged on. The system name specified for the user (when added to the list of valid users) will be used in the message. The construct is as follows:

```
M user_name -NOW -ON ZSVn
YOU HAVE SOME MAIL^207
```

where n is the system name known to MAILBOX. If n was incorrectly supplied, the user will not receive notification.

NOTE: When the recipient is not logged on, the notification will not take place and the normal "Unknown addressee" returned by MESSAGE is suppressed. However, when sender and recipient are on different systems, the "Unknown addressee" message is not suppressed.

2.8 MESSAGE IDENTIFICATION

In addition to the actual text of the message, each message will be terminated with indentification information of the form shown below.

```
END OF MESSAGE FROM USER:  STU ON SUN, MAY 01 1983 AT 12:12:39
```

If the message was sent to more than one user, the names of each recipient will also be included after the message text.

2.9 RUNNING MAILBOX FROM A CPL FILE

If you wish to send mail by invoking MAILBOX from your own CPL file, the -CPL option must be used. For example:

```
&ARGS USER_NAME
&DATA MAILBOX -SEND %USER_NAME% -CPL
Here is a message sent by a CPL file. It works!

&END
```

NOTE: The blank line before &END must be included to terminate message entry mode.

The -CPL option suppresses the message notification feature. Failure to use -CPL when sending mail from a CPL file will cause MAILBOX to hang.

3 RECEIVING MAIL

The "standard" method of receiving mail is to display it directly at the terminal. Several options exist regarding the disposition of the message after display as described in the following sections.

3.1 READ WITH DELETE

By invoking MAILBOX with no options, any existing mail will be displayed. After the display is completed, mailbox contents will be automatically deleted.

When messages are long, the display will pause each 23 lines and prompt for more. If the display is aborted while paused, by a "NO" answer, the message will not be deleted.

3.2 READ ONLY

To read mail without deletion, use the CHECK option as follows:

```
MAILBOX -CHECK
```

This is the recommended way of reading mail, especially when doing so via your LOGIN.COMI file.

3.3 HARD COPY

A printout of your mailbox contents can be obtained by using the SPOOL option as follows:

```
MAILBOX -SPOOL
```

The message:

```
MAIL_FOR_USER (n RECORDS) IS IN THE QUEUE AS PRTXXX
```

will be returned where USER is your login name, n is the number of records spooled, and XXX is the spool queue number. The banner on the header page of the listing will be MAIL_FOR_USER

3.3.1 REMOTE SITES

The listing may be directed to any printer which is available to you over PRIMENET by using the AT option as follows:

```
MAILBOX -SPOOL -AT printer_id
```

where printer_id is the name of the required remote printer.

3.4 SAVING MAIL

Your mailbox contents may also be copied to a system file. This allows convenient accumulation of mail. To invoke this feature, use the SAVE option as follows:

MAILBOX -SAVE save_filename

where save_filename is the name of the file in which to save mail. If save_filename exists, the contents of your mailbox will be appended to it, so that mail is easily accumulated.

The contents of your mailbox will be automatically deleted after saving. To prevent this, use the -CHECK option as follows:

MAILBOX -SAVE save_filename -CHECK

This will cause mail to be displayed as normal as well as being written to save_filename without deletion.

If you wish to display, save, and delete mail simutaneously, use:

MAILBOX -SAVE save_filename -CHECK -DELETE

Useful tip: If you use the [DATE -<option>] function in the save_filename, your mail will be saved to a file which automatically includes the date and/or time (dependent on -<option>) in the filename. See the CPL User's Guide for more information on the [DATE] function.

3.5 DELETING MAIL

To force deletion of mail without reading, use the DELETE option as follows:

MAILBOX -DELETE

This may be used in conjunction with the SPOOL option to obtain a printout and then to delete mail.

MAILBOX -SPOOL -DELETE

The delete option will always be executed last no matter where it appears on the command line.

4 REPLY MAIL

When mail is sent to you, a reply message can be automatically transmitted to the sender. This is useful if you are on vacation etc., to notify the sender that you will not receive the message until your return.

4.1 CREATING A REPLY

To create a reply message, use the REPLY option as follows:

MAILBOX -REPLY

The "message entry mode" will be entered and text should be entered as for a normal mailbox message and is subject to the same limitations as a normal message.

This text will be received by all users who send you mail for as long as the reply message is present.

4.2 CANCELLING A REPLY

To cancel reply messages, use the NOREPLY option as follows:

MAILBOX -NOREPLY

This will completely delete any existing reply messages. Whenever the REPLY option is used, any existing reply messages will be overwritten.

5 MAILBOX STATUS

Status is available for several types of MAILBOX information. By using the appropriate option, the "read" status of single or multiple users, the systems served by MAILBOX, or a list of valid users on each system may be determined.

5.1 MAIL STATUS

In order to determine whether mail to a particular user has been picked up or not, the STATUS option can be used as follows:

MAILBOX -STATUS user_name

where user_name is the login name of the recipient. The time of the last message received and picked up will be displayed alongside the user name and system name for that user.

If ALL is used instead of user_name, the status of all users with messages outstanding will be displayed.

5.2 SYSTEM STATUS

To determine which systems are served by MAILBOX, use the SYSTEMS option as follows:

MAILBOX -SYSTEMS

The names of all systems which MAILBOX is able to communicate with will be displayed. The system which contains the message files will also be displayed. Users on other systems will notice some degradation in speed when reading mail, since the message will be transmitted over PRIMENET.

5.3 VALID USER STATUS

To determine which users are known to MAILBOX for a particular system, use the VALID option as follows:

MAILBOX -VALID system_name

where system_name is the name of the system for which a list of valid users is required. If ALL is used instead of system_name, valid users for all systems will be displayed.

6 SETTING USER ATTRIBUTES

By using the options described in the following sections, you can configure MAILBOX to provide the most convenient mode of operation for your needs.

6.1 TERMINAL TYPE

Currently, MAILBOX will take advantage of the TVI950 terminal features when so directed. Features may be added from time to time which enhance performance when using a TVI950. To tell MAILBOX you are using a TVI950 terminal, use the TVI950 option as follows:

MAILBOX -TVI950

This need only be done once. If you change to another type of terminal, use

MAILBOX -NOTVI950

This is the default mode of operation.

6.2 NOTIFICATION CONTROL

When you are logged on and someone sends you mail, you will receive a "YOU HAVE SOME MAIL" notification message. To suppress this notification, use the REJECT option as follows:

MAILBOX -REJECT

This will suppress all notification messages until they are enabled again. To enable messages, use the ACCEPT option as follows:

MAILBOX -ACCEPT

This is the default mode of operation.

7 SUMMARY OF FUNCTIONS

The following list summarizes the functions available and shows the required syntax for invocation. The minimum abbreviations for each option keyword are shown capitalized.

Items shown enclosed in <angle_brackets> are user specified option text. If they are not specified on the command line, MAILBOX will prompt for the missing data. Items shown surrounded by braces {} are optional.

MAILBOX	{<options>}	When used without options, displays and then deletes any existing messages.
-ACcept		Enables notification of receipt of mail.
-AT	<printer_id>	Used with -SPOOL to direct output to a remote location specified by <printer_id>.
-Check		Causes confirmation of message text to be displayed in SEND mode, prevents deletion of message in RECEIVE mode.
-COPy	<filename>	Used to make a copy of message text when using terminal entry.mode. Text is saved to <filename>.
-CPL		Must be used when MAILBOX is run from a CPL file and -SEND or -MESSAGE is used ie: Send Mode.
-DELete		Deletes any existing messages.
-Help	{<help_type>}	Displays help options available or displays help for specified <help_type>.
-List	<filename>	Specifies that the message is to be sent to a list of user names contained in <filename>.
-Message	<filename>	Specifies that the message text to be sent is the contents of <filename>.

-NOReply			Deletes any existing reply messages.
-NOTVI950			Causes MAILBOX not to use any TVI950 terminal features.
-REJect			Disables notification of receipt of mail.
-REPly			Enters message entry mode for creation of a reply message.
-SAve	<filename>	{-Check}	Copies and appends any existing mail to <filename>. Mail is then deleted unless -CHECK is specified.
-SEnd	<user_names>	{-Check}	Enter terminal message entry mode for mail to be sent to <user_names>. Display text prior to transmission if -CHECK is specified.
-SPool	{-AT <printer_id>}		Causes MAILBOX contents to be listed on the default printer, or a remote printer if -AT is specified.
-STatus	<user_name> <ALL>		Displays mail receipt status for specified <user_name> or <ALL> users.
-SYstems			Displays names of systems served by MAILBOX.
-TVI950			Causes MAILBOX to utilize features of TVI950 terminal.
-Valid	<system_name> <ALL>		Displays list of valid users on <system_name> or <ALL> systems.

ZL USER'S GUIDE

Doc: 20-010-416 Rev: A	Issued: September 1984	ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA. P.O. BOX 62379, SUNNYVALE, CA 94088 TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

Table of Contents

1 INTRODUCTION

1.1 SCOPE

This user's guide describes the operating principles and usage of the ZL utility program. ZL provides ZyP Design System Prime user the capability to display the ZyPSIM Logic Simulator Models for all standard cells.

2.1 OVERVIEW

The ZL ZyP Listing Utility allows the ZyP Prime user to quickly reference any cell model data including cell syntax for inputs and outputs, the cell primitive(s) itself, information on the number of transistors, and grid size. It also includes comments on the update history of the cell.

1.3 RELATED DOCUMENTS

The ZyPSIM Reference Manual (Doc. No. 20-010-209) explains how to interpret the models.

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, the following conventions are used.

a. All user inputs to the computer are shown underlined.

b. <CR> indicates the single typed key RETURN.

2 OPERATION

ZL operates in two different modes. The first is a menu mode, where the program prompts the user. The second is command line entry, where the user instructs the program. Both modes are described in detail in this section.

2.1 MENU MODE

When in the menu mode, ZL will prompt the user for the Cell or Model Name. ZL will then prompt for the Technology either "Zy4", "Zy5", or "user_lib" where Zy4=Zy40000 CMOS technology and Zy5=Zy50000 CHMOS technology and user_lib = a user installed library. Each prompt answer must be followed with a <CR>. When the last prompt is answered the program will display the ZyPSIM Logic Simulator Model for the cell. ZL also provides a help function option when it prompts for the technology. The following is a sample of executing ZL in the Menu Mode for the Inverter Cell IAA1 and CHMOS technology. During execution the "HELP" mode has been accessed to display the help menu.

```
OK, ZL <CR>

                                ZL
                        ZyP Listing Utility
                    Version 5.1.2     Release Date: 09/11/84
                A  proprietary  product  of  ZyMOS  Corporation
            477 North Mathilda, Sunnyvale, California, 94086, USA

ENTER CELL OR MODEL NAME: IAA1 <CR>

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)>? <CR>

Zy4             ZyMOS Zy40000 series 5u silcon gate CMOS library.
Zy5             ZyMOS Zy50000 series 3u CHMOS library.
'user_lib'      User defined and installed library.
<CR>            Carriage return, program exit.
?               Help, type this text.

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)>ZY5 <CR>

   ZyPSIM logic simulator model:  IAA1                (3u Silicon Gate CHMOS)

*          ZyPSIM model -- generated by ZyPDE, version 3.05 [04-19-83]
*
*          5 IAA1 A -- 3u SILICON GATE CHMOS
*          Function -- INVERTER
*
*          Input capacitance values:
*          IN = .13
*
.MACRO IAA1 IN OUT
.OFF
IAA1 INV 3.2+13,1.2+7:5+16,3.7+8 IN=IN:.13 OUT=OUT
.INFO 28 2 0 0
.EOM
*
More <yes> ? <CR>
**           Released on: 10/05/83
*

OK,
```

2.2 COMMAND LINE ENTRY

The user can also enter all inputs on the command line. The format for the command line entry is:

ZL (cell or model name) (technology ZY4/ZY5/USER_LIB)

The user can also access the "HELP" mode by typing ZL ?<CR>. This will access ZL and display the "MENU" for technology and help as shown in the Menu Mode.

The following is the command line entry to display the IAA1 CHMOS Inverter Cell Logic Simulator Model.

```
OK, ZL IAA1 ZY5 <CR>


                                ZL
                        ZyP Listing Utility
                    Version 5.1.2     Release Date: 09/11/84
                A proprietary product of ZyMOS Corporation
            477 North Mathilda, Sunnyvale, California, 94086, USA


   ZyPSIM logic simulator model:  IAA1                (3u Silicon Gate CHMOS)

*          ZyPSIM model -- generated by ZyPDE, version 3.05 [04-19-83]
*
*          5 IAA1 A -- 3u SILICON GATE CHMOS
*          Function -- INVERTER
*
*          Input capacitance values:
*          IN = .13
*
.MACRO IAA1 IN OUT
.OFF
IAA1 INV 3.2+13,1.2+7:5+16,3.7+8 IN=IN:.13 OUT=OUT
.INFO 28 2 0 0
.EOM
*
More <yes> ? <CR>
**           Released on: 10/05/83
*
OK>
```

ZyMEM USER'S GUIDE

Doc: 20-010-214 Rev: B | Issued: September 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: B
Table of Contents	Rev: B
Page 1-1	Rev: B
Pages 2-1 through 2-3	Rev: B
Page 3-1	Rev: B
Pages 4-1 through 4-8	Rev: B

TABLE OF CONTENTS

1

1 INTRODUCTION

1.1 OVERVIEW

This user's guide describes the operating principles and usage of the ZyP Memory Utility (ZyMEM). ZyMEM has two separate functions. First, ZyMEM is a silicon compiler that allows ZyP design system user's to create Random Access Memories (RAM), customized programmable Read Only Memories (ROM) and Programmable Logic Arrays (PLA). The operator selects the appropriate options and ZyMEM automatically selects the proper artwork elements; generates the ZyPSIM compatible MACRO netlist; creates the supporting artwork files for placement, routing, and photomask generation; and produces the specific Data Sheet for RAMs and PLAs.

ZyMEM also provides the capability to convert Intel object code files to ZyPSIM format or 80C49 plot files.

1.2 SCOPE

This program is intended for use during network development. It provides a means of generating customized ZyPSIM memory models required during the system logic definition and design, as well as producing compatible artwork models for later use by other software.

1.3 RELATED DOCUMENTS

IMPORTANT: It is essential that the ZyP Memory User's Guide shown below be used in conjunction with this document.

20-010-117	Zy50000 CHMOS Design Guide.
20-010-107	Zy40000 CMOS Design Guide.
20-010-009	ZyPSIM User's Guide.
20-010-014	ZyP Memory User's Guide
20-010-019	Zy80C49 User's Guide

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, the following conventions are used.

a. All user inputs are shown underlined.

b. <CR> indicated the single typed key RETURN.

2 PRINCIPLES OF OPERATION

2.1 BACKGROUND

The ZyP Design system provides cells for designing RAM, ROM, and PLA memory elements. All memory elements except Zy50000 ROM are compiled. The Zy50000 ROM is available as predefined cells. All memory elements are available for direct use in simulation. ZyMEM generates custom data sheets for each of the compiled memory elements. Generic data sheets are available for each type of compiled memory element and a specific data sheet is available for the Zy50000 ROM. These data sheets are located in the ZyPLIB Memories, Volume 1 (Document No. 20-010-114).

ZyMEM allows the user to define the cells, as well as customize the ROM and PLA cells by loading the program information.

2.2 MEMORY FUNCTIONS

ZyMEM creates a logic model file (net list) and the artwork files with the user supplied custom cell code name. The files created are:

a) Net list, identified as XXXX (where XXXX=user specified code name). This file contains the Macro for logic simulation by ZyPSIM and for chip development by the ZyP System.

b) Translator Model, identified as XXXX.XLT. Contains cell input/output identification that translates net list connection data into data suitable for routing.

c) Artwork model, identified as XXXX.ART. Contains cell dimension and connection location data for cell placement and interconnection.

d) Data Sheet, identified as XXXX.DSH. Contains all the summary information such as; performance, physical size, and port locations.

e) Plot data file, identified as ZC4XXXXB or ZC5XXXXB. ZC4 identifies 5u plot data and ZC5 identifies 3u plot data. The plot data file provides internal plot information for each cell.

2.3 80C49 FUNCTIONS

The 80C49 ROM option of ZyMEM creates one of the two following files:

a) ZyPSIM ROM code format file, identifies as XXXXZ. This file contains the ROM code in ZyPSIM format.

b) Plot data file, identified as ZC5XXXXA. The plot data file provides the program pattern for the 80C49 processor.

2.4 REQUIREMENTS

The code name for your RAM, ROM, or PLA cell must be a four character name with one alpha character followed by three numeric digits (e.g., C451). The only restriction for the alpha characters A, P, or Z cannot be used. **Do not use the characters A, P, or Z as your cell name.**

The user must include the .LIBRARY command in the simulation file so that the coded logic model may be used in the simulation. Refer to the ZyPSIM User's Manual for an explanation of the .LIBRARY command.

Prior to accessing ZyMEM, the user should create code files if they are to be used to program ROM or PLA cells. Code files can be created interactively during cell customization, however creating the code files prior to accessing ZyMEM speeds the process and has less chance for error.

2.3.1 ROM CODE FILE FORMAT

Code files for ZyP ROM cells are an address/data sequence, preferably in order of ascending address and may be in binary, octal, or hexadecimal format. The code file can be formatted as shown in the ZyPSIM User's manual for coding ROM primitives. Also, ZyCOMP-4 assembly code output is directly acceptable. ZyMEM will read the code file, place the addresses in ascending order, fill in any unspecified addresses with "0" data, and partition the data into columns for insertion into the individual ROM primitives of the telescoping elements. An example of a HEX format code file for a 512 x 4 ROM is shown below.

```
*** TEST CODE FOR 512 WORD ROMS ****
*** DATA IS SAME AS LSB OF ADDRESS ****
*** WHEN BROKEN INTO COLUMNS IT IS A BINARY STRING
000/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
020/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
040/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
060/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
080/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
0A0/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
0C0/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
0E0/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
100/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
120/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
140/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
160/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
180/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
1A0/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
1C0/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
1E0/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F/0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F
```

2.3.2 PLA CODE FILE FORMAT

Code files for PLA elements are essentially a binary equivalent of the PLA product terms as shown in the PLA section of the ZyP Memory User's Guide. The actual entries are a sequence of "1", "0", or "-" symbols indicating the presence of transistors in the array products. Each line of program code represents one product term and comments may be placed on the same line following an "@" symbol as shown in the following example. The following example shows the code file for a BCD UP/DOWN COUNTER.

```
*** THIS FILE CONTAINS THE PLA CODE FOR IMPLEMENTING
*** A CD4510 BCD U/D COUNTER IN A PLA CELL
*** THE INPUT AND OUTPUT PIN ASSIGNMENTS ARE AS FOLLOWS:
*** PRE U_D R NCI A B C D Q1 Q2 Q3 Q4 (INPUTS)
*** Q1 Q2 Q3 Q4 NCO N.C. N.C. N.C.  (OUTPUTS)
********** PROGRAM CODE FOLLOWS **********
+  01010101--------10010101/00000---  @ CNT DOWN 1-0
+  01010101--------01100101/10001---  @ CNT DOWN 2-1
+  01010101--------10100101/01001---  @ CNT DOWN 3-2
+  01010101--------01011001/11001---  @ CNT DOWN 4-3
+  01010101--------10011001/00101---  @ CNT DOWN 5-4
+  01010101--------01101001/10101---  @ CNT DOWN 6-5
+  01010101--------10101001/01101---  @ CNT DOWN 7-6
+  01010101--------01010110/11101---  @ CNT DOWN 8-7
+  01010101--------10010110/00011---  @ CNT DOWN 9-8
+  01010101--------01010101/10011---  @ CNT DOWN 0-9
+  01100101--------01010110/10011---  @ CNT UP 8-9
+  01100101--------10101001/00011---  @ CNT UP 7-8
+  01100101--------01101001/11101---  @ CNT UP 6-7
+  01100101--------10011001/01101---  @ CNT UP 5-6
+  01100101--------01011001/10101---  @ CNT UP 4-5
+  01100101--------10100101/00101---  @ CNT UP 3-4
+  01100101--------01100101/10101---  @ CNT UP 2-3
+  01100101--------10010101/01001---  @ CNT UP 1-2
+  01100101--------01010101/10001---  @ CNT UP 0-1
+  01100101--------10010110/00000---  @ CNT UP 9-0
+  10--01--01100110--------/00000---  @ PRESET 10
+  10--01--10010110--------/10011---  @ PRESET 9
+  10--01--01010110--------/00011---  @ PRESET 8
+  10--01--10101001--------/11101---  @ PRESET 7
+  10--01--01101001--------/01101---  @ PRESET 6
+  10--01--10011001--------/10101---  @ PRESET 5
+  10--01--01011001--------/00101---  @ PRESET 4
+  10--01--10100101--------/11001---  @ PRESET 3
+  10--01--01100101--------/01001---  @ PRESET 2
+  10--01--10010101--------/10001---  @ PRESET 1
+  10--01--01010101--------/00001---  @ PRESET 0
+  ----10----------------/00000000  @ RESET
```

3 RUNNING THE PROGRAM

Before accessing ZyMEM all code files should be ready. ZyMEM gives the option of inserting code data interactively or inputting code data from a user prepared file. Inputting code data interactively is not recommended, because of the greater chance of input error.

3.1 ZyMEM ACCESS

Users access ZyMEM directly in the ZyP Design System by typing ZyMEM. ZyMEM then displays menu choices for selection of the memory element type: RAM, ROM, or PLA. Each cell type has its own set of prompts that are displayed for user selection. The prompts provide for recovery from incorrect selections by offering the ability to return to the previous or main menus. These options are not shown in the menus, but are shown in the Zy40000 RAM example in Section 4. They are described in the following paragraph and are also available by typing **H** or **HELP** for any menu selection. Section 4 also contains examples of user inputs for each cell type.

3.2 INPUT OPTIONS

The following input options are available for each menu displayed and are in addition to the options shown by a specific menu.

H or Help - Displays a help menu with the following options:

0 (Zero) - Return to the previous menu. This command is repetitive and each usage will move back one menu.

@ - Return to the first (main) menu. All previous entries are deleted when this option is selected.

4 ZyMEM EXAMPLES

The following are examples of three different sessions creating a 64 word by 8 bit RAM, a 512 X 4 bit ROM, and a 12 input / 8 output telescoping PLA.

4.1 RAM EXAMPLE

The following RAM example, except for the underbar to show operator input, is actual ZyMEM screen output during a session creating a RAM of 64 words by 8 bits. The procedure for creating either Zy40000 CMOS RAM or Zy50000 CHMOS RAM is the same.

ZYMEM<CR>

```
                        ZyMEM
                  ZyP Memory Utility
              Version 5.1.3     Release Date: 09/22/84
          A proprietary product of ZyMOS Corporation
       477 North Mathilda, Sunnyvale, California, 94086, USA

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY4<CR>

Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>2<CR>

Enter word capacity:

  (1) 512
  (2) 1024
  (3) 2048

Select 1, 2 or 3. (<CR>:Quit)>0<CR>
```

NOTE: By entering 0<CR> as an entry ZyMEM returns to the previous menu, allowing a correction to the previous entry.

```
Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>1<CR>

Enter word capacity. (16 <= # of words <=128) (<CR>:Quit)>64<CR>
```

```
Enter number of bits per word. (32 bits max) (<CR>:Quit)>16<CR>

Enter MACRO name for the net list.
(1 alpha. + 3 digits, e.g., R001) (<CR>:Quit)>@<CR>
```

NOTE: By entering @<CR> ZyMEM goes back to the beginning menu and allows a complete new start.

```
Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY4<CR>

Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>1<CR>

Enter word capacity. (16 <= # of words <=128) (<CR>:Quit)>64<CR>

Enter number of bits per word. (32 bits max) (<CR>:Quit)>8<CR>

Enter MACRO name for the net list.
(1 alpha. + 3 digits, e.g., R001) (<CR>:Quit)>R401<CR>

**** The RAM logic model is located in file R401
**** The translator model is located in file R401.XLT
**** The artwork model is located in file R401.ART
**** The data sheet is located in file R401.DSH
**** The plot data is located in file ZC4R401B

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)><CR>
```

4.2 ROM SESSION

ROM has two different configurations for Zy40000 CMOS and Zy50000 CHMOS ROM. Two examples are shown one for Zy40000 CMOS ROM and Zy50000 CHMOS ROM. The terminal sessions shown are actual ZyMEM screen output during a session creating and coding a ROM. The only exception is that user inputs are bold and underlined.

4.2.1 Zy50000 ROM SESSION

The following terminal session shows creating a 2K X 8 Zy50000 CHMOS ROM>

ZYMEM<CR>

```
                              ZyMEM
                        ZyP Memory Utility
                  Version 5.1.3     Release Date: 09/22/84
              A proprietary product of ZyMOS Corporation
          477 North Mathilda, Sunnyvale, California, 94086, USA

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY5<CR>

Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>2<CR>

  (1) 2K x 8 ROM
  (2) 80C49 plot file

Select 1 or 2. (<CR>:Quit)>1<CR>

Enter MACRO name for the net list.
(1 alpha. + 3 digits, e.g., R001) (<CR>:Quit)>R502<CR>

The format of the ROM code is:

A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 / Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
Where Q8 is the least significant bit.

How do you want to load the ROM code?

  (1) from a ROM code file
  (2) load the ROM code interactively
  (3) no ROM code to load

Select 1, 2 or 3. (<CR>:Quit)>1<CR>
```

```
Enter the filename. (<CR>:Quit)>ROM.2KX8<CR>

Enter data type of the ROM code:

  (1) in hex-decimal
  (2) in octal
  (3) in binary

Select 1, 2, or 3. (<CR>:Quit)>1<CR>

**** The ROM logic model is located in file R502
**** The translator model is located in file R502.XLT
**** The artwork model is located in file R502.ART
**** The plot data is located in file ZC5R502A

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)><CR>
```

2.4.2 Zy40000 ROM SESSION

The following is a terminal session creating a 512 X 8 bit Zy40000 CMOS Compiled ROM.

```
ZYMEM<CR>

                                ZyMEM
                          ZyP Memory Utility
                    Version 5.1.3     Release Date: 09/22/84
               A proprietary product of ZyMOS Corporation
            477 North Mathilda, Sunnyvale, California, 94086, USA

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY4<CR>

Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>2<CR>

Enter word capacity:

  (1) 512
  (2) 1024
  (3) 2048

Select 1, 2 or 3. (<CR>:Quit)>1<CR>
```

```
Enter number of bits per word. (32 bits max) (<CR>:Quit)>8<CR>

Enter MACRO name for the net list.
(1 alpha. + 3 digits, e.g., R001) (<CR>:Quit)>R402>CR>

The format of the ROM code is:

A8 A7 A6 A5 A4 A3 A2 A1 A0 / Q1 Q2 Q3 Q4
Where Q4 is the least significant bit.

How do you want to load the ROM code?

  (1) from a ROM code file
  (2) load the ROM code interactively
  (3) no ROM code to load

Select 1, 2 or 3. (<CR>:Quit)>1<CR>

Enter the filename. (<CR>:Quit)>ROM.CODE512X8<CR>

Enter data type of the ROM code:

  (1) in hex-decimal
  (2) in octal
  (3) in binary
  (4) in ZyCOMP4 assembly code

Select 1, 2, 3 or 4. (<CR>:Quit)> 1<CR>

**** The ROM logic model is located in file R402
**** The translator model is located in file R402.XLT
**** The artwork model is located in file R402.ART
**** The plot data is located in file ZC4R402B

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)><CR>
```

4.3 PLA SESSION

The following PLA example, except for the underbar to show operator input, is actual ZyMEM screen output during a session creating and coding a Zy50000 CHMOS U/D COUNTER from a 12 input / 8 output compiled PLA.

ZYMEM<CR>

ZyMEM
ZyP Memory Utility
Version 5.1.3 Release Date: 09/22/84
A proprietary product of ZyMOS Corporation
477 North Mathilda, Sunnyvale, California, 94086, USA

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> **ZY5<CR>**

Types of memory cells:

(1) RAM
(2) ROM
(3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>**3<CR>**

Enter # of inputs. (24 bits max) (<CR>:Quit)>**12<CR>**

Enter # of outputs. (12 bits max) (<CR>:Quit)>**8<CR>**

Enter # of products. (4096 terms max) (<CR>:Quit)>**32>CR>**

Enter MACRO name for the net list.
(1 alpha. + 3 digits, e.g., R001) (<CR>:Quit)>**R503<CR>**

The format of the PLA code is:

```
 N N N N N N N N N N N N 00000000
IIIIIIIIIIIIIIIIIIIIIIII UUUUUUUU
NNNNNNNNNNNNNNNNNNNNNNNN TTTTTTTT
000000000000000000111111 00000000
112233445566778899001122 12345678
```

How do you want to load the PLA code?

(1) from a PLA code file
(2) load the PLA code interactively
(3) no PLA code to load

Select 1, 2 or 3. (<CR>:Quit)>**1<CR>**

Enter the filename. (<CR>:Quit)>**PLA.CNTRCODE<CR>**

```
**** The PLA logic model is located in file R503
**** The translator model is located in file R503.XLT
**** The artwork model is located in file R503.ART
**** The data sheet is located in file R503.DSH
**** The plot data is located in file ZC5R503A

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)><CR>
```

4.4 CONVERTING INTEL OBJECT FILE TO ZYPSIM FORMAT

The following example, except for underlining to show user input, is an actual terminal session converting an Intel object code file into ZyPSIM format for insertion in an 80C49 ROM.

```
ZYMEM<CR>


                                ZyMEM
                          ZyP Memory Utility
                    Version 5.1.3     Release Date: 09/22/84
                  A proprietary product of ZyMOS Corporation
              477 North Mathilda, Sunnyvale, California, 94086, USA

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY5<CR>

Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>2<CR>

  (1) 2K x 8 ROM
  (2) 80C49 plot file

Select 1 or 2. (<CR>:Quit)>2<CR>

Convert Intel object file into

  (1) ZyPSIM format
  (2) layout plot file

Please select one. (1 or 2) (<CR>:Quit)>1<CR>

Enter Intel object filename. (<CR>:Quit)>MOVX.80C49<CR>

What is the default data? (e.g., FF) (<CR>:Quit)>00<CR>

**** The code with ZyPSIM format is located in MOVX.80C49Z

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)><CR>
```

4.5 CONVERTING INTEL OBJECT FILE TO LAYOUT PLOT FILE

The following example, except for underlining to show user input, is an actual terminal session converting an Intel object code file into a Layout Plot file for insertion in an 80C49 ROM.

```
ZYMEM<CR>

                        ZyMEM
                    ZyP Memory Utility
              Version 5.1.3     Release Date: 09/22/84
          A proprietary product of ZyMOS Corporation
      477 North Mathilda, Sunnyvale, California, 94086, USA

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)> ZY5<CR>

Types of memory cells:

  (1) RAM
  (2) ROM
  (3) PLA

Please select memory type. (1, 2 or 3) (<CR>:Quit)>2<CR>

  (1) 2K x 8 ROM
  (2) 80C49 plot file

Select 1 or 2. (<CR>:Quit)>2<CR>

Convert Intel object file into

  (1) ZyPSIM format
  (2) layout plot file

Please select one. (1 or 2) (<CR>:Quit)>2<CR>

Enter 80C49 plot filename.
(1 alpha. + 3 digits, e.g., Z001) (<CR>:Quit)>J588<CR>

Enter Intel object filename. (<CR>:Quit)>MOVX.80C49<CR>

What is the default data? (e.g., FF) (<CR>:Quit)>00<CR>

Enter reference coordinates.
(e.g., 464.5 -990.5) (<CR>:Quit)>0.0 0.0<CR>
```

NOTE: The reference coordinates are the coordinates of the ROM reference point (which is the lower left corner of the ROM) in the whole circuit.

```
**** The plot data is located in file ZC5J588A

Please select library (Zy4, Zy5, 'user_lib', <CR>=Quit, ?=Help)><CR>
```

ZyPROM USER'S GUIDE

Doc: 20-010-314 Rev: A | Issued: September 1984

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

PAGE	REVISION
Title	Rev: A
Table of Contents	Rev: A
Page 1-1	Rev: A
Page 2-1	Rev: A
Pages 3-1 through 3-6	Rev: A

TABLE OF CONTENTS

1 INTRODUCTION

1.1 SCOPE

This user's guide describes hardware installation and use of the ZyP utility ZyPROM with the Logical Devices PROM PRO-7 EPROM Programmer.

1.2 OVERVIEW

The ZyP Design System supports user design of standard cell networks containing ROM arrays. Examples include ROM cells compiled by ZyMEM and the 80C49 and ZyCOMP-4 microcomputers. ZyPROM provides a link between the ZyP system and ROM data. It supports programming of 2716, 2732, 2764 and 8749 EPROM's as well as reading (up loading) EPROM or ROM data into the ZyP System.

1.3 RELATED DOCUMENTS

To aid in understanding the application of ZyPROM to ZyP circuit design, the following documents are recommended.

20-010-014	ZyP Memory User's Guide
20-010-214	ZyMEM User's Guide
20-010-212	ZyCOMP-4 Assembler Reference Manual
20-010-319	80C49 Macro Assembler User's Guide
20-010-619	ZyCE User's Guide

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, user input is show underlined. For example:

TYPEME

Where illustrations of commands are shown, lower case text is used to indicate additional input. Items shown enclosed in angle brackets < > are user defined. For example:

COMMAND <filename>

Items enclosed in braces { } are optional. For example:

COMMAND <filename> {<option>}

2 HARDWARE REQUIREMENTS

Using the EPROM programming and up loading facilities provided by ZyP, ZyPROM and the PROM PRO-7 requires specific hardware. The three major components are as follows:

* Modem - Standard Bell 103/212 compatible, 300 or 1200 baud, required for remote operation only.

* Terminal - ZyPROM is terminal dependent due to the need to connect the terminal to the PROM PRO-7. The DEC VT125, TeleVideo TVI 950 or compatibles will be required.

* PROM PRO-7 - Logical Devices EPROM programmer.

For remote use, the modem and terminal must both be set to either 300 or 1200 baud. For local operation, the terminal should be set to 300 or 1200 baud.

NOTE: Some systems may handle 2400 baud. Contact your system administrator to determine if you can run at this rate.

CAUTION: WHEN USING A VT125 TERMINAL YOU MUST INSTALL AN RS232 LINE DRIVER BETWEEN THE TERMINAL AND THE PROM PRO-7.

Connect the PROM PRO-7 to the terminal printer port using standard 25 pin D connectors with pins 1, 2, 3 and 7 connected.

Certain hardware configurations may have difficulty running ZyPROM in the up loading mode. Please contact your system administrator if you experience problems. Also, be sure to verify correct up loading, especially if you are transferring a ROM code for artwork generation.

3 OPERATION

3.1 DESCRIPTION

ZyPROM is the ZyP to PROM PRO-7 EPROM programmer utility program. ZyPROM can be operated in either menu or command line modes. It performs three basic functions as follows:

3.1.1 ZyCOMP4 FILE UTILITIES

The ZyCOMP4 macro assembler generates an object file of the form filename.ROM. The filename.ROM file can be used by ZyPSIM as well as ZyPROM. When used by ZyPROM, the filename.ROM file has to be converted to standard Intel hex format. In addition, the ZyCOMP4 emulator board accepts 2716 or 2732 EPROMs. If 2716 EPROMs are used, ZyPROM will generate high and low EPROM files.

3.1.2 DOWN LOADING

ZyPROM will down load files of the form filename.HEX or filename.EPM from the ZyP system to the PROM PRO-7. Filename.HEX files are in Intel standard hex format and can be generated in three ways:

1. 80C49 ASM49 Assembler Object File
2. Manually
3. Down Loading filename.EPM Files

The first two ways of generating filename.HEX files are straight forward but the third requires further explanation. Files of the form .EPM are actually only intermediate files. They represent converted ZyCOMP4 assembler .ROM files but are not yet in Intel standard hex format. This intermediate form is a convenience to the ZyCOMP4 emulator board. EPM files can either be converted into high and low ROM data files or down loaded. However, when a .EPM file is down loaded, it is first automatically converted to Intel standard hex format and stored in a file of the form filename.HEX. This file is available directly as a .HEX file for future application.

3.1.3 UP LOADING

ZyPROM will up load (read) EPROM data from the PROM PRO-7 to the ZyP system. A file of EPROM data in standard Intel hex format will be created with the form filename.HEX.

3.2 FILES

3.2.1 INPUT FILES

ZyPROM requires input files as follows:

filename.ROM - Generated by ZyCOMP4 assembler

filename.HEX - Down loaded file, generated by 80C49 ASM49 assembler, manually, or from .EPM files. Contains EPROM data in Intel standard hex format.

filename.EPM - Down loaded file, generated by ZyPROM from filename.ROM. Also used by ZyPROM to generate high and low 2K EPROM files for the ZyCOMP4 emulator board.

3.2.2 OUTPUT FILES

ZyPROM creates four output files as follows:

filename.HEX - Created when up loading to ZyP from the PROM PRO-7 or when down loading a .EPM file. This file contains EPROM data in standard Intel hex format.

filename.EPM - Created when converting filename.ROM file to intermediate format for down loading.

HIfilename.EPM - Created when converting filename.EPM to two 2K word data files. This is the high data file.

LOWfilename.EPM - Created when converting filename.EPM to two 2K word data files. This is the low data file.

3.3 MENU MODE

ZyPROM is started in menu mode by typing ZyPROM. The program displays an option menu as follows:

1. Convert .ROM files to .EPM files
2. Convert .EPM files to 2K format
3. Transfer files to PROM PRO-7
4. Transfer data from PROM PRO-7
5. End

Enter the number of the option desired. ZyPROM responds with a prompt for file name. Enter the file name without extension. ZyPROM will look for or create files with extensions, as follows:

OPTION	FILENAME
1	filename.ROM
2	filename.EPM
3	filename.HEX HIfilename.EPM LOWfilename.EPM filename.EPM
4	filename.HEX

At the completion of options 1 thru 4, ZyPROM will report status and return to the main menu. To exit ZyPROM, select option 5.

The following are example terminal sessions running ZyPROM in the menu mode.

ZYPROM <CR>

```
                        ZYPROM
                  PROM PRO-7 UTILITIES
            Version 5.1.0     Release Date: 08/20/84
          A proprietary product of ZyMOS Corporation
      477 North Mathilda, Sunnyvale, California, 94086, USA


1) CONVERT .ROM FILES TO .EPM FILES
2) CONVERT .EPM FILES TO 2K FORMAT
3) TRANSFER FILES TO PROM PRO-7
4) TRANSFER DATA FROM PROM PRO-7
5) END

INPUT OPTION (1-5) ?1<CR>
Enter file prefix (everything up to the period).  (<CR>:Quit)> BIRD<CR>
CONVERTING TO .EPM
```

```
1) CONVERT .ROM FILES TO .EPM FILES
2) CONVERT .EPM FILES TO 2K FORMAT
3) TRANSFER FILES TO PROM PRO-7
4) TRANSFER DATA FROM PROM PRO-7
5) END


INPUT OPTION (1-5) ?2<CR>
Enter file prefix (everything up to the period).  (<CR>:Quit)> BIRD<CR>
CONVERTING INTO 2K FILES


1) CONVERT .ROM FILES TO .EPM FILES
2) CONVERT .EPM FILES TO 2K FORMAT
3) TRANSFER FILES TO PROM PRO-7
4) TRANSFER DATA FROM PROM PRO-7
5) END


INPUT OPTION (1-5) ?3<CR>
Enter file prefix (everything up to the period).  (<CR>:Quit)> HIBIRD<CR>
DOWN LOADING

FINISH WITH CHECKSUM ADD

I
I*
I*EOJ*
T0:0307FA00000000FC
:00000001FF
*
*EOJ*
*

*EOJ*
*


1) CONVERT .ROM FILES TO .EPM FILES
2) CONVERT .EPM FILES TO 2K FORMAT
3) TRANSFER FILES TO PROM PRO-7
4) TRANSFER DATA FROM PROM PRO-7
5) END
```

```
INPUT OPTION (1-5) ?3<CR>
Enter file prefix (everything up to the period). (<CR>:Quit)> LOWBIRD<CR>
DOWN LOADING
FINISH WITH CHECKSUM ADD

I
I*
I*EOJ*
T0:2000000001060000080300000C080300080F03080F04080F05010704080303080701040
B1D
:2000200004030807010408050308070100080808F0004060C0900030F0104000C000509001
3
:2000400000080300000C080F00080F01080F02010708030000E080F03080F04080F0509C
6
:200060000F00040B0303080F05040B0403080F05040B0503080F05080B0204070C0902089
C
:120080000F02030F0B0800050105000005000000000226
:00000001FF
*
*EOJ*
*

*EOJ*
*



1) CONVERT .ROM FILES TO .EPM FILES
2) CONVERT .EPM FILES TO 2K FORMAT
3) TRANSFER FILES TO PROM PRO-7
4) TRANSFER DATA FROM PROM PRO-7
5) END

INPUT OPTION (1-5) ?5<CR>
```

3.4 COMMAND LINE MODE

ZyPROM can operate in command line mode by typing a command of the form:

ZyPROM <filename> {<option ... -option>}

Filename is the appropriate file without extension and -option is as follows:

OPTION	DESCRIPTION
-CONVERT	Convert filename.ROM file to filename.EPM
-HALF	Split filename.EPM into two 2K files, HI filename.EPM and LOW filename.EPM
-DOWN	Down load filename.HEX or filename.EPM to PROM PRO-7
-UP	Up Load EPROM from PROM PRO-7 to ZyP. Creates file of the form filename.HEX
-END	Terminates ZyPROM
-ALL	Causes first three options to be executed and terminates ZyPROM.

The following is an example terminal session running ZyPROM:

```
ZYPROM BIRD -CONVERT -HALF<CR>


                        ZYPROM
                  PROM PRO-7 UTILITIES
            Version 5.1.0     Release Date:  08/20/84
            A proprietary product of ZyMOS Corporation
        477 North Mathilda, Sunnyvale, California, 94086, USA


CONVERTING TO .EPM
CONVERTING INTO 2K FILES
```

ZyCE USER'S GUIDE

Doc: 20-010-619 Rev: A | Issued: September 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

TABLE OF CONTENTS

1 INTRODUCTION

1.1 SCOPE

This user's guide describes hardware installation and use of the ZyP utility ZyCE with the Applied Microsystem EM-149 Emulator.

1.2 OVERVIEW

The ZyP Design System supports user design of 80C49 based standard cell circuits. ZyCE is a specific ZyP utility which enables the user to communicate with the Applied Microsystems EM-149 Emulator. ZyCE in combination with other 80C49 utilities, such as the ASM49 assembler and SIM49 simulator, permit design verification, simulation and emulation of 80C49 standard cell designs.

ZyCE specifically addresses linking the ZyP System with 80C49 emulation. It supports down loading and up loading of 80C49 object files in standard Intel format and permits up loading and display of trace buffers and single line disassemblies.

1.3 RELATED DOCUMENTS

ZyCE is heavily linked to 80C49 based designs. To aid in understanding the application of ZyCE to the 80C49 and the ZyP system, the following documents are recommended.

20-010-019	80C49 User's Guide
20-010-5UCA	80C49 Data Sheet
20-010-319	80C49 Macro Assembler User's Guide
20-010-419	80C49 Simulator User's Guide

Detailed understanding of the EM-149 emulator is required. Refer to the EM-149 Diagnostic Emulator User's Guide for specific EM-149 operating information.

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, user input is show underlined. For example:

TYPEME

Where illustrations of commands are shown, lower case text is used to indicate additional input. Items shown enclosed in angle brackets < > are user defined. For example:

COMMAND <filename>

Items enclosed in braces { } are optional. For example:

COMMAND <filename> {<-option>}

2 HARDWARE REQUIREMENTS

Using the emulation facilities provided by ZyP, ZyCE and the EM-149 requires specific hardware. The three major components are as follows:

* MODEM

  Standard Bell 103/212 compatible, 300 or 1200 baud, required for remote operation only.

* TERMINAL

  ZyCE is terminal dependent due to the need to connect the terminal to the EM-149. The DEC VT125 or TeleVideo TVI950, or compatibles, will be required.

* EM-149

  Applied Microsystems 80C49 Emulator.

For remote use, the modem and terminal must both be set to either 300 or 1200 baud. For local operation, the terminal can be set to 300, 1200 or 9600 baud.

Connect the EM-149 to the terminal printer port using standard 25 pin D connectors with pins 1, 2, 3 and 7 connected.

Certain hardware configurations may have difficulty running ZyCE in one of the up loading modes. Please contact your system administrator if you experience problems. Also, be sure to verify correct up loading, especially if you are transferring a ROM code for masking.

If used with VT125 terminal, you must install a RS232 line driver between the terminal and the emulator.

3 OPERATION

3.1 DESCRIPTION

ZyCE is the ZyP to EM-149 80C49 emulator utility program. ZyCE can be operated in either the menu or command line modes. It performs three basic file transfer functions as follows:

1. Transfer of a standard Intel format ROM data file from the ZyP System to the EM-149. ROM data must reside in a file of the form filename.HEX. This transfer may be referred to as down loading.

2. Transfer of a standard Intel format ROM data file from the EM-149 to the ZyP System. A file of the form filename.HEX is created. This transfer may be referred to as up loading.

3. Transfer of the EM-149 trace buffer (single line disassembly) from the EM-149 to the ZyP System. A file of the form filename.DMP is created and the data will also be displayed at the terminal.

3.2 FILES

3.2.1 INPUT FILES

ZyCE requires an input file of the form filename.HEX when down loading. This file is created by the ASM49 assembler and can also be created manually. It must be in Intel standard hex format.

3.2.2 OUTPUT FILES

ZyCE creates two output files, depending on the operation, as follows:

filename.HEX - Created when up loading to ZyP from the EM-149. Contains ROM data in standard Intel format.

filename.DMP - Created when up loading the EM-149 trace buffer.

3.3 MENU MODE

ZyCE is started in menu mode by typing ZyCE. The program displays an option menu as follows:

```
1)  DOWN LOAD TO EMULATOR
2)  UP LOAD INTEL FORMAT
3)  UP LOAD DISASSEMBLY
4)  END
```

Enter the number of the option desired. ZyCE responds with a prompt for filename. If option 1 is selected, a file of the form filename.HEX is required. Enter the filename, excluding the extension. For options 2 and 3, ZyCE will create files of form filename.HEX and filename.DMP, respectively.

After completing menu options 1 thru 3, ZyCE returns to the option menu. To exit ZyCE, select option 4.

The following is an example terminal session running ZyCE in the menu mode.

```
ZYCE <CR>

                                ZYCE
                          ZYCE UTILITIES
                    Version 5.1.0     Release Date: 08/20/84
                  A proprietary product of ZyMOS Corporation
            477 North Mathilda Ave., Sunnyvale, California 94086, USA


1)  DOWN LOAD TO EMULATOR
2)  UP LOAD INTEL FORMAT
3)  UP LOAD DISASSEMBLY
4)  END

OPTION (1-4) ? 1 <CR>
FILE NAME ? MICRO <CR>

LOAD REGISTER F WITH 4E AND EXECUTE CODE DB THEN,
ENTER CODE C3 ON EMULATOR AND THEN PRESS RETURN

1)  DOWN LOAD TO EMULATOR
2)  UP LOAD INTEL FORMAT
3)  UP LOAD DISASSEMBLY
4)  END

OPTION (1-4) ? 4 <CR>
```

3.4 COMMAND LINE MODE

ZyCE can be generated in the command line mode by typing a command line of the form,

ZyCE <filename> {<-option -option .. -option>}

Filename is the name of the down load or up load file, excluding extension, and -option is as follows:

OPTION	DESCRIPTION
-HELP	ZyCE Help Message
-UP	Transfer Intel standard format ROM data from the EM-149 to the ZyP System. Data is stored in a file of the form filename.HEX.
-DOWN	Transfer Intel standard format ROM data file from the ZyP System to the EM-149. ROM data must reside in a file of the form filename.HEX.
-PRINT	Transfer single line disassembly to the ZyP System. Data will be displayed at the terminal and stored in a file of the form filename.DMP.
-END	Terminate ZyCE

The following is an example terminal session running ZyCE in the command line mode.

```
ZYCE MICRO -DOWN <CR>

                        ZYCE
                   ZYCE UTILITIES
            Version 5.1.0     Release Date: 08/20/84
          A proprietary product of ZyMOS Corporation
      477 North Mathilda Ave., Sunnyvale, California 94086, USA

LOAD REGISTER F WITH 4E AND EXECUTE CODE DB THEN,
ENTER CODE C3 ON EMULATOR AND THEN PRESS RETURN
```

ZySPICE USER'S GUIDE

Doc: 20-010-008 Rev: E | Issued: August 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

LIST OF EFFECTIVE PAGES

PAGE	REVISION
Title	Rev: E
Table of Contents	Rev: E
Page 1-1	Rev: E
Page 2-1	Rev: E
Pages 3-1 through 3-2	Rev: E
Pages 4-1 through 4-2	Rev: E
Pages 5-1 through 5-6	Rev: E
Pages 6-1 through 6-2	Rev: E
Pages 7-1 through 7-4	Rev: E
Page A-1	Rev: E
Pages B-1 through B-2	Rev: E

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1 INTRODUCTION

1.1 SCOPE

This document describes the use of ZySPICE, a general purpose circuit analysis program. ZySPICE is a ZyMOS modified version of the popular SPICE program which originates from the University of California.

ZySPICE is based on version V2E.3 of SPICE and contains software revisions and enhancements for improved short channel performance. The channel length modulation section of ZySPICE is significantly different from SPICE. Users familiar with SPICE should read section 3 to avoid any possible confusion.

Support for ZySPICE consists of pre-defined MOS models mapped to the ZyMOS process parameters, macro models of ZyP cells at the transistor level and operation on ZyPNET VAX or Prime, or user in house system.

1.2 RELATED DOCUMENTS

It is assumed that prior to using ZySPICE, the user is familiar with the operating system, editor etc., of PRIME or VAX. The appropriate document is shown below together with other relevant ZyP Design System documents.

20-010-001	Introduction to the ZyP system.
20-010-003	ZyPNET Prime User's Guide.
20-010-118	ZyPNET Vax User's Guide.
20-010-208	ZySPICE Reference Manual.

1.3 DOCUMENT CONVENTIONS

In the examples of program execution presented in this document, the following conventions are used.

All user input to the computer is shown underlined. Lower case input is used to indicate user supplied items such as filenames. The remaining text is the computers response.

NOTE: In the examples of execution of command files, the underlined portions are the contents of the command files.

2 ZySPICE CAPABILITIES

2.1 GENERAL

Although SPICE is a general purpose circuit simulator, the support for ZySPICE is oriented towards a specific area: - MOS circuit simulation. Models support both DC and transient analysis for digital cells as well as AC analysis for analog cells.

2.2 CIRCUIT COMPLEXITY

ZySPICE imposes no limitation on the size or complexity of the circuit to be simulated, the only requirement is that all necessary data fit in memory.

2.3 OSCILLATOR CIRCUITS

As a general rule the simulation of oscillator circuits should be performed with extreme care. Circuit simulators occasionally exhibit non-convergence when dealing with feedback circuits. R-C relaxation type oscillators which use threshold detection and delayed feedback, can be simulated with a good measure of success. However, crystal oscillators should never be attempted in a "closed loop" simulation. The extremely high Q of a crystal leads to dynamic voltage swings which cause simulator problems and non-convergence can result.

If circuit verification of a crystal oscillator is required, a recommended approach is to determine the gain/phase response of the amplifier and the attenuation characteristics of the feedback network. This should ensure that oscillation conditions can be met.

2.4 CONVERGENCE

For most types of simulations, no problems due to non-convergence will be encountered. However, there are certain types of circuits which often result in convergence problems. As mentioned in 2.3, closed loop simulations are probably the main offender but certain DC circuits are equally prone to non-convergence. Schmitt triggers, where positive feedback causes fast voltage changes, and comparators with fast voltage changes are among the components that can cause problems.

ZySPICE contains several program control options which allow the user to change the internal convergence criteria. These are documented in the ZySPICE Reference Guide in section 9.6, the .OPTIONS statement.

By changing the values of parameters such as ABSTOL and ITL, the program can often be made to converge when problems occur. This should be considered as a last resort however, since inadvertent use can increase CPU time drastically without any real improvement in performance.

3 PROGRAM MODIFICATIONS

3.1 SUMMARY

This section describes the changes made to SPICE by ZyMOS. They consist of revised equations for saturation voltage and channel length modulation, scaling and sizing options and file inclusion facilities.

The change to the saturation voltage is active only when model parameter VMAX is specified. Only the LEVEL 2 model is changed.

3.2 LAMBDA - CHANNEL LENGTH MODULATION PARAMETER

The channel length modulation parameter used by SPICE was not directly related to a process parameter. It was an empirical value which caused a "spot" value of channel length modulation not related to operating conditions other than VDS. The program was modified to use a process related parameter which is consistent with other circuit simulators, ie ASPEC and ISPICE.

The parameter LAMBDA expected by ZySPICE is as follows:

LAMBDA = {(2 * E Si) / (q * Nsub)}

where E Si = 11.7 * 8.854E-12 (F/M) (Internal to ZySPICE)
q = 1.6E-19 (Coulombs)
Nsub = substrate doping (/M^3)

This is a permanent feature and is not dependent on the VMAX parameter as are the other enhancements described in this section.

3.3 SATURATION VOLTAGE

For short channel length devices, saturation occurs at a lower voltage than given by the SPICE standard VDSAT equation. SPICE contains an equation which modifies VDSAT whenever appropriate parameters are entered. Unfortunately, SPICE used an iterative solution which was unsuccessful.

ZySPICE uses an alternative equation which is a good approximation, based on the same parameters as SPICE. The modified equation used by ZySPICE is:

VDSAT' = VDSAT + VC - (VDSAT^2 + VC^2) ^.5

Where VDSAT' is the modified saturation voltage and VC is given by:

VC = VMAX * (LO / Ueff)

L0 is the channel length, U eff is the effective mobility and VMAX is an empirical factor representing the scattering velocity in meters/second.

ZySPICE will use these equations to modify VDSAT whenever VMAX is entered in the device model parameter list. Results of device measurement versus ZySPICE analysis show this approach is accurate to within several percent.

3.4 CHANNEL LENGTH MODULATION

The SPICE channel length modulation (CLM) equations apply at all values of VDS, even when VDS is less than VDSAT. This causes appreciable errors for short channel devices. The current in the linear region (given by SPICE) is greater than that of actual short channel devices.

ZySPICE uses an alternate channel length modulation equation which is only used when the parameter VMAX is specified. Otherwise, the original SPICE equation is used. The modified CLM model is:

dL/L0 = (LAMBDA / L0) * {(Va^2 + VDS - VDSAT')^.5 - Va)}

where Va = VMAX (LAMBDA / 2 * Ueff)

This only affects the current when VDS is greater than VDSAT'.

3.5 SIZING AND SCALING OPTIONS

Two new options are added which facilitate sizing and scaling of device dimensions. These options are useful if dimensions of all the devices need to be changed by the same amounts. This is useful in situations where masks get shrunk by a certain percentage or device dimensions change due to a change in the process technology.

3.6 FILE INCLUSION AND LIBRARY SEARCH

New statements are added to facilitate and simplify the preparation of the input file. Files can be included using the .INCLUDE statement and user-specified libraries can be searched by using the .USRMOD and .USRSUB statements.

3.7 DIODE SPECIFICATION

The format of the diode specification is changed so that a periphery factor can be specified in addition to an area factor. These two factors are used to compute the zero-bias junction capacitance.

4 DEVICE MODELS

4.1 SUMMARY

Standard SPICE has two modes of model parameter specification, empirical and analytical. In the analytical mode, process parameters are used such as oxide thickness, substrate doping etc. The empirical model uses the characteristics derived from device measurements directly. It is the empirical model parameters that are supported for the ZyMOS processes.

Also, standard SPICE has three types of models which differ in the equations us for device current calculation. The type of model is selected by using LEVEL = n when the device models are specified. The process parameters supported by ZyMOS use the LEVEL 2 model for Zy40000 5u CMOS, and LEVEL 3 for Zy50000 3u CHMOS.

4.2 MODEL FILES

Models are provided for both of the major ZyMOS processes. For each process, there are three sets of parameters representing minimum, typical and maximum process conditions. In total, ten model files are therefore provided.

The minimum process condition models refer to low current transistors or slow speed operation. Maximum process condition models refer to high current transistors or maximum power/fastest speed operation.

The files are identified as follows:

PROCESS	MIN	TYP	MAX
5u Zy40000 CMOS	CSIGMIN	CSIGTYP	CSIGMAX
3u Zy50000 CHMOS	HCSIGMIN	HCSIGTYP	HCSIGMAX

Each file contains models for the appropriate P channel or N channel transistors, together with the diffusion diode models. Appendix B.2 contains a listing of the typical parameter model for the Zy40000 CMOS process. See section 4.4 regarding model file accessing.

4.2.1 DIFFUSION CAPACITANCE

Although ZySPICE allows the transistor element statement to define diffusion areas directly, diode statement entry is supported by the ZyMOS models. The diode model is defined to enable area to be entered in square microns, the correct diffusion capacitance for the applied voltage is then produced.

4.3 MODEL NAMES

In order to define the required model, the name of the model must be used in the circuit description whenever a transistor or diode is used. The full format for model specification is described in section 7 of the ZySPICE Reference Manual.

The names of the models provided by ZyMOS are as follows:

PROCESS gate length	P CH TRANS. < 10u	N CH TRANS. < 10u	P CH TRANS. > 10u	N CH TRANS. > 10u	P DIFF DIODE	N DIFF DIODE
Zy40000 CMOS	PCSG	NCSG	PCSG	NCSG	PJD	NJD
Zy50000 CHMOS	PCSGN	NCSGN	PCSGW	NCSGW	PJD	NJD

In addition to the model name, the correct file (MIN,TYP,MAX) of the required process must be used in creating the ZySPICE input file. The method of model specification is described in section 5.

4.4 MODEL FILE ACCESSING

Model files are protected and cannot be modified other than by ZyMOS authorized personnel. The files allow access for read or copy only.

4.4.1 MODEL FILES

The files described in 4.2 are resident in a directory called MODELS. In order to inspect the contents of a file reference Appendix B.1 for examples on access.

5 ZySPICE FILE GENERATION

5.1 SUMMARY

This section describes the requirements of ZySPICE input files. An example of a simple Zy40000 CMOS inverter is used to illustrate the various points of input generation. This is developed into a complete simulation example. The output from the simulation is contained in section 7.1.

5.2 GENERATING A ZySPICE INPUT FILE

ZySPICE requires that all input data be contained in a single input file. Although this is a usable method, it does not allow the use of global model files in a "safe" manner.

ZyMOS will provide and maintain the process model files described in section 4. These will be updated from time to time as new data becomes available. It is therefore desirable, to have all ZySPICE jobs be able to automatically receive the updated models. The techniques described will ensure that all jobs, whenever they are run, will always contain current device model data.

It is not recommended that a single input file be constructed by loading the model files into your input file directly. If this is done the models will not be updated at run time and may eventually become outdated.

Computer specific example for loading separate files containing the circuit, model and simulation data are shown in Appendix A. Please note the particular manuals for referencing the detailed information concerning these techniques.

5.3 INPUT FILE PARTITIONING

The above requirement means that the input file needs to be generated at run time from the model file and the remainder of the input data. It is more convenient to further split the input file, into a total of three sections: circuit description, device models and simulation input stimuli and control.

Fortunately, there are only two format requirements imposed by ZySPICE which affect the partitioning. These are as follows:

The first statement of an input file MUST be a title line. ZySPICE will print this verbatim on all output listings.

The last line of the input file must be a .END statement.

The circuit description section will therefore have the title as its first line. The input stimuli and control section can contain the .END statement to complete the file. The model parameter section can now be

inserted into the body of the file as the center section. This allows easy changes to the simulation by editing or substitution of files. A change of circuit configuration, device model or simulation conditions is readily achievable with a minimum of effort.

The method of combining the three files and submitting the job to ZySPICE is described in Appendix A. Creation of the required files will now be described.

5.4 CIRCUIT DESCRIPTION

The first section of ZySPICE input is the description of the circuit elements and their interconnections. In order to describe the circuit, node numbers must be assigned to each node in the circuit. Node 0 must be used for the common ground connection and it is recommended that node 1 be the main power node. Nodes may be assigned in any order. Section 4 of the ZySPICE Reference Manual deals with input data and circuit elements in detail.

Figure 5.1 shows the ZySPICE representation of a CMOS inverter circuit.

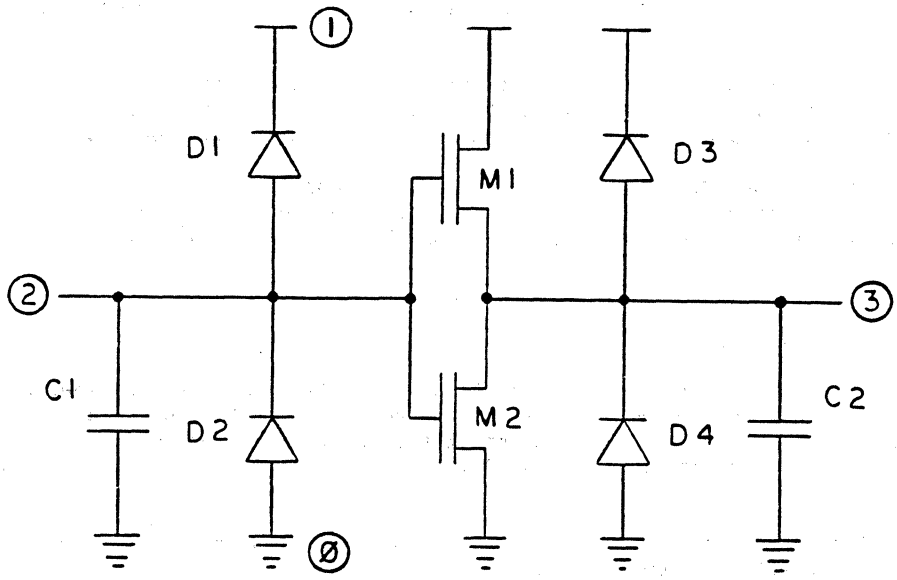


FIGURE 5.1 Zy40000 CMOS INVERTER ZySPICE EXAMPLE

The input statements needed to fully define the circuit are as follows:

```
Zy40000 CMOS INVERTER ZySPICE EXAMPLE
M1 3 2 1 1 PCSG W=12U L=6U
M2 3 2 0 0 NCSG W=12U L=6U
D1 2 1 PJD 210
D2 0 2 NJD 474
D3 3 1 PJD 552
D4 0 3 NJD 604
C1 2 0 .05PF
C2 3 0 .05PF
```

FIGURE 5.2 ZySPICE CIRCUIT DESCRIPTION FOR EXAMPLE CIRCUIT OF FIGURE 5.1

Each element in the circuit is defined with a keyletter followed by a unique identifier. Sequential numbers or node numbers are a common means of identification. This example uses sequential numbers.

Key letters, element identification and syntax of data entry are described in the ZySPICE Reference Manual in section 5, CIRCUIT DESCRIPTION and section 6, under ELEMENTS. The keyletters used in this example are M (transistor), D (diode) and C (capacitor).

The keyletter and identifier are followed by the node connections for the element. This is followed by data appropriate to the element being defined.

5.4.1 TRANSISTORS

The sequence of node assignment is DRAIN GATE SOURCE BULK. In most cases, the BULK node will be connected to VDD for a P channel device and to VSS for an N channel device. Exceptions to this will be analog circuits where special bias conditions exist.

The next item in the transistor element statement is the model name. This will be the appropriate name from those described in section 4.3. The final item requiring specification is the device size. Both width and length are required for each element statement. The default units are meters (careful!), the normal scale factor is U (10-6) which allows entry in microns. Scale factors are described in section 4 of the ZySPICE Reference Manual.

In Figure 5.2, M1 is therefore:

DRAIN = NODE 3, GATE = NODE 2, SOURCE = NODE 1, BULK = NODE 1.

The model is PCSG (P channel Zy40000 CMOS), sizes are width 12 microns, length 6 microns.

1

5.4.2 DIODES

The order of node assignment is ANODE CATHODE. For diodes used as diffusion capacitance, as per this example, the diodes are reverse biased. This means that the CATHODE node should always be more positive than the ANODE.

The model name is the next item, this is either PJD or NJD dependent on which type of diffusion is represented. The final item is the diffusion area in In Figure 5.2, D1 is a P diode, ANODE = NODE 2, CATHODE = NODE 1 (most positive). Model name is PJD and the number 210 represents a 14u x 15u area of P diffusion.

5.4.3 CAPACITORS

The node assignment for capacitors is not critical since they are not voltage dependent. The value of capacitance directly follows the node numbers. In Figure 5.2, C1 and C2 represent the capacitance of the metal lines associated with the input and output nodes.

5.5 DEVICE MODELS

If you are going to run a simulation using a standard ZyMOS process, creation of a device model file is not required. The model file will be automatically inserted into your ZySPICE input file at run time if you use a job control file as described in appendix A.1 and A.2.

However, should you require a special model or wish to generate a model with different parameters, a new file is required. Use the ZyMOS model file as a guide and refer to the model section of the ZySPICE Reference Manual in order to create your own model file.

The file should be given a unique name which is then used instead of a standard ZyMOS model file name when creating your job control file.

5.6 SIMULATION DATA

The simulation data provides ZySPICE with information such as type of analysis to be performed, the type of output required and how long the simulation should run. This is the simulation control data.

The input data to perform the simulation, input signals, i.e. clocks, data etc. to with power and/or bias voltages must be supplied. The simulation data file for the inverter example circuit is shown in Figure 5.3.

```
VIN 2 0 PULSE(0 5 10N 20N 20N 50N 130N)
VDD 1 0 5
.TRAN 2N 200N
.PLOT TRAN V(2) V(3) (0,5)
.END
```

FIGURE 5.3 SIMULATION DATA FOR CIRCUIT OF FIGURE 5.1

5.6.1 INPUT SIGNALS

Several types of standard input signal generators are provided by ZySPICE, they are documented in section 6.1 (INDEPENDENT SOURCES) of the ZySPICE Reference

The type to be used will depend on the type of analysis to be performed. In the case of the inverter circuit, TRANSIENT analysis will be used. The types of analysis provided by ZySPICE are discussed in section 2 of the Reference Manual.

5.6.1.1 Pulse Source

Of all the sources available, by far the most commonly used in MOS simulation is the PULSE source. Figure 5.4 shows the input signal to the inverter example provided by the PULSE statement in the simulation data file of Figure 5.2.

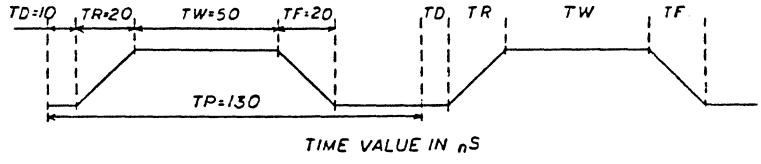


FIGURE 5.4 FORMAT OF PULSE INPUT

The pulse can be specified as a current, keyletter I, or more commonly as a voltage with a keyletter of V. The keyletter is followed by the nodes to which the input is connected, one of which is usually ground. Then follows the keyword PULSE and the details of timing and amplitude of the pulse. The order of pulse data specification is as follows:

V1	Initial pulse amplitude.
V2	Amplitude during pulse width time (TW)
TD	Delay from time zero to start of rise or fall time.
TR	Rise time from V1 to V2
TF	Fall time from V2 to V1
TW	Pulse width (time is width at amplitude V2).
TP	Time from zero at which the sequence is to repeat.

The default units are volts and seconds. In the pulse of Figure 5.3 the pulse name is VIN, connected between node 2 and ground (inverter input). The pulse is delayed 10nS from the simulation start time, rise and fall times are both 20nS. The width is 50nS and the

sequence will repeat after 130nS. The initial value is 0v and the value during the 50ns period is 5 volts.

To complete the input data, the power source for the circuit must be specified. This is defined as VDD (keyletter V) connected between node 1 and node 0. The default units are volts so the example circuit will operate at 5 volts.

5.6.1.2 Piecewise Linear Source

The pulse source although most commonly used, cannot provide a pulse sequence of differing pulse widths, or time between pulses. In order to generate this type of input the piecewise linear source must be used. It is assigned similar to the PULSE source except that the keyword PWL is used. An example usage is:

```
VIN 2 0 PWL(0 0 20N 5 50N 5 80N 0 130N 0 150N 5 230N 5)
```

The data items are pairs of Time/Voltage specifications. The above example would start at 0v, rise to 5v in 20nS, stay at 5v for 30nS, fall to 0v in 30nS etc. In this manner any desired sequence can be constructed.

5.6.2 SIMULATION CONTROL

Control statements begin with a period (.) and indicate to ZySPICE how to perform the simulation. Section 9 of the Reference Manual describes all the various control statements available.

The inverter circuit requires very few in order to produce meaningful data. The first of these is the .TRAN statement. This is used whenever a transient analysis is required. The numbers which follow determine the output printing interval, in this case every 2nS. Next is total time of simulation, in this case 200nS.

An arbitrary limit to the number of output points which can be plotted or printed of 201 exists. This can be changed if required by using the .OPTIONS statement and the LIMPTS=X option where X is the new limit. See section 9.6 of the Reference Manual for the .OPTIONS statement syntax. For the example circuit, only 100 points are requested so no increase in the limit is required.

To determine the output data, a .PLOT or .PRINT statement is used dependent on whether tabular or graphical output is needed. The voltage at any node in the circuit may be requested by a list of nodes following the .PLOT or .PRINT statement. In Figure 5.3, nodes 2 and 3 will be plotted. An additional scale specifier may be defined when the .PLOT statement is used. This limits the bounds of the plotted output to be between the values specified. If this scale is not supplied, ZySPICE will choose its own scale which is usually less than useful. Also, different scales are usually chosen for each output, so specifying your own scale (VSS to VDD) is good practice.

6 ZyP CELL TRANSISTOR LEVEL MODELS

ZySPICE has a subcircuit facility which allows creation of macros for circuits which will be repeatedly used. Section 8 of the ZySPICE Reference Manual describes the subcircuit feature.

ZyMOS has used this feature to create transistor level models for cells in the ZyP Design System libraries. The models are defined using standard ZySPICE elements and are enclosed in .SUBCIRCUIT and .ENDS statements in order to create the macro.

An example of a macro is shown below.

The following listing shows the definition of a macro (subcircuit) for the cell 4GAA1. Note that the comment section at the top of the listing defines the order of node assignments when the macro is used.

```
*
* ZYP Zy40000 CMOS LIBRARY CELL 4-GAA-1 (2 INPUT NAND)
*
*        VDD=1   IN1=2   IN2=3   OUT=4

.SUBCKT GAA1 1 2 3 4
M1 4 3 1 1 PCSG W=79U L=5U
r2 4 2 1 1 PCSG W=79U L=5U
M3 4 3 5 0 NCSG W=77U L=5U
M4 5 2 0 0 NCSG W=77U L=5U
*
D1 4 1 PJD 1786
*
D2 0 4 NJD 1033
D3 0 5 NJD 563
*
*
C1 2 0 .017PF
C2 3 0 .019PF
C3 4 0 .028PF
C4 5 0 .002PF
*
.ENDS
```

EXAMPLE USAGE

To use the macro of this 2 INPUT NAND gate, the following statement would be used:

```
X1 1 6 7 9 GAA1
```

The keyletter X, is used to signify that a macro expansion (use) is being defined. After the keyletter is a macro identifier, in this case 1. Then follow the node assignments and the macro name. The order of nodes in this example is:

1 = VDD, 6 = IN1, 7 = IN2 and 9 = OUT.

Multiple uses of the GAA1 cell can now be made by referencing the GAA1 macr. This is achieved by using several X statements, each one with its own unique identifier eg. X1, X2 etc. In this way, complete circuits can be defined and simulated using the library cell macros.

In order to use the macro, the definition must be present in your ZySPICE input file. The following sections describe the procedures necessary to achieve this.

6.1 MACRO ACCESS

The macros are contained in the following directories.

ZY4LIB	Zy40000 CMOS
ZY5LIB	Zy50000 CHMOS

Each macro is a file in the appropriate directory called cell_name, where cell_name is the standard ZyP name for the required cell.

The files are protected but may be inspected or copied using the methods described for model files in appendix B.1. To insert the macros into your input file use the appropriate editor.

CAUTION: ZySPICE uses large amounts of CPU time. The simulation of even small networks using the library cell macros can become expensive. This is especially true when complex macros such as flip-flops, counters etc. are used. Using ZySPICE for network simulation purposes should be used if ZyPSIM shows a (possible) critical timing path problem, and for AC analysis of analog elements.

7 OUTPUT LISTING

7.1 OUTPUT LISTING FOR EXAMPLE ZySPICE RUN

The following is a partial listing of the output file, 0.INV.SCT, produced by simulating the example inverter circuit of FIG 5.1. The first section is the input data seen by ZySPICE from the file INV.SCT, which was generated by combining the circuit, model and data files.

```
INPUT FILE IS: INV.SCT
*****08/20/84 ****   ZYMOS ZYSPICE 5.1.1 08/14/84****11:42:35***

ZY40000 CMOS INVERTER ZySPICE EXAMPLE

     INPUT LISTING                    TEMPERATURE =   27.000 DEG C
******************************************************************************


M1 3 2 1 1 PCSG W=12U L=6U
M2 3 2 0 0 NCSG W=12U L=6U
D1 2 1 PJD 210
D2 0 2 NJD 474
D3 3 1 PJD 552
D4 0 3 NJD 604
C1 2 0 .05PF
C2 3 0 .05PF
*
*SILICON GATE CMOS PROCESS (ISC1)- DEVICE MODELS WITH SLOW PARAMETERS
*
*P CHANNEL
*
.MODEL PCSG PMOS(LEVEL=2 KP=7.2U VTO=-1 GAMMA=.9 LAMBDA=.9U
+XJ=1E-6 LD=.5 PHI=.6 UCRIT=.9E6 TOX=.8633E-7)
*
*N CHANNEL
*
.MODEL NCSG NMOS(LEVEL=2 KP=22.5U VTO=1 GAMMA=1.9 LAMBDA=.28U
+XJ=1E-6 LD=.4 PHI=.6 UCRIT=.9E6 TOX=.8633E-7)
*
*BIPOLAR MODEL *** (NOTE: THESE ARE TYPICAL PARAMETERS) ***
*
.MODEL QMOD NPN (BF=83 BR=.1 IS=1.4E-14 RB=2.5K RC=2K RE=10 VA=70V
+TF=5NS TR=15NS CJE=3.9E-14 PE=0.7 ME=.5 CJC=1E-13 PC=.7 MC=.5 PT=3)
*
*P DIODE
*
.MODEL PJD D(PB=.7 CJO=1.3E-16 IS=6E-18)
*
*N DIODE
*
.MODEL NJD D(PB=.7 CJO=2.8E-16 IS=2E-17)
*
VIN 2 0 PULSE 0 5 10N 20N 20N 50N 130N
```

```
    VDD 1 0 5
    *
    .TRAN 2N 200N
    .PLOT TRAN V(2) V(3) (0,5)
    .END

1*****08/20/84 ********ZYMOS   ZYSPICE 5.1.1   08/14/84********17:12:40******

OSILICON GATE CMOS INVERTER EXAMPLE

0     TRANSIENT ANALYSIS                TEMPERATURE =   27.000 DEG C

0********************************************************************************


OLEGEND:

 +: V(2)
 *: V(3)
     TIME       V(2)

X(+*)----------------   0.000D-01 1.250D+00 2.500D+00 3.750D+00 5.000D+00
                        - - - - - - - - - - - - - - - - - - - - -
  0.000D-01     0.000D-01   +          .          .          .          *
  2.000D-09     0.000D-01   +          .          .          .          *
  4.000D-09     0.000D-01   +          .          .          .          *
  6.000D-09     0.000D-01   +          .          .          .          *
  8.000D-09     0.000D-01   +          .          .          .          *
  1.000D-08     0.000D-01   +          .          .          .          *
  1.200D-08     5.000D-01   .   +      .          .          .          *
  1.400D-08     1.000D+00   .       +  .          .          .          *
  1.600D-08     1.500D+00   .          . +        .          .          *
  1.800D-08     2.000D+00   .          .    +     .          .          *
  2.000D-08     2.500D+00   .          .         +          .       *  .
  2.200D-08     3.000D+00   .          .           .  +       .    *     .
  2.400D-08     3.500D+00   .          .           .       +* .          .
  2.600D-08     4.000D+00   .          .           *          .  +       .
  2.800D-08     4.500D+00   .          *           .          .     +    .
  3.000D-08     5.000D+00   . *        .           .          .          +
  3.200D-08     5.000D+00   *          .           .          .          +
  3.400D-08     5.000D+00   *          .           .          .          +
  3.600D-08     5.000D+00   *          .           .          .          +
  3.800D-08     5.000D+00   *          .           .          .          +
  4.000D-08     5.000D+00   *          .           .          .          +
  4.200D-08     5.000D+00   *          .           .          .          +
  4.400D-08     5.000D+00   *          .           .          .          +
  4.600D-08     5.000D+00   *          .           .          .          +
  4.800D-08     5.000D+00   *          .           .          .          +
  5.000D-08     5.000D+00   *          .           .          .          +
  5.200D-08     5.000D+00   *          .           .          .          +
  5.400D-08     5.000D+00   *          .           .          .          +
  5.600D-08     5.000D+00   *          .           .          .          +
  5.800D-08     5.000D+00   *          .           .          .          +
```

6.000D-08	5.000D+00	* . . . +
6.200D-08	5.000D+00	* . . . +
6.400D-08	5.000D+00	* . . . +
6.600D-08	5.000D+00	* . . . +
6.800D-08	5.000D+00	* . . . +
7.000D-08	5.000D+00	* . . . +
7.200D-08	5.000D+00	* . . . +
7.400D-08	5.000D+00	* . . . +
7.600D-08	5.000D+00	* . . . +
7.800D-08	5.000D+00	* . . . +
8.000D-08	5.000D+00	* . . . +
8.200D-08	4.500D+00	* . . . + .
8.400D-08	4.000D+00	* . . . + .
8.600D-08	3.500D+00	* . . + . .
8.800D-08	3.000D+00	* . . + . .
9.000D-08	2.500D+00	* . + . .
9.200D-08	2.000D+00	. * . + . .
9.400D-08	1.500D+00	. * . + . .
9.600D-08	1.000D+00	. X . . .
9.800D-08	5.000D-01	. + . * . .
1.000D-07	-2.118D-13	+ . . * . .
1.020D-07	0.000D-01	+ . . . * .
1.040D-07	0.000D-01	+ . . . . *
1.060D-07	0.000D-01	+ . . . . *.
1.080D-07	0.000D-01	+ . . . . *
1.100D-07	0.000D-01	+ . . . . *
1.120D-07	0.000D-01	+ . . . . *
1.140D-07	0.000D-01	+ . . . . *
1.160D-07	0.000D-01	+ . . . . *
1.180D-07	0.000D-01	+ . . . . *
1.200D-07	0.000D-01	+ . . . . *
1.220D-07	0.000D-01	+ . . . . *
1.240D-07	0.000D-01	+ . . . . *
1.260D-07	0.000D-01	+ . . . . *
1.280D-07	0.000D-01	+ . . . . *
1.300D-07	0.000D-01	+ . . . . *
1.320D-07	0.000D-01	+ . . . . *
1.340D-07	0.000D-01	+ . . . . *
1.360D-07	0.000D-01	+ . . . . *
1.380D-07	0.000D-01	+ . . . . *
1.400D-07	0.000D-01	+ . . . . *
1.420D-07	5.000D-01	. + . . . *
1.440D-07	1.000D+00	. + . . . *
1.460D-07	1.500D+00	. . + . . *
1.480D-07	2.000D+00	. . + . . *
1.500D-07	2.500D+00	. . + . . *.
1.520D-07	3.000D+00	. . . + . * .
1.540D-07	3.500D+00	. . . +*. .
1.560D-07	4.000D+00	. . *. . + .
1.580D-07	4.500D+00	. *. . . + .
1.600D-07	5.000D+00	. * . . . +
1.620D-07	5.000D+00	* . . . +
1.640D-07	5.000D+00	* . . . +
1.660D-07	5.000D+00	* . . . +

```
 1.680D-07    5.000D+00  *          .          .          .          +
 1.700D-07    5.000D+00  *          .          .          .          +
 1.720D-07    5.000D+00  *          .          .          .          +
 1.740D-07    5.000D+00  *          .          .          .          +
 1.760D-07    5.000D+00  *          .          .          .          +
 1.780D-07    5.000D+00  *          .          .          .          +
 1.800D-07    5.000D+00  *          .          .          .          +
 1.820D-07    5.000D+00  *          .          .          .          +
 1.840D-07    5.000D+00  *          .          .          .          +
 1.860D-07    5.000D+00  *          .          .          .          +
 1.880D-07    5.000D+00  *          .          .          .          +
 1.900D-07    5.000D+00  *          .          .          .          +
 1.920D-07    5.000D+00  *          .          .          .          +
 1.940D-07    5.000D+00  *          .          .          .          +
 1.960D-07    5.000D+00  *          .          .          .          +
 1.980D-07    5.000D+00  *          .          .          .          +
 2.000D-07    5.000D+00  *          .          .          .          +
                         - - - - - - - - - - - - - - - - - - - - - -

1          JOB CONCLUDED
```

APPENDIX A

A.1 PRIME ZySPICE INPUT FILE

The circuit and data files generated, together with the required model file must be combined to make a single input file suitable for ZySPICE. A PRIME utility program called CONCAT is used for this purpose.

CONCAT is described fully in the PRIMOS COMMANDS REFERENCE GUIDE. ZyP users will find this in volume 2 of document 20-010-103, the ZyPNET PRIME User's Manual.

The following command sequence is used and will become part of the final job control file.

```
CONCAT zyspice_filename.SCT -COMMAND -OVER -NHE
INS circuit_filename
INS MODELS>model_filename
INS simulation_data_filename
QUIT
```

The program is invoked by the line containing CONCAT. This is immediately followed by the name which you wish to give to the combined ZySPICE input file. Several options are then used.

-COMMAND indicates that the command mode will be used with the file data to follow.

-OVER will overwrite (delete) any existing file named zyspice_filename.

-NHE supresses headers which would otherwise be inserted between the files being combined.

The filenames are inserted using the INS command, one per line. This will allow easy changes when CONCAT is used in a job control file. The order of insertion is important and must be as above, ie: circuit,models,data. ZYSPICE can now be run on the new concatenated file.

A.2 VAX ZySPICE INPUT FILE

The VAX system allows concatenation of the circuit, models and data files using the COPY command with the CONCATENATE option. The COPY command and options are fully described in the VAX/VMS COMMAND LANGUAGE USER'S GUIDE - PART 2. Please reference document 20-010-118.

The following command string is used and will become part of the final job control file.

```
COPY/CONC circuit_filename,model_filename,simulation_data_filename -
 zyspice_filename.SCT
```

This /CONC option will combine the separate files into the one file called zyspice_filename.SCT and ZYSPICE can be run on the concatenated file.

APPENDIX B

B.1 MODEL FILE ACCESSING

To view the models using the PRIME system just type the following:

SLIST ZYP>ZYSPICE>MODELS>modelname

and to copy this model to the resident directory using PRIME just type:

COPY ZYP>ZYSPICE>MODELS>modelname

Where modelname is the filename of the required model. For example, typing SLIST ZYP>ZYSPICE>MODELS>CSIGTYP would produce a listing much like the one shown in Appendix B.2.

To access the macros within the particular directories just type:

ATTACH ZYP>ZYSPICE>ZY4LIB or ZY5LIB

To view the models using the VAX system just type the following:

TYPE SYS$ZYPROOT:[ZYSPICE]modelname.

and to copy the contents of modelname into resident directory just type:

COPY SYS$ZYPROOT:[ZYSPICE]modelname.

To access the macros within the particular directories just type:

SET DEF SYS$ZYPROOT:[ZYSPICE.ZY4LIB or ZY5LIB]

B.2 Zy40000 CMOS TYPICAL PROCESS MODELS

The following listing shows the contents of a MODEL file, in this case the typical parameters for the Zy40000 CMOS process.

```
*
*Zy40000 CMOS PROCESS - DEVICE MODELS WITH TYPICAL PARAMETERS
*
*P CHANNEL
*
.MODEL PCSG PMOS(LEVEL=2 KP=8.0U VTO=-.7 GAMMA=.75 LAMBDA=1.02U
+XJ=1E-6 LD=0.5 PHI=.6 UCRIT=.9E6 TOX=.7938E-7)
*
*N CHANNEL
*
.MODEL NCSG NMOS(LEVEL=2 KP=25.0U VTO=.75 GAMMA=1.6 LAMBDA=.33U
+XJ=1E-6 LD=0.4 PHI=.6 UCRIT=.9E6 TOX=.7938E-7)
*
*BIPOLAR MODEL
*
.MODEL QMOD NPN (BF=83 BR=.1 IS=1.4E-14 RB=2.5K RC=2K RE=10 VA=70V
+TF=5NS TR=15NS CJE=3.9E-14 PE=0.7 ME=.5 CJC=1E-13 PC=.7 MC=.5 PT=3)
```

```
*
*P DIODE
*
.MODEL PJD D(PB=.7 CJO=1.15E-16 IS=3E-18)
*
*N DIODE
*
.MODEL NJD D(PB=.7 CJO=2.75E-16 IS=1E-17)
```

1

ZySPICE REFERENCE MANUAL

Doc: 20-010-208 Rev: B | Issued: September 1984

ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

LIST OF EFFECTIVE PAGES

PAGE	REVISION
Title	Rev: B
Table of Contents	Rev: B
Page 1-1	Rev: B
Pages 2-1 through 2-3	Rev: B
Page 3-1	Rev: B
Page 4-1	Rev: B
Page 5-1	Rev: B
Pages 6-1 through 6-8	Rev: B
Pages 7-1 through 7-6	Rev: B
Page 8-1	Rev: B
Pages 9-1 through 9-12	Rev: B
Pages A-1 through A-7	Rev: B

TABLE OF CONTENTS

1

1 INTRODUCTION

1.1 SCOPE

ZySPICE is a ZyMOS modified version of the University of California SPICE general-purpose circuit simulation program for nonlinear DC, nonlinear transient, and linear AC analyses. Circuits may contain resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four types of dependent sources, transmission lines, and the four most common semiconductor devices: diodes, BJTs, JFETs, and MOSFETs.

ZySPICE has built-in models for the semiconductor devices, and the user need specify only the pertinent model parameter values. The model for the BJT is based on the integral charge model of Gummel and Poon; however if the Gummel-Poon parameters are not specified, the model reduces to the simpler Ebers-Moll model. In either case, charge storage effects, ohmic resistances, and a current-dependent output conductance may be included. The diode model can be used for either Junction Diodes or Schottky Barrier Diodes. The JFET model is based on the FET model of Shichman and Hodges. The model for the MOSFET is based on the Frohman-Grove model; however, channel-length modulation, sub-threshold conduction, and some short-channel effects are included. A simpler MOSFET model is also available which uses a square-law I-V characteristic.

New program enhancements include file management capability, enhanced MOS models, and scaling and sizing options. The level-2 model has been modified with a process dependent channel-length modulation parameter, a more accurate modulation equation, and an enhanced saturation voltage model for short-channel transistors. The level-3, El-Mansy model, was enhanced for better accuracy and convergence characteristics.

1.2 RELATED DOCUMENTS

ZySPICE is supported with ZyMOS process models. Relevant ZyP Design System documents are as follows.

20-010-001	Introduction to the ZyP System
20-010-003	ZyPNET Prime User's Guide
20-010-008	ZySPICE User's Guide
20-010-118	ZyPNET VAX User's Guide

1

2 TYPES OF ANALYSIS

2.1 DC ANALYSIS

The DC analysis portion of ZySPICE determines the DC operating point of the circuit with inductors shorted and capacitors opened. A DC analysis is automatically performed prior to a transient analysis to determine the transient initial conditions, and prior to an AC small-signal analysis to determine the linearized, small-signal models for non-linear devices. If requested, the DC small-signal value of a transfer function (ratio of output variable to input source) input resistance, and output resistance will also be computed as a part of the DC solution. The DC analysis can also be used to generate DC transfer curves: a specified independent voltage or current source is stepped over a user-specified range and the DC output variables are stored for each sequential source value. If requested, ZySPICE also will determine the DC small-signal sensitivities of specified output variables with respect to circuit parameters. The DC analysis options are specified in the .DC, .TF, .OP, and .SENS control statements.

2.2 AC SMALL-SIGNAL ANALYSIS

The AC small-signal portion of ZySPICE computes the AC output variables as a function of frequency. The program first computes the DC operating point of the circuit and determines linearized, small-signal models for all of the nonlinear devices in the circuit. The resultant linear circuit is then analyzed over a user-specified range of frequencies. The desired output of an AC small-signal analysis is usually a transfer function (voltage gain, transimpedance, etc.). If the circuit has only one AC input, it is convenient to set that input to unity and zero phase, so that output variables have the same value as the transfer function of the output variable with respect to the input.

The generation of white noise by resistors and semiconductor devices can also be simulated with the AC small-signal portion of ZySPICE. Equivalent noise source values are determined automatically from the small-signal operating point of the circuit, and the contribution of each noise source is added at a given summing point.

The total output noise level and the equivalent input noise level are determined at each frequency point. The output and input noise levels are normalized with respect to the square root of the noise bandwidth and have the units volts/RT HZ or AMPS/RT HZ. The output noise and equivalent input noise can be printed or plotted in the same fashion as other output variables. No additional input data is necessary for this analysis.

Flicker noise sources can be simulated in the noise analysis by including values for the parameters KF and AF in the appropriate device model statements.

The distortion characteristics of a circuit in the small-signal mode can be simulated as a part of the AC small-signal analysis. The analysis is performed assuming that one or two signal frequencies are imposed at the input.

The frequency range and the noise and distortion analysis parameters are specified in the .AC, .NOISE, and .DISTO control statements.

2.3 TRANSIENT ANALYSIS

The transient analysis portion of ZySPICE computes the transient output variables as a function of time over a user-specified time interval. The initial conditions are automatically determined by a DC analysis. All sources which are not time dependent (for example, power supplies) are set to their DC value. For large-signal sinusoidal simulations, a fourier analysis of the output waveform can be specified to obtain the frequency domain fourier coefficients. The transient time interval and the fourier analysis options are specified in the .TRAN and .FOUR control statements.

2.4 ANALYSIS AT DIFFERENT TEMPERATURES

All input data for ZySPICE is assumed to have been measured at 27 degrees C (300 degrees K). The simulation also assumes a nominal temperature of 27 degrees C. The circuit can be simulated at other temperatures by using a .TEMP control statement.

Temperature appears explicitly in the exponential terms of the BJT and diode model equations. In addition, saturation currents have a built-in temperature dependence. The temperature dependence of the saturation current in the BJT models is determined by:

$$IS(TEMP) = 10*(TEMP\hat{}PT)*EXP(-Q*EG/(K*TEMP))$$

Where K is Boltzmans constant, Q is the electronic charge, 10 is a constant, EG is the energy gap which is a model parameter, and PT is the saturation current temperature exponent (also a model parameter, and usually equal to 3). The temperature dependence of the saturation current in the junction diode model is determined by:

$$IS(TEMP) = 10*(TEMP\hat{}(PT/N))*EXP(-Q*EG/(K*N*TEMP))$$

Where N is the emission coefficient, which is a model parameter, and the other symbols have the same meaning as above. Note that for Schottky Barrier Diodes, the value of the saturation current temperature exponent, PT, is usually 2.

Temperature appears explicitly in the value of junction potential PHI, for all the device models. The temperature dependence is determined by:

$$PHI(TEMP) = K*TEMP/Q*LOG(NA*ND/NI(TEMP)^2)$$

Where K is Boltzmans constant, Q is the electronic charge, NA is the acceptor impurity density, ND is the donor impurity density, NI is the intrinsic concentration, and EG is the energy gap.

Temperature appears explicitly in the value of surface mobility, UO, for the MOSFET model. The temperature dependence is determined by:

$$UO(TEMP) = UO(TNOM)/(TEMP/TNOM)^(1.5)$$

Temperature appears explicitly for resistor values according to the following expression:

$$VALUE(TEMP) = VALUE(TNOM)*(1+TC1*(TEMP-TNOM)+TC2*(TEMP-TNOM)^2)$$

Where TEMP is the circuit temperature, TNOM is the nominal temperature, and TC1 and TC2 are the first and second-order temperature coefficients.

3 CONVERGENCE

Both DC and transient solutions are obtained by an iterative process which is terminated when both of the following conditions hold:

1) The nonlinear branch currents converge to within a tolerance of 0.1 percent or 1 picoamp (1.0E-12Amp), whichever is larger.

2) The node voltages converge to within a tolerance of 0.1 percent or 1 microvolt (1.0E-6 Volt), whichever is larger.

Although the algorithm used in ZySPICE has been found to be very reliable, in some cases it will fail to converge to a solution. When this failure occurs, the program will print the node voltages at the last iteration and terminate the job. In such cases, the node voltages that are printed are not necessarily correct or even close to the correct solution.

Failure to converge in the DC analysis is usually due to an error in specifying circuit connections, element values, or model parameter values. Regenerative switching circuits or circuits with positive feedback probably will not converge in the DC analysis unless the "off" option is used for some of the devices in the feedback path.

4 INPUT

4.1 INPUT FILE

ZySPICE requires all input data and commands in a single master file. Refer to the ZySPICE User's Manual for specific information and requirements for the input file.

4.2 INPUT FORMAT

The input format for ZySPICE is of the free format type. Fields in a statement are separated by one or more blanks, a comma, an equal (=) sign, or a left or right parentheses; extra spaces are ignored. A statement can be continued, for one or more lines, by typing a plus (+) sign in column 1. ZySPICE will then continue reading the statement beginning with column 2.

A name field must begin with a letter (A through Z) and cannot contain any delimiters. Only the first eight characters of the name are used.

A number field may be an integer field (12, -44), a floating point field (3.14159), either an integer or floating point number followed by an integer exponent (1E-14, 2.65E3), or either an integer or a floating point number followed by one of the following scale factors:

G = 1E9	MIL = 25.4E-6	N = 1E-9
MEG = 1E6	M = 1E-3	P = 1E-12
K = 1E3	U = 1E-6	F = 1E-15

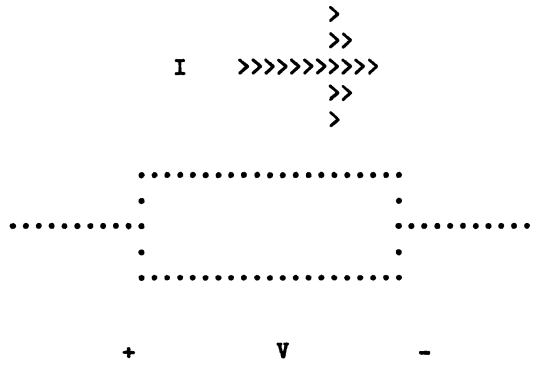
Letters immediately following a number that are not scale factors are ignored, and letters immediately following a scale factor are ignored. Hence, 10, 10V, 10VOLTS, and 10HZ all represent the same number, and M, MA, MSEC, and MHOS all represent the same scale factor. Note that 1000, 1000.0, 1000HZ, 1E3, 1.0E3, 1KHZ, and 1K all represent the same number.

5 CIRCUIT DESCRIPTION

The circuit to be analyzed is described to ZySPICE by a set of element statements, which define the circuit topology and element values, and a set of control statements, which define the model parameters and the run controls. The first statement in the input file must be a title statement and the last statement must be a .END statement. The order of the remaining statements is arbitrary (except, of course, that continuation statements must immediately follow the statement being continued).

Each element in the circuit is specified by an element statement that contains the element name, the circuit nodes to which the element is connected, and the values of the parameters that determine the electrical characteristics of the element. The first letter of the element name specifies the element type. The format for the ZySPICE element types is given in what follows. The strings "XXXXXXX", "YYYYYYY", and "ZZZZZZZ" denote arbitrary alphanumeric strings. For example, a resistor name must begin with the letter R and can contain from one to eight characters. Hence, R, R1, RSE, ROUT, and R3AC2ZY are valid resistor names.

Data fields that are enclosed in square brackets ([ ]) are optional. All indicated punctuation (parentheses, equal signs, etc.) are required. With respect to branch voltages and currents, ZySPICE uniformly uses the associated reference convention indicated in the drawing below:



Nodes must be nonnegative integers but need not be numbered sequentially. The datum (ground) node must be numbered zero. The circuit cannot contain a loop of voltage sources and/or capacitors. Each node in the circuit must have a DC path to ground. Every node must have at least two connections except for transmission line nodes (to permit unterminated transmission lines).

6 ELEMENTS

6.1 RESISTORS

General Form RXXXXXXX N1 N2 VALUE [TC=TC1[,TC2]]

Example RC1 12 17 1K TC=0.001,0.015

N1 and N2 are the two element nodes. VALUE is the resistance (in ohms) and may be positive or negative but not zero. TC1 and TC2 are the (optional) temperature coefficients; if not specified, zero is assumed for both. The value of the resistor as a function of temperature is given by:

VALUE(TEMP) = VALUE(TNOM)*(1+TC1*(TEMP-TNOM)+TC2*(TEMP-TNOM)^2)

6.2 CAPACITORS AND INDUCTORS

General Form CXXXXXXX N+ N- VALUE [IC=V]
LYYYYYYY N+ N- VALUE [IC=I]

Examples CBYP 13 0 1UF
LLINK 42 69 1UH

N+ and N- are the positive and negative element nodes, respectively. VALUE is the element value. For capacitors, VALUE is the capacitance (in farads). For inductors, VALUE is the inductance (in henries).

For the capacitor, the (optional) initial condition is the initial (time-zero) value of capacitor voltage (in volts). For the inductor, the (optional) initial condition is the initial (time-zero) value of inductor current (in amps) that flows from N+, through the inductor, to N-. Note that the initial conditions (if any) apply only if the UIC option is specified in the .TRAN statement.

6.3 COUPLED (MUTUAL) INDUCTORS

General Form KXXXXXXX LYYYYYYY LXXXXXXX VALUE

Examples K43 LAA LBB 0.999

LYYYYYYY and LZZZZZZZ are the names of the two coupled inductors, and VALUE is the coefficient of coupling, K, which must be greater than 0 and less than or equal to 1. The coupled inductors can be entered in either order. In terms of the dot convention, this coupling puts a dot by the first node of each of the coupled inductors.

6.4 TRANSMISSION LINES (LOSSLESS)

General Form TXXXXXXX N1 N2 N3 N4 Z0=VALUE [TD=VALUE]
+ [F=FREQ [NL=NRMLEN]] [IC=V1,I1,V2,I2]

Example T1 1 0 2 0 Z0=50 TD=10NS

N1 and N2 are the nodes at port 1; N3 and N4 are the nodes at port 2. Z0 is the characteristic impedance. The length of the line may be expressed in either of two forms. The transmission delay, TD, may be specified directly (as TD=10NS, for example). Alternatively, a frequency F may be given, together with NL, the normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency F. If a frequency is specified but NL is omitted, 0.25 is assumed (that is, the frequency is assumed to be the quarter wave frequency). Note that although both forms for expressing the line length are indicated as optional, one of the two must be specified.

Note that this element models only one propagating mode. If all four nodes are distinct in the actual circuit, then two modes may be excited. To simulate such a situation, two transmission-line elements are required. (See the example in Appendix A for further clarification).

The (optional) initial condition specification consists of the voltage and current at each of the transmission line ports. Note that the initial conditions (if any) apply only if the UIC option is specified in the .TRAN statement.

6.5 LINEAR DEPENDENT SOURCES

ZySPICE allows circuits to contain linear dependent sources characterized by any of the four equations

I=G*V V=E*V I=F*I V=H*I

Where G, E, F, and H are constants representing transconductance, voltage gain, current gain, and transresistance, respectively.

NOTE: A more complete description of dependent sources as implemented in ZySPICE is given in Appendix B.

6.6 LINEAR VOLTAGE-CONTROLLED CURRENT SOURCES

General Form GXXXXXXX N+ N- NC+ NC- VALUE

Example G1 2 0 5 0 0.1MMHO

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. NC+ and NC- are the positive and negative controlling nodes, respectively. VALUE is the transconductance (in mhos).

6.7 LINEAR VOLTAGE-CONTROLLED VOLTAGE SOURCES

General Form EXXXXXXX N+ N- NC+ NC- VALUE

Example E1 2 3 14 1 2.0

N+ is the positive node, and N- is the negative node. NC+ and NC- are the positive and negative controlling nodes, respectively. VALUE is the voltage gain.

6.8 LINEAR CURRENT-CONTROLLED CURRENT SOURCES

General Form FXXXXXXX N+ N- VNAM VALUE

Example F1 13 5 VSENS 5

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. VNAM is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of VNAM. VALUE is the current gain.

6.9 LINEAR CURRENT-CONTROLLED VOLTAGE SOURCES

General Form HXXXXXXX N+ N- VNAM VALUE

Example HX 5 17 VZ 0.5K

N+ and N- are the positive and negative nodes, respectively. VNAM is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of VNAM. VALUE is the transresistance (in ohms).

6.10 INDEPENDENT SOURCES

General Form
```
VXXXXXXX N+ N- [[DC] DC/TRAN VALUE]
+ [AC [ACMAG [ACPHASE]]]
IYYYYYYY N+ N- [[DC] DC/TRAN VALUE]
+ [AC [ACMAG [ACPHASE]]]
```

Examples
```
VCC 10 0 DC 6
VIN 13 2 0.001 AC 1 SIN(0 1 1MEG)
ISRC 23 21 AC 0.333 45.0 SFFM(0 1 10K 5 1K)
VMEAS 12 9
```

N+ and N- are the positive and negative nodes, respectively. Note that voltage sources need not be grounded. Current is assumed to flow from the positive node, through the source, to the negative node.

DC/TRAN is the DC and transient analysis value of the source. If the source value is zero both for DC and transient analyses, this value may be omitted. If the source value is time-invariant (e.g., a power supply). Then the value may be optionally preceded by the letters DC.

ACMAG is the AC magnitude and ACPHASE is the AC phase. The source is set to this value in the AC analysis. If ACMAG is omitted following the keyword AC, a value of unity is assumed. If ACPHASE is omitted, a value of zero is assumed. If the source is not an AC small-signal input, the keyword AC and the AC values are omitted.

Any independent source can be assigned a time-dependent value for transient analysis. If a source is assigned a time-dependent value, the time-zero value is used for DC analysis. There are five independent source functions: pulse, exponential, sinusoidal, piece-wise linear, and single-frequency FM. If parameters other than source values are omitted or set to zero, the default values shown will be assumed. TSTEP is the printing increment and TSTOP is the final time (see the .TRAN statement for explanation).

6.10.1 PULSE

General Form PULSE(V1 V2 TD TR TF PW P PER)

Example
```
VIN 3 0 PULSE(-1 1 2NS 2NS 2NS 50NS
+ 100NS)
```

PARAMETERS AND DEFAULT VALUES			UNITS
V1	Initial Value	---	Volts or Amps
V2	Pulsed Value	---	Volts or Amps
TD	Delay Time	0.0	Seconds
TR	Rise Time	TSTEP	Seconds
TF	Fall Time	TSTEP	Seconds
PW	Pulse Width	TSTOP	Seconds
PER	Period	TSTOP	Seconds

A single pulse so specified is described by the following table:

TIME	VALUE
0	V1
TD	V1
TD+TR	V2
TD+TR+PW	V2
TD+TR+PW+TF	V1
TSTOP	V1

Intermediate points are determined by linear interpolation.

6.10.2 SINUSOIDAL

General Form SIN(VO VA FREQ TD THETA)

Example VIN 3 0 SIN(0 1 100MEG 1NS 1E10)

PARAMETERS AND DEFAULT VALUES			UNITS
VO	Offset	---	Volts or Amps
VA	Amplitude	---	Volts or Amps
FREQ	Frequency	1/TSTOP	HZ
TD	Delay	0.0	Seconds
THETA	Damping Factor	0.0	1/Seconds

The shape of the waveform is described by the following table:

TIME	VALUE
0 to TD	VO
TD to TSTOP	VO + VA*EXP(-(TIME-TD)*THETA) *SINE(TWOPI*FREQ*(TIME-TD))

6.10.3 EXPONENTIAL

General Form EXP(V1 V2 TD1 TAU1 TD2 TAU2)

Example VIN 3 0 EXP(-4 -1 2NS 30NS 60NS 40NS)

PARAMETERS AND DEFAULT VALUES			UNITS
V1	Initial Value	---	Volts or Amps
V2	Pulsed Value	---	Volts or Amps
TD1	Rise Delay Time	0.0	Seconds
TAU1	Rise Time Constant	TSTEP	Seconds
TD2	Fall Delay Time	TD1+TSTEP	Seconds
TAU2	Fall Time Constant	TSTEP	Seconds

The shape of the waveform is described by the following table:

TIME	VALUE
0 to TD1	V1
TD1 to TD2	V1+(V2-V1)*(1-EXP(-(TIME-TD1)/TAU1))
TD2 to TSTOP	V1+(V2-V1)*(1-EXP(-(TIME-TD1)/TAU1)) +(V1-V2)*(1-EXP(-(TIME-TD2)/TAU2))

6.10.4 PIECE-WISE LINEAR

General Form PWL(T1 V1 [T2 V2 T3 V3 T4 V4 ...])

Example
```
VCLOCK 7 5 PWL (0 -7 10NS -7 11NS -3 17NS
+ -3 18NS -7 50NS -7)
```

PARAMETERS AND DEFAULT VALUES

Each pair of values (TI, VI) specifies that the value of the source is VI (in volts or amps) at Time=TI. The value of the source at intermediate values of time is determined by using linear interpolation on the input values.

6.10.5 SINGLE-FREQUENCY FM

General Form SFFM(VO VA FC MDI FS)

Example V1 12 0 SFFM(0 1M 20K 5 1K)

PARAMETERS AND DEFAULT VALUES			UNITS
VO	Offset	---	Volts or Amps
VA	Amplitude	---	Volts or Amps
FC	Carrier Frequency	1/TSTOP	HZ
MDI	Modulation Index	---	---
FS	Signal Frequency	1/TSTOP	HZ

The shape of the waveform is described by the following equation:

VALUE = VO + VA*SINE((TWOPI*FC*TIME) + MDI*SINE(TWOPI*FS*TIME))

6.11 SEMICONDUCTOR DEVICES

The elements that have been described to this point typically require only a few parameter values to specify completely the electrical characteristics of the element. However, the models for the four semiconductor devices that are included in the ZySPICE program require many parameter values. Moreover, many devices in a circuit often are defined by the same set of device model parameters. For these reasons, a set of device model parameters is defined in a separate .MODEL statement and assigned a unique model name. The device element statements in ZySPICE then reference the model name. This scheme alleviates the need to specify all of the model parameters in each device element statement.

Each device element statement contains the device name, the nodes to which the device is connected, and the device model name. In addition, two optional parameters may be specified for each device: an area factor, and an initial condition.

The area factor determines the number of equivalent parallel devices of a specified model. The affected parameters are marked with an asterisk under the heading "area" in the model descriptions below.

Two different forms of initial conditions may be specified for devices. The first form is included to improve the DC convergence for circuits that contain more than one stable state. If a device is specified off, the DC operating point is determined with the terminal voltages for that device set to zero. After convergence is obtained, the program continues to iterate to obtain the exact value for the terminal voltages. If a circuit has more than one DC stable state, the off option can be used to force the solution to correspond to a desired state. If a device is specified off when in reality the device is conducting, the program will still obtain the correct solution (assuming the solutions converge) but more iterations will be required since the program must independently converge to two separate solutions.

The second form of initial condition specification (using IC=...) is provided to allow the user to bypass the DC operating point calculation normally made before the start of transient analysis. These initial conditions are used by ZySPICE only if the UIC option is given in the .TRAN statement.

6.11.1 JUNCTION DIODES

General Form DXXXXXXX N+ N- MNAME [AREA] [PERIPHERY] [OFF] [IC=VD]

Example
```
DBRIDGE 2 10 DIODE1
DCLMP 3 7 DMOD 3.0 2.0 IC=0.2
```

N+ and N- are the positive and negative nodes, respectively. MNAME is the model name, AREA is the area factor, PERIPHERY is the periphery factor, and OFF indicates an (optional) initial condition on the device for DC analysis. If the area factor is omitted, a value of 1.0 is assumed. If the periphery factor is omitted, a value of 0.0 is assumed. The AREA factor is used to compute the saturation current, by multiplying the value of IS in the diode model by AREA factor. The AREA and PERIPHERY factors are used to compute the zero-bias junction capacitance of the diode. The total zero-bias junction capacitance is computed as AREA * CJO + PERIPHERY * CJSW. CJO and CJSW are specified in the diode model. The (optional) initial condition specification using IC=VD is intended for use with the UIC option in the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point.

6.11.2 BIPOLAR JUNCTION TRANSISTORS

General Form QXXXXXXX NC NB NE MNAME [AREA] [OFF] [IC=VBE,VCE]

Example
```
Q23 10 24 13 QMOD IC=0.6,5.0
```

17

NC, NB, and NE are the collector, base, and emitter nodes, respectively. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) initial condition on the device for the DC analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specifications using IC=VBE,VCE is intended for use with the UIC option in the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point.

6.11.3 JUNCTION FIELD-EFFECT TRANSISTORS

General Form JXXXXXXX ND NG NS MNAME [AREA] [OFF] [IC=VDS,VGS]

Example J1 7 2 3 JM1 OFF

ND, NG, and NS are the drain, gate, and source nodes, respectively. MNAME is the model name, AREA is the area factor, and OFF indicates an (optional) initial condition on the device for DC analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification using IC=VDS, VGS is intended for use with the UIC option in the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point.

6.11.4 MOSFETS

General Form MXXXXXXX ND NG NS NB MNAME [W=VAL] [L=VAL]
[AD=VAL] [AS=VAL] + [PD=VAL] [PS=VAL] [OFF]
[IC=VDS,VGS,VBS]

Example M31 2 17 6 10 MODM L=2MIL W=0.5MIL

ND, NG, MS, and NB are the drain, gate, source, and bulk (substrate) nodes, respectively. MNAME is the model name. W and L are the channel width and length, in meters; if omitted, both W and L are assumed to be 1.0. AD and AS are the areas of the drain and source diffusions, in M^2; if not specified, both areas are assumed to be 1.0E-10. PD and PS are the peripheral lengths of the drain and source diffusions, in M. If not specified, both lengths are assumed to be 1.0E-6. OFF indicates an (optional) initial condition on the device for DC analysis. The (optional) initial condition specification using IC=VDS,VGS,VBS is intended for use with the UIC option in the .TRAN statement, when a transient analysis is desired starting from other than the quiescent operating point.

7 .MODEL STATEMENT

General Form .MODEL MNAME TYPE(PNAME1=PVAL1 PNAME2=PVAL2 ...)

Example .MODEL MOD1 NPN(BF=50 IS=1E-13 VA=50)

The .MODEL statement specifies a set of model parameters that will be used by one or more devices. MNAME is the model name, and TYPE is one of the following seven types:

NPN	NPN BJT MODEL
PNP	PNP BJT MODEL
D	DIODE MODEL
NJF	N-CHANNEL JFET MODEL
PJF	P-CHANNEL JFET MODEL
NMOS	N-CHANNEL MOSFET MODEL
PMOS	P-CHANNEL MOSFET MODEL

Parameter values are defined by appending the parameter name, as given below for each model type, followed by an equal sign and the parameter value. Model parameters that are not given a value are assigned the default values given below for each model type.

7.1 DIODE MODEL

The DC characteristics of the diode are determined by the parameters IS and N. An ohmic resistance, RS, is included. Charge storage effects are modeled by a transit time, TT, and a nonlinear depletion layer capacitance which is determined by the parameters CJO, PB, M, CJSW and MSW. The temperature dependence of the saturation current is defined by the parameters EG, the energy gap, and PT, the saturation current temperature exponent. Reverse breakdown is modeled by an exponential increase in the reverse diode current and is determined by the parameters BV and IBV (both of which are positive numbers). A value of 0.0 for BV means that no breakdown effects will be modeled.

	AREA	NAME	PARAMETER	DEFAULT	TYPICAL
1	*	IS	Saturation Current	1.0E-14	1.0E-14
2	*	RS	Ohmic Resistance	0	10
3		N	Emission Coefficient	1	1.0
4		TT	Transit Time	0	0.1 NS
5	*	CJO	Zero-Bias Junction Capacitance	0	2PF
6		PB	Junction Potential	1	0.6
7		M	Grading Coefficient	0.5	0.5
8		EG	Energy Gap	1.11	1.11 SI 0.69 SBD 0.67 GE

17

	AREA	NAME	PARAMETER	DEFAULT	TYPICAL
9		PT	Saturation Current Temp. Exponent	3.0	3.0 JN 2.0 SBD
10		KF	Flicker Noise Coefficient	0	
11		AF	Flicker Noise Exponent	1	
12		FC	Forward-Bias Nonideal Junction Capacitance Coefficient	0.5	
13		BV	Reverse Breakdown Voltage	0.0	40.0
14		IBV	Current at Breakdown Voltage	1.0E-3	
15	peri phery	CJSW	Zero-bias junction capacitance for the periphery	0	2PF
16		MSW	grading coefficient for peripheral capacitance	0.33	0.33

7.2 BJT MODELS (BOTH NPN AND PNP)

The bipolar junction transistor model in ZySPICE is an adaption of the integral charge control model of Gummel and Poon; however, if the additional Gummel-Poon parameters are not specified, the simpler Ebers-Moll model of ZySPICE, version 1, is used. The DC model is defined by the parameters BF, C2, IK, and NE, which determine the forward current gain characteristics, BR, C4, IKR, and NC, which determine the reverse current gain characteristics, VA and VB, which determine the output conductance for forward and reverse regions, and the saturation current, IS. Three ohmic resistances, RB, RC, and RE, are included. Base charge storage is modeled by forward and reverse transit times, TF and TR, and nonlinear depletion layer capacitances which are determined by CJE, PE, and ME for the B-E Junction, and CJC, PC, and MC for the B-C Junction. A constant collector-substrate capacitance, CCS, is also included. The temperature dependence of the saturation current is determined by the energy gap, EG, and the saturation current temperature exponent, PT.

The -DELAY- parameter may be used to specify group delay for the device model. Two forms of specification are recognized, based on the parameter value. If the value is less than unity, ZySPICE assumes that the number given is the (frequency-independent) group delay time. If the value is greater than unity, ZySPICE treats the number given as the phase delay, in degrees, at a frequency of 1/(2*PI*TF).

	AREA	NAME	PARAMETER	DEFAULT	TYPICAL
1		BF	Ideal Forward Current Gain	100	100
2		BR	Ideal Reverse Current Gain	1	0.1
3	*	IS	Saturation Current	1.0E-14	1.0E-16
4	*	RB	Base Ohmic Resistance	0	100
5	*	RC	Collector Ohmic Resistance	0	10
6	*	RE	Emitter Ohmic Resistance	0	1
7		VA	Forward Early Voltage	Infinite	200
8		VB	Reverse Early Voltage	Infinite	200

	AREA	NAME	PARAMETER	DEFAULT	TYPICAL
9	*	IK	Forward High-Current Knee Current	Infinite	10MA
10		C2	Forward Low-Current Nonideal Base Current Coefficient	0	1000
11		NE	Nonideal Low-Current Base-Emitter Emission Coefficient	2.0	2.0
12	*	IKR	Reverse High-Current Knee Current	Infinite	100MA
13		C4	Reverse Low-Current Nonideal Base Current Coefficient	0	1.0
14		NC	Nonideal Low-Current Base-Collector Emission Coefficient	2.0	2.0
15		TF	Forward Transit Time	0	0.1NS
16		TR	Reverse Transit Time	0	10NS
17	*	CCS	Collector-Substrate Capacitance	0	2PF
18	*	CJE	Zero-Bias B-E Junction Capacitance	0	2PF
19		PE	B-E Junction Potential	1.0	0.7
20		ME	B-E Junction Grading Coefficient	0.5	0.33
21	*	CJC	Zero Bias B-C Junction Capacitance	0	1PF
22		PC	B-C Junction Potential	1.0	0.5
23		MC	B-C Grading Coefficient	0.5	0.33
24		EG	Energy Gap	1.11	1.11 SI 0.67 GE
25		PT	Saturation Current Temp. Exponent	3.0	
26		KF	Flicker Noise Coefficient	0	6.6E-16 NPN 6.3E-13 PNP
27		AF	Flicker Noise Exponent	1	1.0 NPN
28		FC	Forward-Bias Nonideal Junction Capacitance Coefficient	0.5	1.5 PNP
29		DELAY	Group Delay	0	30 (DEG)

7.3 JFET MODELS (BOTH N AND P CHANNEL)

The JFET model is derived from the FET model of Shichman and Hodges. The DC characteristics are defined by the parameters VTO and BETA, which determine the variation of drain current with gate voltage, LAMBDA, which determines the output conductance, and is, the saturation current of the two gate junctions. Two ohmic resistances, RD and RS, are included. Charge storage is modeled by nonlinear depletion layer capacitances for both gate junctions which vary as the -1/2 power of junction voltage and are defined by the parameters CGS, CGD, and PB.

	AREA	NAME	PARAMETER	DEFAULT	TYPICAL
1		VTO	Threshold Voltage	-2.0	-2.0
2	*	BETA	Transconductance Parameter	1.0E-4	1.0E-3
3		LAMBDA	Channel Length Modulation Parameter	0	1.0E-4
4	*	RD	Drain Ohmic Resistance	0	100
5	*	RS	Source Ohmic Resistance	0	100
6	*	CGS	Zero-Bias G-S Junction Capacitance	0	5PF
7	*	CGD	Zero-Bias G-D Junction Capacitance	0	1PF
8		PB	Gate Junction Potential	1	0.6
9	*	IS	Gate Junction Saturation Current	1.0E-14	1.0E-14
10		KF	Flicker Noise Coefficient	0	
11		AF	Flicker Noise Exponent	1	
12		FC	Forward-Bias Nonideal Junction Capacitance Coefficient	0.5	

7.4 MOSFET MODELS (BOTH N AND P CHANNEL)

ZySPICE provides three MOSFET device models which differ in the formulation of the I-V characteristic. The variable *LEVEL* specifies the model to be used:

LEVEL=1 -> Shichman-Hodges
LEVEL=2 -> MOS2 (as described in [1])

The DC characteristics of the MOSFET are defined by the device parameters VTO, KP, LAMBDA, PHI, and GAMMA. These parameters are computed by ZySPICE if process parameters (NSUB, TOX, ...) are given, but user-specified values always override. VTO is positive (negative) for enhancement mode and negative (positive) for depletion mode N-channel (P-channel) devices. Charge storage is modeled by three constant capacitors, CGS, CGD, and CGB, which represent drawn overlap capacitance for LEVEL=2 and LEVEL=3 models, by the nonlinear oxide capacitance (for LEVEL=1, only when TOX is specified) distributed among the gate-source, gate-drain, and gate-bulk regions using the formulation of J.E. Meyer, and by the nonlinear depletion-layer capacitances for both substrate junctions which vary as the -1/2 power of junction voltage and are determined by the parameters CBD, CBS, CDP, CSP, and PB.

---

[1] D.R. Alexander, et al, "SPICE2 MOS Modeling Handbook," BDM Corp., Report BDM/A-77-071-TR, Albuquerque, N.M., May 1977.

	NAME	PARAMETER	DEFAULT	TYPICAL	UNITS
1	LEVEL	Model Complexity	1		
2	VTO	Zero-Bias Threshold Voltage	0.0	-0.1	V
3	KP	Intrinsic Transconductance Parameter	2.0E-5	2.5E-5	A/V^2
4	GAMMA	Bulk Threshold Parameter	0.0	0.37	V^(1/2)
5	PHI	Surface Potential at Strong Inversion	0.6	0.65	V
6	LAMBDA	Channel-Length Modulation Parameter	0.0	0.02	/V
7	RD	Drain Ohmic Resistance	0.0	1.0	OHMS
8	RS	Source Ohmic Resistance	0.0	1.0	OHMS
9	CGS	Drawn Gate-Source Overlap Capacitance Per M Channel Width	0.0	4.0E-11	F/M
10	CGD	Drawn Gate-Drain Overlap Capacitance Per M Channel Width	0.0	4.0E-11	F/M
11	CGB	Gate-Bulk Overlap Capacitance Per M Channel Length	0.0	2.0E-10	F/M
12	CBD	Zero-Bias B-D Junction Capacitance Per M^2 of Junction Area	0.0	2.0E-4	F/M^2
13	CBS	Zero-Bias B-S Junction Capacitance Per M^2 of Junction Area	0.0	2.0E-4	F/M^2
14	TOX	Oxide Thickness If LEVEL=1.....Infinity Otherwise.....1.0E-7		1.0E-7	M
15	PB	Bulk Junction Potential	0.8	0.87	V
16	JS	Bulk Junction Reverse Saturation Current Per M^2 of Junction Area	1.0E-4	1.0E-4	A/M^2
17	NSUB	Effective Substrate Doping If LEVEL not 3.....0.0 If LEVEL=3.....1.0E21		4.0E21	/M^3
18	NSS	Effective Surface State Density	0.0	1.0E14	/M^2
19	NFS	Effective Fast Surface State Density	0.0	1.0E14	/M^2
20	XJ	Metallurgical Junction Depth	0.0	1.0E-6	M
21	LD	Lateral Diffusion Coefficient	0.8	0.8	
22	NGATE	Polysilicon Gate Doping	Al Gate	1.0E26	/M^3
23	TPS	Type of Polysilicon: +1 (-1) for Opposite (Same) as Substrate	+1.0		

	NAME	PARAMETER	DEFAULT	TYPICAL	UNITS
24	UO	Surface Mobility	0.06	0.06	M^2/V-S
25	UCRIT	Critical Field for Mobility Degradation	1.0E+6	1.0E+6	V/M
26	UEXP	Critical Field Exponent (Mobility)	0.0	0.1	
27	UTRA	Transverse Field Coefficient (Mobility)	0.0	0.3	
28	KF	Flicker Noise Coefficient	0.0	1.0E-26	
29	AF	Flicker Noise Exponent	1.0	1.2	
30	FC	Forward-Bias Nonideal Junction Capacitance Coefficient	0.5		
31	CDP	Zero Bias B-D Peripheral Capacitance Per M of Length	0.0		F/M
32	CSP	Zero Bias B-S Peripheral Capacitance Per M of Length	0.0		F/M
33	VMAX	Maximum Drift Velocity of Carriers	0.0	5.0E+4	M/S
34	NEFF	Total Channel Charge (Fixed and Mobile) Coefficient	1.0	5.0	

NOTE: If NSUB is specified and no values are input for CBD and CBS, the Zero Bias Junction Capacitances will be computed assuming a one-sided step junction.

8 SUBCIRCUITS

A subcircuit that consists of ZySPICE elements can be defined and referenced in a fashion similar to device models. The subcircuit is defined in the input file by a grouping of element statements; the program then automatically inserts the group of elements wherever the subcircuit is referenced. There is no limit on the size or complexity of subcircuits, and subcircuits may contain other subcircuits. An example of subcircuit usage is given in Appendix A.

8.1 .SUBCKT STATEMENT

General Form .SUBCKT SUBNAM N1 [N2 N3 ...]

Example .SUBCKT OPAMP 1 2 3 4

A subcircuit definition is begun with a .SUBCKT statement. SUBNAM is the subcircuit name, and N1, N2, ... are the external nodes, which cannot be zero. The group of element statements which immediately follow the .SUBCKT statement define the subcircuit. The last statement in a subcircuit definition is the .ENDS statement (see below). Control statements may not appear within a subcircuit definition; however, subcircuit definitions may contain anything else, including other subcircuit definitions, device models, and subcircuit calls (see below). Note that any device models or subcircuit definitions included as part of a subcircuit definition are strictly local (i.e., such models and definitions are not known outside the subcircuit definition). Also, any element nodes not included in the .SUBCKT statement are strictly local, with the exception of 0 (ground) which is always global.

8.2 .ENDS STATEMENT

General Form .ENDS [SUBNAM]

Example .ENDS OPAMP

This statement must be the last one for any subcircuit definition. The subcircuit name, if included, indicates which subcircuit definition is being terminated; if omitted, all subcircuits being defined are terminated. The name is needed only when nested subcircuit definitions are being made.

8.3 SUBCIRCUIT CALLS

General Form XYYYYYYY N1 [N2 N3 ...] SUBNAM

Example X1 2 4 17 3 1 MULTI

Subcircuits are used in ZySPICE by specifying pseudo-elements beginning with the letter X, followed by the circuit nodes to be used in expanding the subcircuit, followed by the subcircuit name. The nodes must be in the order that they are defined in the .SUBCKT statement.

9 CONTROL STATEMENTS

9.1 TITLE STATEMENT

Example POWER AMPLIFIER CIRCUIT

This statement must be the first statement in the input file. Its content are printed verbatim as the heading for each section of output.

9.2 .END STATEMENT

Example .END

This statement must always be the last statement in the input file. Note that the period is an integral part of the name.

9.3 COMMENT

General Form * Any Comments

Example * RF=1K GAIN SHOULD BE 100

This statement is printed out in the input listing but is otherwise ignored.

9.4 .TEMP STATEMENT

General Form .TEMP T1 [T2 [T3 ...]]

Example .TEMP -55.0 25.0 125.0

This statement specifies the temperatures at which the circuit is to be simulated. T1, T2, ... are the different temperatures, in degrees C. Temperatures less than -223.0 Degrees C are ignored. Model data is specified at TNOM degrees (see the .OPTIONS statement for TNOM); if the .TE statement is omitted, the simulation also will be performed at a temperature of TNOM.

9.5 .WIDTH STATEMENT

General Form .WIDTH IN=COLNUM OUT=PAGE_WIDTH

Example .WIDTH IN=72

COLNUM is the last column read from each line of input; the setting takes effect with the next line read. The default value for colnum is 80.

PAGE_WIDTH is the number of print positions on the output medium. The size of any plots is 32 less than the PAGE_WIDTH. Thus for the default PAGE_WIDTH of 132 characters, the plots will have 100 points.

9.6 .OPTIONS STATEMENT

General Form .OPTIONS OPT1 OPT2 ... (or OPT=OPTVAL ...)

Example .OPTIONS ACCT LIST NODE

This statement allows the user to reset program control and user options for specific simulation purposes. Any combination of the following options may be included, in any order. "X" (below) represents some positive numbers.

OPTION	EFFECT
ACCT	Causes the execution time for the various sections of the program to be printed as well as other accounting information to be printed.
LIST	Causes the summary listing of the input data to be printed.
NOMOD	Suppresses the printout of the model parameters.
NOPAGE	Suppresses page eject.
NODE	Causes the node table to be printed.
OPTS	Causes the option values to be printed.
GMIN=X	Resets the value of GMIN, the minimum conductance allowed by the program. The default value is 1.0E-12.
RELTOL=X	Resets the relative error tolerance of the program. The default is 0.001 (0.1 percent).
ABSTOL=X	Resets the absolute current error tolerance of the program. The default value is 1 picoamp.
VNTOL=X	Resets the absolute voltage error tolerance of the program. The default value is 1 microvolt.
TRTOL=X	Resets the transient error tolerance. The default value is 7.0. This parameter is an estimate of the factor by which ZySPICE overestimates the actual truncation error.

CHGTOL=X Resets the charge tolerance of the program. The default value is 1.0E-14.

NUMDGT=X Resets the number of significant digits printed for output variable values. X must satisfy the relation $0 < X < 8$. The default value 4.

NOTE: This option is independent of the error tolerance used by ZySPICE (i.e., if the values of options RELTOL, ABSTOL, etc. are not changed then one may be printing numerical "noise" for NUMDGT > 4.)

TNOM=X Resets the nominal temperature. The default value is 27 Degrees C (300 Degrees K).

ITL1=X Resets the DC iteration limit. The default if 100.

ITL2=X Resets the DC transfer curve iteration limit. The default is 20.

ITL3=X Resets the lower transient analysis iteration limit. The default value is 4.

ITL4=X Resets the transient analysis timepoint iteration limit. The default is 10.

ITL5=X Resets the transient analysis total iteration limit. The default is 5000.

LIMTIM=X Resets the amount of time reserved by ZySPICE for generating plots. The default value is 2 (seconds).

LIMPTS=X Resets the total number of points that can be printed or plotted in a DC, AC, or transient analysis. The default value is 201.

LVLCOD=X If X is 2 (two), then machine code for the matrix solution will be generated. Otherwise, no machine code is generated. The default value is 2.

LVLTIM=X If X is 1 (one), the iteration timestep control is used. If X is 2 (two), the truncation error timestep is used. The default value is 2. If METHOD=GEAR and MAXORD>2 then LVLTIM is set to 2 by ZySPICE.

METHOD=NAME Sets the numerical integration method used by ZySPICE. Possible names are gear or trapezoidal. The default is trapezoidal.

MAXORD=X

Sets the maximum order for the integration method if GEAR*S variable-order method is used. X must be between 2 and 6. The default is 2.

LIBRARY=ON,OFF

This option turns ON or OFF the printing of input data as it is read from the user specified libraries. The default is OFF.

INCLUDE=ON,OFF

This option turns ON or OFF the printing of input data as it is read from the files specified by the .INCLUDE statement. The default is OFF.

SIZE[R,C,D,J,B,M] = X

The size parameters add to the element values the specified size factor for R and C. For DIODEs and BJTs, size parameter is added to the linear dimension and the areas are recomputed accordingly. For MOS transistors, the size parameter is added to the length and width dimensions but the source and drain diffusion areas and peripheries are not affected. If the option SIZE is specified without any qualifier, size parameters for all elements are set to the specified value.

SCALE[R,C,D,J,B,M] = X

The scale parameters are applied after applying the size parameters. The scale parameters multiply the element values by the specified scale factor for R and C; for other elements, linear dimensions are multiplied by the scale factor and areas are multiplied by the square of the scale factor. If the option SCALE is specified without any qualifier, then the scale factors for all the above elements are set to the specified value and the scale factor for R is set to 1.

The new size and scale factors will override the earlier size and scale factors. Size and scale factors become effective as soon as they are specified. All the subsequent values will be sized and scaled, where applicable, as soon as they are read from the input statement.

9.7 .OP STATEMENT

General Form .OP

The inclusion of this statement in an input file will force ZySPICE to determine the DC operating point of the circuit with inductors shorted and capacitors opened.

NOTE: A DC analysis is automatically performed prior to a transient analysis to determine the transient initial conditions, and prior to an AC small-signal analysis to determine the linearized, small-signal models for non-linear devices.

ZySPICE performs a DC operating point analysis if no other analyses are requested.

9.8 .DC STATEMENT

General Form .DC SRCNAM VSTART VSTOP VINCR

Example .DC VIN 0.25 5.0 0.25

This statement defines the DC transfer curve source and sweep limits. SRCNAM is the name of an independent voltage or current source. VSTART, VSTOP, and VINCR are the starting, final, and incrementing values respectively. The above example will cause the value of the voltage source VIN to be swept from 0.25 volts to 5.0 volts in increments of 0.25 volts.

9.9 .TF STATEMENT

General Form .TF OUTVAR INSRC

Example .TF V(5,3) VIN

This statement defines the small-signal output and input for the DC small-signal analysis. OUTVAR is the small-signal output variable and INSRC is the small-signal input source. If this statement is included, ZySPICE will compute the DC small-signal value of the transfer function (output/input), input resistance, and output resistance. For the above example, ZySPICE would compute the ratio of V(5,3) to VIN, the small-signal input resistance at VIN, and the small-signal output resistance measured across nodes 5 and 3.

9.10 .SENS STATEMENT

General Form .SENS OV1 [OV2 ...]

Example .SENS V(9) V(4,3) V(17)

If a .SENS statement is included in the input file, ZySPICE will determine the DC small-signal sensitivities of each specified output variable with respect to every circuit parameter.

NOTE: For large circuits, large amounts of output can be generated.

9.11 .AC STATEMENT

General Form
```
.AC DEC ND FSTART FSTOP
.AC OCT NO FSTART FSTOP
.AC LIN NP FSTART FSTOP
```

Examples
```
.AC DEC 10 1 10KHZ
.AC OCT 20 1 100KHZ
.AC LIN 100 1 100HZ
```

DEC stands for decade variation, and ND is the number of points per decade. OCT stands for octave variation, and NO is the number of points per octave. LIN stands for linear variation, and NP is the number of points. FSTART is the starting frequency, and FSTOP is the final frequency. If this statement is included in the file, ZySPICE will perform an AC analysis of the circuit over the specified frequency range. Note that in order for this analysis to be meaningful, at least one independent source must have been specified with an AC value.

9.12 .DISTO STATEMENT

General Form .DISTO RLOAD [INTER [SKW2 [REFPWR [SPW2]]]]

Example .DISTO RL 2 0.95 1.0E-3 0.75

This statement controls whether ZySPICE will compute the distortion characteristics of the circuit in a small-signal mode as a part of the AC small-signal sinusoidal steady-state analysis. The analysis is performed assuming that one or two signal frequencies are imposed at the input; let the two frequencies be F1 (the nominal analysis frequency) and F2 (=SKW2*F1). The program then computes the following distortion measures:

HD2	-	The magnitude of the frequency component 2*F1 assumes that F2 is not present.
HD3	-	The magnitude of the frequency component 3F*F1 assumes that F2 is not present.
SIM2	-	The magnitude of the frequency component F1 + F2.
DIM2	-	The magnitude of the frequency component F1 - F2.
DIM3	-	The magnitude of the frequency component 2*F1 -F2.

RLOAD is the name of the output load resistor into which all distortion power products are to be computed. INTER is the interval at which the summary printout of the contributions of all nonlinear devices to the total distortion is to be printed. If omitted or set to zero, no summary printout will be made. REFPWR is the reference power level used in computing the distortion products. If omitted, a value of 1 MW (that is, DBM) is used. SKW2 is the ratio of F2 to F1. If omitted, a value of 0.9 is used (I.E., F2 = 0.9*F1). SPW2 is the amplitude of F2. If omitted, a value of 1.0 is assumed.

The distortion measures HD2, HD3, SIM2, DIM2, and DIM3 may also be printed and/or plotted (see the description of the .PRINT and .PLOT statements).

9.13 .NOISE STATEMENT

General Form .NOISE OUTV INSRC NUMS

Example .NOISE V(5) VIN 10

This statement controls the noise analysis of the circuit. OUTV is a voltage output variable which defines the summing point. INSRC is the name of the independent voltage or current source which is the noise input reference. NUMS is the summary interval. ZySPICE will compute the equivalent output noise at the specified output as well as the equivalent input noise at the specified input. In addition, the contributions of every noise generator in the circuit will be printed at every NUMS frequency points (the summary interval). If NUMS is zero, no summary printout will be made.

The output noise and the equivalent input noise may also be printed and/or plotted (see the description of the .PRINT and .PLOT statements).

9.14 .TRAN STATEMENT

General Form .TRAN TSTEP TSTOP [TSTART [TMAX]] [UIC]

Examples
```
.TRAN 1NS 100NS
.TRAN 1NS 1000NS 500NS
.TRAN 10NS 1US UIC
```

TSTEP is the printing increment, TSTOP is the final time, and TSTART is the initial time. If TSTART is omitted, it is assumed to be zero. The transient analysis always begins at time zero. In the interval [ZERO, TSTART], the circuit is analyzed (to reach a steady state), but no outputs are stored. In the interval [TSTART, TSTOP], the circuit is analyzed and outputs are stored. TMAX is the maximum stepsize that ZySPICE will use (default value is a function of the circuit data).

UIC (use initial conditions) is an optional keyword which indicates that the user does not want ZySPICE to solve for the quiescent operating point before beginning the transient analysis. If this keyword is specified, ZySPICE uses the values specified using IC=... on the various elements as the initial transient condition and proceeds with the analysis.

9.15 .FOUR STATEMENT

General Form .FOUR FREQ OV1 [OV2 OV3 ...]

Example .FOUR 100KHZ V(5)

This statement controls whether ZySPICE performs a fourier analysis as a part of the transient analysis. FREQ is the fundamental frequency, and OV1, ..., are the output variables for which the analysis is desired. The fourier analysis is performed over the interval [TSTOP-PERIOD, TSTOP], where TSTOP is the final time specified for the transient analysis, and period is one period of the fundamental frequency. The DC component and the first nine components are determined. For maximum accuracy, TMAX (see the .TRAN statement) should be set to period/100.0 (or less for very high-Q circuits).

9.16 .PRINT STATEMENT

General Form .PRINT PRTYPE OV1 [OV2 ... OV8]

Examples
```
.PRINT TRAN V(4) I(VIN)
.PRINT AC VM(4,2) VR(7) VP(8,3)
.PRINT DC V(2) I(VSRC) V(23,17)
.PRINT NOISE INOISE
.PRINT DISTO HD3 SIM2(DB)
```

This statement defines the contents of a tabular listing of one to eight output variables. PRTYPE is the type of the analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The form for voltage or current output variables is as follows:

V(N1[,N2])

Specifies the voltage difference between nodes N1 and N2. If N2 (and the preceding comma) is omitted , ground (0) is assumed. For the AC analysis, five additional outputs can be accessed by replacing the letter V by:

VR - Real Part
VI - Imaginary Part
VM - Magnitude
VP - Phase
VDB - 20*Log10(Magnitude)

I(VXXXXXXX)

Specifies the current flowing in the independent voltage source named VXXXXXXX. Positive current flows from the positive node, through the source, to the negative node. For the AC analysis, the corresponding replacements for the letter I may be made in the same way as described for voltage outputs.

NOTE: VM(8,9,) is the AC magnitude of the voltage difference, not VM(8)-VM(9).

Output variables for the noise and distortion analyses have a different form from that of the other analyses. The general form is:

OV[(X)]

Where OV is any of ONOISE (output noise), INOISE (equivalent input noise), HD2, HD3, SIM2, DIM2, or DIM3 (see description of distortion analysis), and X may be any of:

R - Real Part
I - Imaginary Part
M - Magnitude (default if nothing specified)
P - Phase
DB - 20*Log10(Magnitude)

Thus, SIM2 (or SIM2(M)) describes the magnitude of the SIM2 distortion measure, while HD2(R) describes the real part of the HD2 distortion measure.

9.17 .PLOT STATEMENT

General Form
```
.PLOT PLTYPE OV1 [(PL01,PHI1)] [OV2 [(PL02,PHI2)]
+ ... OV8]
```

Examples
```
.PLOT DC V(4) V(5) V(1)
.PLOT TRAN V(17,5) (2,5) I(VIN) V(17) (1,9)
.PLOT AC VM(5) VM(31,24) VDB(5) VP(5)
.PLOT DISTO HD2 HD3(R) SIM2
```

This statement defines the contents of one plot of from one to eight output variables. PLTYPE is the type of analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The syntax for the OVI is identical to that for the .PRINT statement, described above.

1

The optional plot limits (PLO, PHI) may be specified after any of the output variables. All output variables to the left of a pair of plot limits (PLO, PHI) will be plotted using the same lower and upper plot bounds. If plot limits are not specified, ZySPICE will automatically determine the minimum and maximum values of all output variables being plotted and scale the plot to fit. More than one scale will be used if the output variable values warrant (i.e., mixing output variables with values which are orders-of-magnitude different still gives readable plots). The overlap of two or more traces on any plot is indicated by the letter X.

9.18 .INITIAL STATEMENT

General Form .INITIAL V(N+[,N-])=VALUE [V(N+[,N-])=VALUE .....]

Example .INITIAL V(2,5)=10 V(3)=5

This statement controls the specification of any node voltage, or node voltage difference. During DC and initial transient analyses, this statement effectively adds voltage sources to the circuit. During transient analysis the "pseudo voltage sources" are absent. The inclusion of these "pseudo voltage sources" must not create any voltage source loops.

N+ and N- are the positive and negative nodes, respectively. Note that "pseudo voltage sources" need not be grounded. If N- is not specified it is assumed to be grounded. Current is assumed to flow from the positive node, through the source, to the negative node.

9.19 .INCLUDE STATEMENT

General Form .INCLUDE filename

Examples .INCLUDE FILE1

This statement includes the contents of a specified file in the main input stream. The specified file is opened and the program starts reading from that file, until end of file is encountered or another .INCLUDE statement is encountered. Up to 10 files can be open at one time. The main file should have the title card and the .END statement. The included file should not contain a .END statement and the first line in the included file should not be a continuation line. The data from the included file is not printed unless specifically requested by using the INCLUDE=ON option in the .OPTIONS statement.

9.20 .USRMOD STATEMENT

General Form .USRMOD filename

Examples .USRMOD MYMODELS

This statement specifies the name of the user model library. This is the first library read after encountering the .END statement. The entire model library file will be read, irrespective of whether these models are referenced in the circuit description. The format of model description is identical to the regular input format. The data from the user model library is not printed unless specifically requested using the LIBRARY=ON option in the .OPTIONS statement. The specified user model library remains effective until it is changed by another .USRMOD statement or until it is specifically disabled by .DUSRMOD statement.

9.21 .DUSRMOD STATEMENT

General Form .DUSRMOD

Example .DUSRMOD

This command disables the user model library.

9.22 .USRSUB STATEMENT

General Form .USRSUB directory name

Examples .USRSUB MYSUBCKTS

This statement specifies the name of the user subcircuit library directory. The subcircuits are read after reading of the user model library is completed. Each subcircuit description should be in a separate file under the specified directory. The format of the subcircuit description is identical to the regular input format. The program will read only those subcircuits which are referenced in the circuit description. The data from the user subcircuit library is not printed unless specifically requested using the LIBRARY=ON option in the .OPTIONS statement. The specified user subcircuit library remains effective until it is changed by another .USRSUB statement or until it is specifically disabled by .DUSRSUB statement.

9.23 .DUSRSUB STATEMENT

General Form .DUSRSUB

Examples .DUSRSUB

This command disables the user subcircuit library.

9.24 .POST STATEMENT

General Form .POST [filename]

Examples .POST
.POST MYFILE

This command is used to generate an output file to be used by SPIGRAPH. SPIGRAPH is a post-processor for ZySPICE which produces graphical output on TeleVideo 950 or VT125 terminals equipped with graphics hardware. The output file generated will be called <filename>.RAW or <inputfilename>.RAW if filename is not specified.

APPENDICES

A.1 EXAMPLES

The following file determines the DC operating point and small-signal transfer function of a simple differential pair.

```
SIMPLE DIFFERENTIAL PAIR
VCC 7 0 12
VEE 8 0 -12
VIN 1 0
RS1 1 2 1K
RS2 6 0 1K
Q1 3 2 4 MOD1
Q2 5 6 4 MOD1
RC1 7 3 10K
RC2 7 5 10K
RE 4 8 10K
.MODEL MOD1 NPN(BF=50 VA=50 IS=1.0E-12 RB=100)
.TF V(5) VIN
.END
```

The following file determines the DC transfer curve and the transient pulse response of a simple RTL inverter. The input is a pulse from 0 to 5 volts with delay, rise, and fall times of 2NS and a pulse width of 30NS. The transient interval is 0 to 100NS, with printing to be done every nanosecond.

```
SIMPLE RTL INVERTER
VCC 4 0 5
VIN 1 0 PULSE(0 5 2NS 2NS 2NS 30NS)
RB 1 2 10K
Q1 3 2 0 Q1
RC 3 4 1K
.PLOT DC V(3)
.PLOT TRAN V(3) (0.5)
.PRINT TRAN V(3)
.MODEL Q1 NPN(BF=20 RB=100 TF=0.1NS CJC=2PF)
.DC VIN 0 5 0.1
.TRAN 1NS 100NS
.END
```

The following file determines the AC small-signal response of a one-transistor amplifier over the frequency range 1HZ to 100 MEGHZ.

```
ONE-TRANSISTOR AMPLIFIER
VCC 5 0 12
VEE 6 0 -12
VIN 1 0 AC 1
RS 1 2 1K
Q1 3 2 4 X33
RC 5 3 500
RE 4 6 1K
CBYPASS 4 0 1UFD
.PLOT AC VM(3) VP(3)
.AC DEC 10 1HZ 100MEGHZ
.MODEL X33 NPN(BF=30 RB=50 VA=20)
.END
```

The following file simulates a four-bit binary adder, using several subcircuits to describe various pieces of the overall circuit.

```
ADDER - 4 BIT ALL-NAND-GATE BINARY ADDER

*** SUBCIRCUIT DEFINITIONS

.SUBCKT NAND 1 2 3 4
*   NODES:  INPUT(2), OUTPUT, VCC
Q1 9 5 1 QMOD
D1CLAMP 0 1 DMOD
Q2 9 5 2 QMOD
D2CLAMP 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDROP 10 3 DMOD
Q5 3 8 0 QMOD
.ENDS NAND
.SUBCKT ONEBIT 1 2 3 4 5 6
*   NODES:  INPUT(2), CARRY-IN, OUTPUT, CARRY-OUT, VCC
X1 1 2 7 6 NAND
X2 1 7 8 6 NAND
X3 2 7 9 6 NAND
X4 8 9 10 6 NAND
X5 3 10 11 6 NAND
X6 3 11 12 6 NAND
X7 10 11 13 6 NAND
X8 12 13 4 6 NAND
X9 11 7 5 6 NAND
.ENDS ONEBIT
```

```
.SUBCKT TWOBIT 1 2 3 4 5 6 7 8 9
*    NODES:  INPUT - BIT0(2) / BIT1(2), OUTPUT - BIT0 / BIT1,
*            CARRY-IN, CARRY-OUT, VCC
X1 1 2 7 5 10 9 ONEBIT
X2 3 4 10 6 8 9 ONEBIT
.ENDS TWOBIT


.SUBCKT FOURBIT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
*    NODES:  INPUT - BIT0(2) / BIT1(2) / BIT2(2) / BIT3(2),
*            OUTPUT - BIT0 / BIT1 / BIT2 / BIT3, CARRY-IN, CARRY-OUT, VCC
X1 2 3 4 9 10 13 16 15 TWOBIT
X2 5 6 7 8 11 12 16 14 15 TWOBIT
.ENDS FOURBIT


*** DEFINE NOMINAL CIRCUIT


.MODEL DMOD D
.MODEL QMOD NPN(BF=75 RB=100 CJE=1PF CJC=3PF)
VCC 99 0 DC 5V
VIN1A 1 0 PULSE(0 3 0 10NS 10NS 10NS 50NS)
VIN1B 2 0 PULSE(0 3 0 10NS 10NS 20NS 100NS)
VIN2A 3 0 PULSE(0 3 0 10NS 10NS 40NS 200NS)
VIN2B 4 0 PULSE(0 3 0 10NS 10NS 80NS 40NS)
VIN3A 5 0 PULSE(0 3 0 10NS 10NS 160NS 800NS)
VIN3B 6 0 PULSE(0 3 0 10NS 10NS 320NS 1600NS)
VIN4A 7 0 PULSE(0 3 0 10NS 10NS 640NS 3200NS)
VIN4B 8 0 PULSE(0 3 0 10NS 10NS 1280NS 6400NS)
X1 1 2 3 4 5 6 7 8 9 10 11 12 0 13 99 FOURBIT
RBIT0 9 0 1k
RBIT1 10 0 1k
RBIT2 11 0 1k
RBIT3 12 0 1k
RCOUT 13 0 1k
.PLOT TRAN V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)
.PLOT TRAN V(9) V(10) V(11) V(12) V(13)
.PRINT TRAN V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)
.PRINT TRAN V(9) V(10) V(11) V(12) V(13)


.TRAN 1NS 6400NS
*** (FOR THOSE WITH MONEY (AND MEMORY) TO BURN)


.OPT ACCT LIST NODE LIMPTS=6401
.END
```

The following file simulates a transmission-line inverter. Two transmission-line elements are required since two propagation modes are excited. In the case of a coaxial line, the first line (T1) models the inner conductor with respect to the shield, and the second line (T2) models the shield with respect to the outside world.

```
TRANSMISSION-LINE INVERTER
V1 1 0 PULSE(0 1 0 0.1N)
R1 1 2 50
X1 2 0 0 4 Tline
R2 4 0 50
.SUBCKT TLINE 1 2 3 4
T1 1 2 3 4 Z0=50 TD=1.5NS
T2 2 0 4 0 Z0=100 TD=1NS
.ENDS TLINE
.TRAN 0.1NS 20NS
.PLOT TRAN V(2) V(4)
.END
```

A.2 NONLINEAR DEPENDENT SOURCES

ZySPICE allows circuits to contain dependent sources characterized by any of the four equations

I=F(V) V=F(V) I=F(I) V=F(I)

where the functions must be polynomials, and the arguments may be multi-dimensional. The polynomial functions are specified by a set of coefficients P0, P1, ..., PN. Both the number of dimensions and the number of coefficients are arbitrary. The meaning of the coefficients depends upon the dimension of the polynomial, as shown in the following examples:

Suppose that the function is one-dimensional (that is, a function of one argument). Then the function value FV is determined by the following expression in FA (the function argument):

FV = P0 + (P1*FA) + (P2*FA^2) + (P3*FA^3) + (P4*FA^4) + (P5*FA^5) + ...

Suppose now that the function is two-dimensional, with arguments FA and FB. Then the function value FV is determined by the following expression:

FV=P0 + (P1*FA) + (P2*FB) + (P3*FA^2) + (P4*FA*FB) + (P5*FB^2) + (P6*FA^3)
+ (P7*FA^2*FB) + (P8*FA*FB^2) + (P9*FB^3) + ...

Consider now the case of a three-dimensional polynomial function with arguments FA, FB, and FC. Then the function value FV is determined by the following expression:

```
FV = P0 + (P1*FA) + (P2*FB) + (P3*FC) + (P4*FA^2) + (P5*FA*FB)
        + (P6*FA*FC) + (P7*FB^2) + (P8*FB*FC) + (P9*FC^2) + (P10*FA^3)
        + (P11*FA^2*FB) + (P12*FA^2*FC) + (P13*FA*FB^2) + (P14*FA*FB*FC)
        + (P15*FA*FC^2) + (P16*FB^3) + (P17*FB^2*FC) + (P18*FB*FC^2)
        + (P19*FC^3) + (P20*FA^4) + ...
```

NOTE: If the polynomial is one-dimensional and exactly one coefficient is specified, then ZySPICE assumes it to be P1 (and P0 = 0.0), in order to facilitate the input of linear controlled sources.

For all four of the dependent sources described below, the initial condition parameter is described as optional. If not specified, ZySPICE assumes 0.0. The initial condition for dependent sources is an initial "guess" for the value of the controlling variable. The program uses this initial condition to obtain the DC operating point of the circuit. After convergence has been obtained, the program continues iterating to obtain the exact value for the controlling variable. Hence, to reduce the computational effort for the DC operating point (or if the polynomial specifies a strong nonlinearity), a value fairly close to the actual controlling variable should be specified for the initial condition.

A.2.1 VOLTAGE-CONTROLLED CURRENT SOURCES

General Form
```
GXXXXXXX N+ N- [POLY(ND)] NC1+ NC1- ... P0
[P1...] + [IC=...]
```

Examples
```
G1 1 0 5 3 0 0.1MMHO
GR 17 3 17 3 0 1M 1.5M IC=2v
GMLT 23 17 POLY(2) 3 5 1 2 0 1M 17M 3.5U
IC=2.5, 1.3
```

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. POLY(ND) only has to be specified if the source is multi-dimensional (one- dimensional is the default). If specified, ND is the number of dimensions, which must be positive. NC1+, NC1-, ... are the positive and negative controlling nodes, respectively. One pair of nodes must be specified for each dimension. P0, P1, P2, ..., PN are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling voltage(s). If not specified, 0.0 is assumed. The polynomial specifies the source current as a function of the controlling voltage(s). The second example above describes a current source with value

$$I = IE-3*V(17,3) + 1.5E-3*V(17,3)^2$$

Note that since the source nodes are the same as the controlling nodes, this source actually models a nonlinear resistor.

A.2.2 VOLTAGE-CONTROLLED VOLTAGE SOURCES

General Form
```
EXXXXXXX N+ N- [POLY(ND)] NC1+ NC1- ... PO [P1...]
+ [IC=...]
```

Examples
```
E1 3 4 21 17 10.5 2.1 1.75
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1
IC=1.5,2.0,17.35
```

N+ and N- are the positive and negative nodes, respectively. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. NC1+, NC1-, ... are the positive and negative controlling nodes, respectively. One pair of nodes must be specified for each dimension. PO, P1, P2, ..., PN are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling voltage(s). If not specified, 0.0 is assumed. The polynomial specifies the source voltage as a function of the controlling voltage(s). The second example above describes a voltage source with value

$$V = V(13,0) + V(15,0) + V(17,0)$$

(In other words, an ideal voltage summer).

A.2.3 CURRENT-CONTROLLED CURRENT SOURCES

General Form
```
FXXXXXXX N+ POLY(ND)] VN1 [VN2 ...] PO [P1 ...]
+ [IC=...]
```

Examples
```
F1 12 10 VCC 1MA 1.3M
FXFER 13 20 VSENS 0 1
```

N+ and N- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. POLY(ND) only has to be specified if the source is multi-dimensional (one- dimensional is the default). If specified, ND is the number of dimensions, which must be positive. VN1, VN2, ... are the names of voltage sources through which the controlling current flows; one name must be specified for each dimension. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of each voltage source. PO, P1, P2, ..., PN are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling current(s) (in amps). If not specified, 0.0 is assumed. The polynomial specifies the source current as a function of the controlling current(s). The first example above describes a current source with value

$$I = 1E\text{-}3 + 1.3E\text{-}3*I(VCC)$$

A.2.4 CURRENT-CONTROLLED VOLTAGE SOURCES

General Form
```
HXXXXXXX N+ N- [POLY(ND)] VN1 [VN2 ...] PO [P1 ...]
+ [IC=...]
```

Examples
```
HXY 13 20 POLY(2) VIN1 VIN2 0 0 0 0 1 IC=0.5 1.3
HR 4 17 VX 0 0 1
```

N+ and N- are the positive and negative nodes, respectively. POLY(ND) only has to be specified if the source is multi-dimensional (one-dimensional is the default). If specified, ND is the number of dimensions, which must be positive. VN1, VN2, ... are the names of voltage sources through which the controlling current flows; one name must be specified for each dimension. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of each voltage source. P0, P1, P2, ..., PN are the polynomial coefficients. The (optional) initial condition is the initial guess at the value(s) of the controlling current(s) (in amps). If not specified, 0.0 is assumed. The polynomial specifies the source voltage as a function of the controlling current(s). The first example above describes a voltage source with value

$$V = I(VIN1)*I(VIN2)$$

ZyPART REFERENCE MANUAL

Doc: 20-010-815 Rev: A	Issued: September 1984	ZyMOS

ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA.
P.O. BOX 62379, SUNNYVALE, CA 94088
TEL. (408) 730-8800 TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

TABLE OF CONTENTS

1 INTRODUCTION

1.1 SCOPE

This manual is a collection of user guides for the group of programs referred to as ZyPART. However, the ZyPART place and route program, ZyPAR, is described in a separate document, the ZyPAR Reference Manual.

1.2 OVERVIEW

ZyPART is a group of programs which perform artwork (composite) generation of standard cell networks in the ZyP Design System. A flow chart showing the relationship and function of the ZyPART programs is given in Figure 1-1.

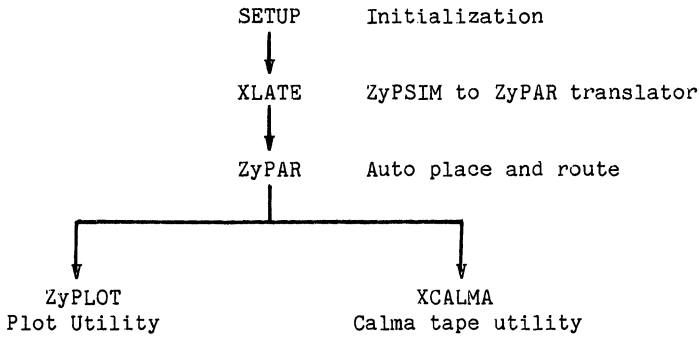


FIGURE 1-1 ZyPART SUMMARY

1.3 RELATED DOCUMENTS

ZyPART is an integral part of the ZyP Design System. To aid in understanding the relationship between ZyPART and ZyP, as well as to aid in execution of ZyPART programs, the following documents are recommended reading.

20-010-101	ZyP Design System Introduction
20-010-103	ZyPNET - PRIME User's Guide
20-010-118	ZyPNET - VAX User's Guide
20-010-415	ZyPAR Reference Manual
20-010-009	ZyPSIM User's Guide

1.4 DOCUMENT CONVENTIONS

In the sample terminal sessions showing program operation, user input is show underlined. For example:

TYPEME

Where illustrations of commands are shown, lower case text is used to indicate additional input. Items shown enclosed in angle brackets < > are user defined. For example:

COMMAND <filename>

Items enclosed in braces { } are optional. For example:

COMMAND <filename> {<-option>}

2 SUMMARY OF OPERATION

2.1 GETTING STARTED

ZyPART requires a circuit network description, process layout rules, and cell model libraries in order to successfully generate a ZyP standard cell circuit composite. Process layout rules and standard cell artwork models are predefined on ZyP. Therefore, all the user needs to provide is the circuit network description. The ZyP system has anticipated the requirement for providing network descriptions to ZyPART. As a matter of fact, the ZyPSIM .NET file already exists. Rather than requiring generation of a new network description, ZyPART has been developed to accept the ZyPSIM .NET file as the circuit network description.

It is recommended that the first step in composite generation using ZyPART is creation of a sub-directory with name ART. Once created, copy the ZyPSIM .NET file into this directory.

On occasion, special cells will be used in ZyP designs. These may be user generated, ZyMEM generated or ZyMOS generated. Special cells require ZyPSIM models for logic simulation and ZyPART models for artwork generation. Specifically, to complete the composite process, XLATE, ZyPAR and Calma data base models will be required for each special cell. ZyPAR models are discussed in the ZyPAR Reference Manual and Calma data base models are provided by the system Calma administrator. XLATE models are discussed in section 4 of this manual.

In summary, to get started on ZyPART:

1. Create an ART sub-directory.

2. Copy the ZyPSIM .NET file into the ART sub-directory.

3. If special cells are used, generate XLATE, ZyPAR and Calma data base models.

NOTE: In the following sections, occasional use is made of filenames with extensions. For VAX generation, a period at the end of these filenames is implied.

2.2 HARDWARE REQUIREMENTS

ZyPART will operate on standard VAX or PRIME computer systems with no special hardware. However, composite generation is greatly aided by the availability of specific graphics resources as follows:

a. Graphics Terminals

ZyPAR will operate on the Tektronix 4113/4115 family of graphics terminals. The 4115 is preferred.

b. Plotters

ZyPLOT will operate on certain dot matrix line printers and Versatec electrostatic plotters. However, all systems may not be supported. Consult your system administrator for information.

c. Graphics Systems

ZyPART generates Calma compatible data base files. These files will typically be loaded onto a Calma (or compatible) system for editing, cell data base merging and pattern generation. Consult your system administrator for information.

2.3 COMPONENTS OF ZyPART

A flow chart showing the relationship and function of the ZyPART programs was presented in Figure 1-1. Each ZyPART program is described further below.

2.3.1 SETUP

SETUP prepares the user directory for composite generation. It is menu driven, prompts the user for circuit information and copies all necessary support files into the user's directory. This includes the ZyPSIM .NET file if it is not already present in the directory.

SETUP also generates a command file with prefix ILOG. Once SETUP has been executed, simply typing ILOG (Initial Lay-Out Generation) will start the ZyPART composite generation process.

Additional information on SETUP is provided in section 3.

2.3.2 XLATE

XLATE is an intelligent ZyPSIM to ZyPAR network description translator. It is normally executed only once and generates files required by ZyPAR. XLATE is run by typing ILOG after SETUP or by typing XLATE.

Additional information on XLATE is provided in section 4.

2.3.3 ZyPAR

ZyPAR is the ZyPART automatic place and route program. The ZyPAR network input file is generated by XLATE. ZyPAR in turn generates a general purpose output file used to create circuit plots and Calma tapes. ZyPAR also generates a ZyPSIM compatible nodal parasitic capacitance file which is used by ZyPSIM to simulate final circuit timing.

Additional information on ZyPAR is provided in the ZyPAR Reference Manual.

2.3.4 ZyPLOT

ZyPLOT is the ZyPART hard copy plotting utility. ZyPLOT uses the ZyPAR generated .PLT file as the input file.

Operation of ZyPLOT is system dependent. Consult your system administrator to determine availability of on-line plotting resources. If plotting is not directly available on your system, use the Calma plot utility as an alternative.

Additional information on ZyPLOT is provided in section 5.

2.3.5 XCALMA

XCALMA is the ZyP composite tape-out utility. XCALMA uses the ZyPAR generated .PLT file to generate a Calma compatible data base tape. This tape can be loaded on the Calma system for editing, plotting or PG generation.

Additional information on XCALMA is provided in section 6.

3 SETUP USER'S GUIDE

3.1 DESCRIPTION

SETUP is the ZyPART initialization program. All input is prompted for from the terminal. Required input is:

* Network file name
* Library
* XLATE and ZyPAR special cell file names

SETUP is invoked by typing SETUP.

3.2 FILES

3.2.1 INPUT FILES

SETUP requires the following input files:

filename.NET	-	ZyPSIM network file
filename.XXX	-	Optional file with special cell XLATE models
filename.XXX	-	Optional file with special cell ZyPAR models

NOTE: Special cell XLATE and ZyPAR files contain all special cell models as required.

3.2.2 OUTPUT FILES

SETUP generates six output files as follows:

ILOG.XXX	-	Command file executed by typing ILOG. The extension XXX is system dependent.
CNTW.XXX	-	Command file executed by ILOG after XLATE. The extension XXX is system dependent.
filename.CEL	-	ZyPAR model file of user defined special cells.
filename.RL1	-	Technology dependent rules file
LOGNTW	-	Modified ZyPSIM network file
XLIB	-	XLATE library including special cells, which are appended.

3.3 OPERATION

3.3.1 WITHOUT SPECIAL CELLS

The following is an example terminal session running SETUP with no special cells. The circuit name is HDEMO and uses the Zy5 library.

```
SETUP <CR>
                                SETUP
                        SETUP PROGRAM FOR ZYPAR
                    Version 5.1.4     Release Date: 09/22/84
                  A proprietary product of ZyMOS Corporation
          477 North Mathilda Ave., Sunnyvale, California 94086, USA

Enter file name (including suffix) (<CR>:Quit):  HDEMO.NET <CR>

Please select Library (Zy4,Zy5,'user_Lib',<CR>=Quit,?=Help)>: ZY5<CR>

Are there any special cells for this chip (Y, N, (<CR>:No)? <CR>

Enter the name of the CHIP : HDEMO <CR>

Processing NETWORK file.  Please wait.
```

3.3.2 WITH SPECIAL CELLS

The following is an example terminal session running SETUP with special cells. The circuit name is HDEMO, the library is Zy5, the ZyPAR special cell model is in file ZYPAR.SCM and the XLATE special cell model is in file XLATE.SCM.

```
SETUP <CR>
                                SETUP
                        SETUP PROGRAM FOR ZYPAR
                    Version 5.1.4     Release Date: 09/22/84
                  A proprietary product of ZyMOS Corporation
          477 North Mathilda Ave., Sunnyvale, California 94086, USA

Enter file name (including suffix) (<CR>:Quit): HDEMO.NET <CR>

Please select Library (Zy4,Zy5,'user_Lib',<CR>=Quit,?=Help)>: ZY5 <CR>

Are there any special cells for this chip (Y, N, (<CR>:No)? Y <CR>

Enter the name of the ZyPAR file. (<CR>:Quit): ZYPAR.SCM <CR>

Enter the name of the XLATE file. (<CR>:Quit): XLATE.SCM <CR>

Enter the name of the CHIP : HDEMO <CR>

Processing NETWORK file.  Please wait.
```

3.4 RULES FILES

ZyPAR uses two rules files, designated filename.RL1 and filename.RL2. Components of the rules files are described in the RULES section of the ZyPAR Reference Manual.

The filename.RL1 file is a system supplied, process dependent, technology file. It is loaded into the user's directory by SETUP and should never be changed by the user.

The filename.RL2 file is generated initially by XLATE, used and modified by ZyPAR. The filename.RL2 file contains rules specific to the current design. It can be examined and modified by the user as part of the interaction with ZyPAR.

4 XLATE USER'S GUIDE

4.1 DESCRIPTION

XLATE is the ZyPSIM to ZyPAR network file translator. The only input required by XLATE, after running SETUP, is the number of cell rows to be used. If the user doesn't know or doesn't care, XLATE will select the number of rows if requested to do so.

XLATE is invoked by typing ILOG after running SETUP.

Normally, XLATE is run only once. However, if the network is changed or cells in the library are changed, XLATE must be run again. In either case, re-run SETUP and invoke XLATE by typing ILOG. For additional information, refer to the ZyPAR Reference Manual sections on changing the library and changing the network.

In performing the ZyPSIM to ZyPAR network translation, XLATE does the following:

1. Un-nests all user defined non-telescoping macros into cell level references.
2. Replaces macro local signal names with global signal names.
3. Removes unused signals from the network.
4. Determines pad locations.
5. Creates special versions of telescoping macros.

4.2 FILES

4.2.1 INPUT FILES

XLATE requires input files as follows:

LOGNTW - Modified ZyPSIM network file generated by SETUP.

XLIB - XLATE cell library including special cells, generated by SETUP.

4.2.2 OUTPUT FILES

XLATE generates the following output files:

filename.NTW	-	Translated LOGNTW network file. Requires further processing by CNTW.XXX, invoked by ILOG, prior to submission to ZyPAR.
LOGNTW.XLO	-	XLATE output file listing which includes LOGNTW, filename.NTW and error messages.
filename.RL2	-	Initial rules 2 file, contains number of cell rows as a minimum.
filename.LIB	-	ZyPAR library model file for telescoping macros.

4.3 OPERATION

XLATE is normally invoked by typing ILOG. If the input files previously described are available, XLATE will execute and prompt only for number of cell rows. Specify the number of rows desired or respond with a <CR> if you wish ZyPAR to specify.

4.4 XLATE LIBRARY DESCRIPTION

XLATE uses a predefined system cell model library to convert a ZyPSIM network file into a network file usable by ZyPAR. Normally, the user need not be concerned with the specification of an XLATE library cell model. However, special cells have to be described to ZyPART so that they can be used in the composite generation. These cell models are stored in a single file which is prompted for by SETUP. XLATE cell model specifications are divided into non-telescoping (regular) cells and telescoping macros as defined below.

4.4.1 NON-TELESCOPING

The XLATE non-telescoping cell model consists of two lines of data. The first line is divided into four fields and the second line provides performance data. The general format is as follows:

Line 1	Cell Name	Cell Type	Inputs	Outputs
Line 2	Load	Rise/Fall pairs		

Specific data for each field is as follows:

LINE 1

Cell Name - User specified descriptive name, eight (8) or less alpha-numeric characters, first character is alpha. Examples are BUF, GATE1, CTR3.

Cell Type - User defined, four characters, first character alpha (except 'A', 'P' or 'Z'), remaining three characters numeric. Examples are G001, F429.

NOTE: For non-telescoping macros, cell name and cell type must be different.

Inputs - Input signals in order of cell model syntax. Each input name is followed by a comma and the input capacitance in units of .0001 pf. If the names are the same the signals are defined to be equivalent. For example, two gate inputs called IN with .3 pf load would be specified as IN,3000 IN,3000 and would be defined as logically equivalent.

Outputs - Output signals in order of cell model syntax. For example, the two outputs of a flip flop could be specified as Q NQ.

LINE 2

Load - No specification needed, enter a 0.

Rise/Fall pairs - A rise/fall pair can be specified for each output. However, none needs be specified for special cells, enter 0 0.

Example 1 - 3 input NAND gate, input capacitance equal to 0.3 pf, no rise/fall data.

```
NAND  G001  IN,3000  IN,3000  OUT
0     0     0
```

Example 2 - D flip flop with Q and NQ outputs, D input capacitance of 0.25 pf, CLK input capacitance of 0.3 pf, no rise/fall data.

```
DFF  F001  D,2500  CLK,3000  Q  NQ
0    0     0      0         0  0
```

4.4.2 TELESCOPING MACRO CELLS

Cells which may be used as part of a telescoping macro have additional information associated with them; the fact that they may only be used inside a macro, and that there are logic signals which are side connections between adjacent telescoping cells.

To indicate that a cell may be used only in a telescoping macro, the cell type and cell name are identical. Signals used as side connections are preceeded by a dollar sign, $. The translator will check the logic network to insure that the side connections are properly connected. In the example below, it will insure that the $CL and $NCL outputs of one cell are connected to the $CL and $NCL inputs of the next cell. To be able to do this, the input signals of a cell must be given the same name as the outputs. For example:

```
SXX1  SXX1  $D,10000  $CL,10000  $NCL,10000  Q  $D  $CL  $NCL
0      0 0     0 0      0 0     0  0
```

In the above example, the $D, $CL and $NCL input signals all come from the previous cell. There is a top or bottom connection named Q. $D, $CL and $NCL are output side connections. The translator will insure that the same signal is used for the $D input signal as was specified for the $D output of the previous cell. Likewise, it will insure that this cell's $D output is used only by the $D input of the next cell.

In many cases, a signal may simply pass thru a cell. The $CL and $NCL signals in the previous example would be an example of this. To simplify both the logic description as well as the translator input file, the output side connections for $CL and $NCL could be dropped. In the case of a macro being built of a control cell and all SXX1 body cells, every SXX1 body cell would require that its $CL and $NCL inputs be the signal names used as $CL and $NCL outputs of the control cell. The translator entry for SXX1 would then be:

```
SXX1  SXX1  $D,10000  $CL,10000  $NCL,10000  Q  $D
0      0 0     0 0
```

As a further extension, in the case where the control cell simply passed its CL and NCL inputs to its output side connections, $CL and $NCL, the control cell could also have its $CL and $NCL outputs omitted.

There are cases where the signal coming out of the side of a cell to the adjacent cell is physically the same signal as one of the top or bottom connections. It is desirable to be able to specify such a signal in the logic network only once, but have it be used as both a top or bottom connection as well as a side connection. In the previous example the Q output could be the same as the $D side output connection. To enter such a cell in the translator file requires that the top or bottom connection be given the same name as the side connection, and further, that it be followed immediately in the I/O list by the side connection. SXX1 would become:

```
SXX1  SXX1  $Q,10000  $CL,10000  $NCL,10000  Q  $Q
0      0  0     0  0
```

In the above entry, the $D signal was renamed $Q to agree with the Q signal. The 'Q $Q' indicates that there will be only one signal in the logic network, but it is to be used as both the top or bottom connection, Q, as well as the side connection, $Q.

5 ZyPLOT USER'S GUIDE

5.1 DESCRIPTION

ZyPLOT is the ZyPART hardcopy plotting utility. ZyPLOT is not supported on all systems, contact your system administrator for availability of plotting resources.

All ZyPLOT input is prompted for from the terminal. Required input is:

* Data Format
* Plot Data Input Source
* Interactive or Batch Mode
* File Name
* Plot Device
* Scale

Alternatively, ZyPLOT supports command line entry. Command options are described in section 5.3.1.

To invoke ZyPLOT in the terminal data entry mode, type ZyPLOT.

5.2 FILES

5.2.1 INPUT FILES

ZyPLOT requires the ZyPAR generated .PLT file as an input file. However, this file can be renamed if desired and specified when running ZyPLOT.

ZyPLOT will also generate plots of Calma data bases on some systems. Calma input files must be provided if plots of Calma data bases are desired.

5.2.2 OUTPUT FILES

ZyPLOT generates no permanent output files.

5.3 OPERATION

5.3.1 COMMAND LINE ENTRY

ZyPLOT can be generated in command line mode by typing a command of the form,

ZyPLOT {<-option ...-option>}

Options are as follows:

OPTION	DESCRIPTION
-HELP	List available options with summary descriptions
-GDS1	Plot data is in Calma GDS1 data base format
-GDS2	Plot data is in Calma GDS2 data base format
-LIST	List plot queues
-DAY	List plot activity for day
-DISK	Source data is on disk
-TAPE	Source data is on tape
-A	Shorthand equivalent of -GDS1 -DISK
-B	Shorthand equivalent of -GDS2 -DISK
-C	Shorthand equivalent of -GDS1 -TAPE
-D	Shorthand equivalent of -GDS2 -TAPE
-KILL	Remove plot from spool queue

5.3.2 QUEUES

ZyPLOT will operate in interactive or batch queue modes. When running, you can specify interactive or batch mode. Also, depending on the system, you may be able to specify several batch queues.

5.3.3 RUNNING

The following is an explanation of data required to run ZyPLOT. All data can be specified from the terminal when invoking ZyPLOT by typing ZyPLOT. Alternatively, command line entry can be used to specify options when starting ZyPLOT. The following explains data entry, whether specified from the terminal or via command line.

a. Data Format

ZyP uses Calma format to represent all composite structures. Either GDS1 or GDS2 can be specified.

b. Plot Data Input Source

Plot data can reside on disk (for example, a ZyPAR generated .PLT file) or tape. Either disk or tape can be specified.

c. Interactive or Batch Mode

As discussed in the QUEUES section above, specify whether you wish to generate in interactive or batch mode.

d. File Name

ZyPLOT requires a plot file name. This is typically the ZyPAR generated .PLT file. Enter file name including extension.

e. Optional Keys

When entering the file name, ZyPLOT will prompt for optional keys. Choices are as follows:

FILL - Generate plot showing all mask layers. Required for all plots except box plot (cell outlines) of .PLT file.

LAYER - Same as FILL but user can specify layers to plot.

PATTERN - Used to change Versatec patterns for mask layers. System will default to standard layers if not specified.

WINDOW - Used to generate sub-plots, user must specify coordinates of window.

NOPLOT - Causes plot to rasterize but not be generated. Normally used only to see if rasterized plot will fit in memory.

Optional keys follow the file name. For example, to generate a filled plot for all IAA1, type,

IAA1 FILL

Multiple keys can be specified.

f. Plot Device

ZyPLOT supports hardcopy plotting on a number of different plotting devices. Select desired device from the menu provided.

g. Scale

ZyPLOT will scale plots to the size desired. Enter a scale factor such as 100 or 625.

Once the above input has been provided, ZyPLOT will execute and generate the requested plot.

6 XCALMA USER'S GUIDE

6.1 DESCRIPTION

XCALMA is the ZyPART Calma tape generation utility. All input is prompted for from the terminal. Required input is:

* File Name
* Library
* Drawing Name
* Prefix
* Suffix
* Tape Drive

XCALMA is invoked by typing XCALMA.

6.2 FILES

6.2.1 INPUT FILES

XCALMA requires the ZyPAR generated .PLT file as an input file. However, this file can be renamed if desired and specified when running XCALMA.

Typically, the ZyPAR generated .PLT file is copied and named .ART when the user is ready to finalize a composite and transfer to the Calma. By renaming the .PLT file, this version of the composite is not lost by restarting ZyPAR. (Restarting ZyPAR will cause a new .PLT file to be generated.)

6.2.2 OUTPUT FILES

XCALMA generates no output files.

6.3 OPERATION

Operation of XCALMA is straight forward, all input can be specified from the terminal. An explanation of requested input follows.

a. Assign Tape Drive

Since XCALMA is a tape out utility, you need to assign a tape drive and mount a blank tape. Calma tapes are normally written at 800 BPI. Contact your system administrator if you are unfamiliar with tape drive generation.

XCALMA will prompt for assignment.

b. Enter Filename

XCALMA prompts for input filename. The required file is the ZyPAR generated .PLT file. However, this file can be renamed and specified at this time. Be sure to include the filename extension, if any.

c. Enter Technology

Respond to the technology prompt in the usual way.

d. Specify Drawing Name

The Calma tape will identify the composite by a drawing name. This name must be eight characters or less and must start with an alpha character.

e. Enter Prefix

When outputting cells, the library prefix must be specified. For standard ZyMOS libraries these are Zy4 and Zy5. For drawings where no prefix is desired, respond with <CR>.

f. Enter Suffix

Again, when outputting cells, the library suffix must be specified. For the ZyP system, library release 5, these are B for Zy4 and A for Zy5. For drawings where no suffix is desired, respond with <CR>.

The request for suffix terminates the terminal entry dialog and a tape will be written as specified.

6.4 NOTE ON CELL NAMES

The Calma system puts some constraints on cell naming; leading character must be alpha and length cannot exceed eight characters. These requirements have been instrumental in naming ZyP cells. A ZyP cell name has three components; prefix, generic name and suffix. For example, the full name of an IAA1 inverter cell in the Zy5 library is ZY5IAA1A. This name uniquely identifies a Calma cell data base and conforms to Calma naming rules.

ZyPAR REFERENCE MANUAL

Doc: 20-010-415 Rev: D	Issued: September 1984	ZyMOS CORP., 477 NO. MATHILDA AVE., SUNNYVALE, CA. P.O. BOX 62379, SUNNYVALE, CA 94088 TEL. (408) 730-8800. TWX 910-339-9530 ZYMOS SUVL

List of Effective Pages

Table of Contents

1 INTRODUCTION

1.1 SCOPE

This manual documents capabilities and usage of ZyPAR, the ZyP automatic place and route program.

1.2 OVERVIEW

ZyPAR is a member of the group of programs which perform artwork (composite) generation of standard cell networks in the ZyP Design System. This group of programs is collectively referred to as ZyPART and all components of ZyPART are described in the ZyPART Reference Manual, except for ZyPAR.

ZyPAR performs automatic placement and routing of standard cell circuits described in a network file. ZyPART programs SETUP and XLATE prepare the circuit ZyPSIM network file (filename.NET) for processing by ZyPAR. ZyPAR in turn generates a general purpose output file used to create circuit plots and Calma tapes. ZyPART programs, ZyPLOT and XCALMA support these tasks. ZyPAR also generates a ZyPSIM compatible nodal parasitic capacitance file which is used by ZyPSIM to simulate final circuit timing.

1.3 RELATED DOCUMENTS

ZyPAR is an integral part of ZyPART and the ZyP Design System. To aid in understanding the relationships among ZyPAR, ZyPART, and ZyP, as well as to aid in execution of ZyPAR, the following documents are recommended reading.

20-010-101	ZyP Design System Introduction
20-010-103	ZyPNET - PRIME User's Guide
20-010-118	ZyPNET - VAX User's Guide
20-010-815	ZyPART Reference Manual
20-010-009	ZyPSIM User's Guide

1.4 DOCUMENT CONVENTIONS

The following conventions are used in this document to describe command format and syntax.

A. <CR> indicates the single typed key RETURN.

B. Lower case input is used to indicate user variable or dependent items. Upper case input is to be entered without change.

2 SUMMARY OF OPERATION

2.1 HARDWARE REQUIREMENTS

For overall ZyPART hardware requirements, refer to the HARDWARE REQUIREMENTS of the ZyPART Reference Manual.

ZyPAR can be run on standard alphanumeric terminals or the Tektronix 4113 and 4115 color graphics terminals. Section nine (9) of this manual explains Tektronix terminal operation. The ZyPAR rule, TEKTRONIX, discussed in section eleven (11), explains how to tell ZyPAR that a Tektronix terminal will be used.

2.2 FILES

2.2.1 INPUT FILES

ZyPAR input files are generated by SETUP and XLATE as follows:

1. filename.LIB — XLATE generated file containing ZyPAR models of telescoping macros.
2. filename.NTW — XLATE generated network file, modified by ILOG (CNTW) used as the ZyPAR network file. (Note: ILOG (CNTW) modifies the chip.NTW file but does not change the name or extension.)
3. filename.RL1 — SETUP generated rules 1 file. The .RL1 file contains technology dependent default rules.
4. filename.RL2 — XLATE generated rules 2 file. The .RL2 file contains technology dependent default rules.
5. filename.SED — XLATE generated file containing bond pad placement.

2.2.2 OUTPUT FILES

ZyPAR generates seven output files as follows:

1. filename.PLT — general plot output file, used by ZyPLOT and XCALMA.
2. filename.CAP — parasitic capacitance file, used by ZyPSIM to simulate final timing.

3. filename.DMP ASCII file of cell placements.

4. filename.SAV binary circuit file.

5. filename.LBS binary ZyPAR library file.

6. filename.LOG listing of commands from latest ZyPAR session.

7. filename.LST statistics and attributes datafile.

2.3 SPECIAL CELLS

If there are special cells in the circuit, these have to be described to ZyPAR in the same format as the standard library. The file containing a description of each of these cells will be copied to a file of the name filename.CEL by SETUP, where filename is the name of the chip.

The ZyPAR special cell file contains the size of the cell and the location of each of the input and output pins. The same name must be used for the I/O pin name as was given in the XLATE file. The first line of the cell contains the name of the cell, its size, the total number of pins, the total number of classes, the number of output classes, and the number of pins on the bottom and on the top.
Format:

INPUT xxxx CELL www hhh nn cc ncl ocl bb tt ll rr

Where: xxxx = cell name

www = cell width

hhh = cell height

nn = total number of I/O connections

ncl = number of classes

ocl = number of output classes

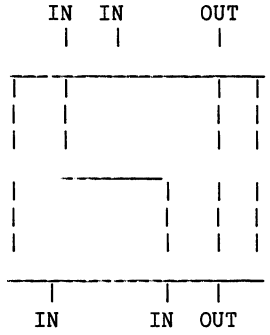
bb = number of bottom I/O connections

tt = number of top I/O connections

ll = number of left I/O connections

rr = number of right I/O connections

The next line is the description of the bottom of the cell. Since ZyPAR thinks in terms of grids, this line contains either the I/O pin name or a 0 (zero) for each pin location, reading left to right. Trailing zeros do not need to be entered. Output I/O pins are preceeded by a slash. On the next line is the capacitive load which each input pin presents to any cells which drive it, and the load which each output pin is capable of driving. The input loads must be negative, the outputs must be positive. If the port is to come out on 2nd level metal, a dollar sign, $, must precede the first letter of the pin name. The format is the same for the top of the cell.



Example: 3 input nand gate

```
INPUT GAA6 CELL 64 272 6 2 1 3 3
IN;A 0 IN /OUT;B
-30000 -30000 300000
IN IN;A 0 /OUT;B
-30000 -30000 300000
```

NOTE: The leftmost top input comes out on the 3rd possible port location from the left on the bottom. The other two inputs do not pass through the cell. The output comes out on both top and bottom.

2.4 GETTING STARTED

Starting a composite design using ZyPART in general and ZyPAR specifically is straight forward. The first step should be a review of the ZyPART and ZyPAR reference manuals. Next, run SETUP and ILOG to initialize files and generate ZyPAR required input files. ILOG will cause ZyPAR to run automatically following XLATE. However, ZyPAR can be run independently by typing ZyPAR as well. Subsequent sections of this manual explain the various steps encountered in completing composite generation through ZyPAR. These include network preparation prior to running SETUP, initial run, additional runs, library modifications, network modifications, and ZyPAR based editing.

3 NETWORK PREPARATION

The network that is used for ZyPSIM simulation must contain the entries which are essential for placement and routing by ZyPAR. The following should be checked prior to execution of ZyPAR.

A. Is the .PADS statement present?

B. Is there a pin number associated with each pin in .PADS?

FORMAT: (pin-name):1, (pin-name):2, etc.

C. Are the pin numbers consecutive, with no numbers skipped?

NOTE: Pin numbers must be present regardless of any need for a specific pin order.

D. Are VDD and VSS pads present both in the .PADs statement and in the network?

PREFERRED FORMAT:
```
VDD PDC1 VDD PWR
VSS PDD1 VSS GND
```

1. PWR and GND could be any name, as long as the name is not used anywhere else in the network.

2. If there is more than one VDD or VSS pad needed, the preferred format for the additional pads is:

```
VDD1 PDC1 VDD1 PWR1
VSS1 PDD1 VSS1 GND1
```

3. The pads in the network are type '1' initially. The pads should be changed to the corresponding types based on the pin locations.

E. Check that pad names used in the .PADS statement are used somewhere within the circuit.

F. Verify that all pads have only one (1) pin signal in the .PADS statement.

G. Check for macros starting with "P" or "A". They can be illegal.

1. Macros starting with "P" tell ZyPAR that those cells are pads, and thus it looks for them in the .PADS statement.

2. Macros starting with "A" tell the router that they are analog cells, and thus must now meet the analog cell requirements, i.e. they must have analog cell endcaps, etc..

H. Check for non-telescoping macros used inside the telescoping macros. These are not allowed.

I. Check for pads inside of macros. Only one pad is allowed.

J. Comment out, with an asterisk, '*', all ZyPSIM primitives.

K. On macro cells with only one pad:

1. The model can have only one input signal.

2. Analog macro pads can have only one input.

L. Macro descriptions must have at least one signal name before a continuation '+' is given.

Example:

```
.MACRO BUFA D        -NOT-     .MACRO BUFA
+CLK  RT                       + D CLK RT
```

4 INITIAL RUN

After the directory has been initialized by running SETUP, an initial placement is run by typing the command: ILOG <CR>. XLATE will be executed and translates the ZyPSIM network file into ZyPAR format, nested MACRO's will be expanded, and a consistency check will be performed. After this, XLATE will ask for the number of rows. If a specific number of rows is required, enter it at this time, if not, simply type a carriage return-- ZyPAR will determine the number of rows.

ZyPAR will perform an initial placement trying several techniques to achieve an optimal result. Several of these placements will be stored as '#BEST_1', '#BEST_2', ..., these may be worked with as well as the one which ZyPAR will select.

When the initial placement has finished, the prompt: ZYPAR> will appear. Any of the commands under Edit Session may be given at this time. A common command which is used at this time is 'SIZE', followed by 'DM*' for second level metal routing and feed cells.

5 ADDITIONAL RUNS

Usually, several edit sessions will be made on a circuit. To return to where ZyPAR left off at the last session, type: ZYPAR filename, where 'filename' is the name of the circuit given to SETUP. Again, all the commands under Edit Session may be given. You may want to review or edit the filename.RL2 file for the ZyPAR rules (referenced in Section 11) before the next ZyPAR session.

6 CHANGING THE LIBRARY

If the library changes between edit sessions, start up an edit session, then type: DUMP <CR>, then EXIT <CR>. A file will be created in the directory by the name: filename.DMP. Perform a new SETUP. Next, edit the filename.RL2 file and delete the line 'RESTORE_LIB'. Perform a new ILOG. ZyPAR will resume with the old placement, but will use the new library cells.

7 CHANGING THE NETWORK

If the logic of the circuit changes, start up an edit session, then type: DUMP <CR>, then EXIT <CR>. A file will be created in the directory by the name: filename.DMP. If the change in the network involves deleting cells from the network, they must be removed form the filename.DMP file. If cells are added to the network, they must be added to the dump file at any reasonable location. The 'RESTORE_LIB' option in the filename.RL2 file should also be deleted. Next perform a new SETUP, and a new ILOG. ZyPAR will resume with the old placement, and new logic.

8 EDIT COMMANDS

These commands may be enterd following the ZyPAR prompt.

8.1 APPEND

Append a cell or a list of cells to the end of the current row. Append or A will not place a non-pad cell at the end of the row if there is a pad already placed there.

abbreviation: A

format: APPEND cell1 cell2 ...

8.2 DELFED

Delete feed-thrus by moving cells from adjacent rows into the row with a feed. The command will eliminate as many feed-thrus as possible.

abbreviation: D

8.3 DOUBLE_METAL

Cause a list of signals to use explicit feeds (i.e. 2nd layer metal), instead of implicit feeds.

abbreviations: DM or DO

format: DOUBLE_METAL signal1 signal2 ...

NOTE: If no signals are specified after the DOUBLEMETAL, a list of the double metal signals will be displayed.

Also, if an asterisk is specified, all signals will use explicit feed thrus.

Whenever possible use explicit feed thrus to improve the performance of the circuit. The ZyMOS standard is explicit feed thrus. The recommended command is DM *

CAUTION: A DUMP or RELOAD loses the DOUBLE_METAL information. After executing a dump command and then reloading the file, the DOUBLE_METAL command must be reentered.

8.4 DUMP

Dump the current placement and orientation of each cell into the file: 'filename.DMP', where 'filename' is the name of the circuit. The DUMP command is used whenever the original network or cell library is changed prior to the next edit session. The filename.RL2 file will be changed such that the placement and orientation specified by the filename.DMP file will be used in the next ZyPAR session.

abbreviation: DU

8.5 D2

Delete feeds, algorithm #2. This method operates several orders of magnitude faster than the command DELFED, and can delete more feeds, but knows nothing about single ported cells.

D2 will not necessarily delete all the feeds. A second and third D2 may delete more feeds. The D2 command may even end up creating additional feed cells. A recommended technique is to use the D2 command until it was unable to delete more feed cells, and then use the slower DELFED command to delete the feeds which are caused by single ported cells. The command can also be used in conjunction with the EVENUP command for a better overall placement.

abbreviation: D2

8.6 EVENUP

Evenup the rows by moving cells out of long rows into short rows. This may cause more feeds to be created.

abbreviation: EV

8.7 EXIT

Exit from ZyPAR. The currently active routing of the circuit will be used to create a filename.PLT file and a filename.CAP file. The placements of the cells will be restored when ZyPAR is run again. All versions of routing held by the SAVE command are lost.

abbreviation: EX

8.8 FIXPOSITION

Fix the position of a list of cells. This will prevent them from moving during subsequent OPTIM's. The FIXPOSITION command will not hold a cell in place if a GLOBAL, DUMP or RESTORE command is given.

abbreviation: FP or FIXP

format: FIXPOSITION cell1 cell2 ... celln

NOTE: If no cells are specified after the FIXPOSITION, a list of the fixed cells will be displayed. A maximum of 40 fixpositions can be specified between D, D2, EV, E2, etc commands.

8.9 FIXORIENTATION

Fix the orientation of a list of cells. This will prevent any of the commands from re-orienting the cells. The FIXORIENTATION command will not hold the orientation if a GLOBAL, DUMP or RESTORE command is given.

abbreviation: FO or FIXO

format: FIXORIENTATION xx cell1 ... celln

where: xx specifies the origin. Options are:

LL Lower Left

LR Lower Right

UL Upper Left

UR Upper Right

NOTE: If no cells are specified after the FIXORIENTATION, a list of the fixed cells will be displayed.

8.10 HELP

Provides information on the use and format of ZyPAR commands.

abbreviation: H

format: HELP command

where command may be one of the following:

APPEND	EXIT	LOGO	RESTORE	UNDOUBLE_METAL
DELETE	FIXPOSITION	OPTIM	ROUTE	UNFIXORIENTATION
DOUBLE	FIXORIENTATION	02	SAVE	UNFIXPOSITION
DUMP	HELP	PLACE	SIZE	1 THRU 99
D2	IC	PLOT	TRACE	
EVENUP	INPUT	QUIT	+	-

8.11 IC

Interchange the position of two cells. The cells will be interchanged even if they are on different rows.

format: IC cell1 cell2

8.12 INPUT

Input edit session commands from a file. When the end of the file is reached, control is returned to the terminal. This file may contain any list of commands which have been found to be useful, or it may be the filename.LOG file created during the previous edit session (if something had gone wrong). Before the .LOG file is used as input, it must be renamed, as it is re-written every edit session. Note that there is a maximum limit of 40 cell movements before a "ROUTE", "EVENUP"," OPTIM" or "02" must be given.

abbreviation: IN

format: INPUT filename

8.13 LOGO

Place the ZyMOS logo and other special edits to the circuit. This requires a Tektronix 4113 or 4115 with tablet, and requires that a PLOT of the circuit has already been sent to the 4113 or 4115. The LOGO command activates the use of the graphics tablet and menu area on the 4113 or 4115.

abbreviation: L

8.14 OPTIM

Optimize the routing by interchanging adjacent cells within a row when this reduces the interconnect length. The 'Y' dimension will be reduced while the 'X' dimension will not be affected. Several passes are made until no furthur improvement is obtained.

abbreviation: O

8.15 O2

"O2" is the optimization technique number two and is much faster (uses less CPU time) than optimization technique "O". This technique operates by placing every cell in each row such that it has the same number of connections pulling it to the left as to the right. It will iterate 9 times. The O2 command works well in conjunction with the OPTIM command since the O2 command looks at the entire row of cells, and the OPTIM command looks at adjacent cells.

abbreviation: O2

8.16 PLACE

Place a list of cells after a specific cell in the current row. The list of cells is given first, then a slash, '/', then the cell after which the list is to be placed.

abbreviation: PL

Format: PLACE cell1 cell2 ... celln / aftercell
or: PLACE cell1 cell2 ... celln

In the first example, cells cell1 thru celln are to be placed after cell 'aftercell'

In the second example, cells 'cell1' thru 'celln' are to be placed at the begining of the current line.

8.17 PLOT

Plot the circuit on the Tektronix 4113 or 4115. This requires the use of a Tektronix 4113 or 4115 as the terminal. A ROUTE is also performed automatically. The PLOT command must not be given when using non Tektronix terminals.

abbreviation: P

8.18 QUIT

Quit ZyPAR and exit without saving any edits from this ZyPAR session.

abbreviation: Q

8.19 RESTORE

Restore a previously saved placement. The same name must be given as was given to the SAVE command to identify the placement.

abbreviation: R

format: RESTORE trialx

NOTE: If no name is specified after the RESTORE, a list of the currently saved placements will be displayed.

8.20 ROUTE

Route the circuit, choosing the best orientation of each cell. The placement of the cells will not change.

abbreviation: RO

8.21 SAVE

Save the current placement for possible future restoration. A name must be given to identify the placement. Up to nine saved placements are supported by ZyPAR. Only the last saved placement is restored when another ZyPAR session is restarted.

abbreviation: S

format: SAVE trialx

NOTE: If no name is specified after the SAVE, a list of saved placements will be displayed.

8.22 SIZE

Print the major statistics on the current size of the circuit. This includes the length of each row, number of feeds and total chip size.

abbreviation: SI

8.23 TRACE

Trace the specified signal on the Tektronix terminal. The signal will appear in yellow. This command requires that a 4113 or 4115 is being used, and that a PLOT command has already been given. To turn the flashing off, use the PLOT command.

abbreviation: T

format: TRACE signal

8.24 UNDOUBLE_METAL

Cause a list of signals to again use implicit feeds instead of explicit feeds (i.e. 2nd layer metal feeds).

abbreviations: UDM UDO UND

format: UNDOUBLE_METAL signal1 signal2 ...

NOTE: If no signals are specified after the UNDOUBLEMETAL, a list of the double metal signals will be displayed.

Also, if an asterisk is specified, all signals will no longer use explicit feed thrus.

8.25 UNFIXORIENTATION

Remove the specified cells from the list of cells which have their orientation fixed.

abbreviation: UFO or UNFIXO

format: UNFIXORIENTATION cell1 ... celln

NOTE: If no cells are specified after the UNFIXORIENTATION, a list of the fixed cells will be displayed.

8.26 UNFIXPOSITION

Remove the specified cells from the list of cells which are fixed in position.

abbreviation: UFP or UNFIXP

format: UNFIXPOSITION cell1 cell2...celln

NOTE: If no cells are specified after the UNFIXPOSITION, a list of the fixed cells will be displayed.

8.27 n

Sets current row to n and displays the cells in row n

8.28 +

Advances to the next row and displays the cells which exist in that row.

8.29 -

Backup to the previous row and displays the cells which exist in that row.

9 OPERATING THE TEKTRONIX 4113 OR 4115 TERMINAL

The Tektronix may be used to display the current routing, and to place special cells, such as the logo, part number, etc.

The screen of the Tektronix will be organized as 34 lines of 80 characters in normal operation outside of ZyPAR, but once in ZyPAR the screen will be arranged into 4 'windows'. At the bottom of the screen will be 5 lines of dialog area. Using the thumbwheels, the dialog area can be scrolled up or down to make up to 100 of the previous lines visible. Above the dialog is the main viewing window, normally used to display either the entire circuit, or whatever portion has been most recently zoomed into. The upper right of the screen is used to zoom in on cell names--this is to allow the viewing of the name of cells which may not be visible in the main view window. Below this is a menu area used in placing the part number, ZyMOS logo, etc.

9.1 OPERATING THE DISPLAY

There are 4 keys on the right side of the keyboard next to the thumbwheels. Two have lights to indicate whether they are active--the Zoom and Pan keys.

The Zoom and Pan keys are push-on, push-off type keys. Push the Zoom key, and the zoom function is activated. Push the zoom key again, and it is de-activated. The same is true for the Pan key.

Whenever the zoom or pan key is active, the thumbwheels manipulate the zoom or pan box--not the scrolling of the text area.

9.1.1 ZOOM

To zoom in on a section of the circuit, press the zoom key. The light on the key will turn on, and a dashed box will appear in the main view window. The thumbwheels may be turned to make the dashed box larger or smaller. When the View key is pushed, the portion of the circuit inside the dashed box will become the next view (i.e. take up the entire main window).

9.1.2 PAN

To pan, i.e. move the dashed box across the screen, press the pan key. The light on the key will turn on, and a dashed box with a plus sign in its center will appear. This box may be moved using the thumbwheels. When the View key is pushed, the portion of the circuit inside the dashed box will become the next view.

9.1.3 RESTORING THE PREVIOUS VIEW

To go back to the previous view of the circuit, press the shift key and the View key simultaneously. This operation may be repeated up to four times to backup to previous views.

9.1.4 VIEWING A CELL NAME

The window on the upper right of the screen is used for viewing the names of the cells without disturbing the drawing in the main view window. This may be done by first setting the zoom box around the name of the cell of interest, then pressing the Next View key twice. This will move the Zoom box to the upper right window. Next press the View key to display the cell name. If the name is still too small, the zoom and pan functions work the same in the upper right window as they do in the main view window.

9.1.5 CANCELING THE DRAWING OF THE CIRCUIT

In many cases, especially with large drawings, it may be desirable to cancel the drawing of the circuit on the screen after pressing the view key. To accomplish this, there is a key in the upper right of the keyboard labeled 'LOCAL' and 'CANCEL'. Pressing the shift key and the LOCAL/CANCEL key simultaneously will cause the Tektronix to stop drawing. Please note that the drawing will stop only if this is during local pan and zoom. The drawing will not stop if this is the display specified by the PLOT command.

9.2 OPERATING THE TABLET

The tablet on the 4113 or 4115 is activated by the LOGO command, and is used to place the ZyMOS logo, the part number, alignment targets, CD cell, pad one indicators, and text. After typing LOGO <CR>, the menu window will be filled with several options from which to choose. The pen may be used to select an option by moving the tip of the pen across the tablet until the cross-hairs on the screen are over the desired option, then pressing down on the pen to select that item. Once a menu option has been selected, a new menu will appear which allows it to be implemented. Once the option is completed, select the 'Done' option, and the previous menu will re-appear.

9.2.1 SPECIAL CELLS

There are several menu selections which allow the placement of special cells -- Logo, CD, Alignment targets, and Pad One. In each of these selections, the pen will operate first in the main window. As the pen is moved, the cell selected will move across the screen. Once the correct position has been found, pressing the pen on the tablet will 'put' it in place. The pen will then operate in the menu area for picking the next operation.

9.2.2 TEXT

Before placing device numbers or other characters, ZyPAR prompts for the characters. Once the characters have been typed, a box will be drawn in the main window which can then be placed in the same manner as the special cells. Once the text has been placed, it may be scaled to the appropriate size by selecting the scale option.

10 STRATEGIES IN OPTIMIZING A CIRCUIT

10.1 INITIAL RUN

The placement from the initial run can be improved with operator assistance. There are at least two aproaches to choose from: try various combinations of the EVENUP, D2, and OPTIM commands on each of the #BEST... placements, or, seed some of the largest cells to fixed positions, and re-try the global placement.

10.1.1 WORKING WITH THE #BEST PLACEMENTS

ZyPAR will try various methods of optimizing the initial placement. The results of each of these are automatically saved during the Global placement. The names of the saved placements are '#BEST_AREA00...04', '#BESTX_DIM00...04', '#BESTY_DIM00...04', for the 5 best area, x dimension, and y dimension placements. Since fewer than 5 techniques are tried, there will be duplicates among these placements. ZyPAR will always start the initial edit session with the best area placement. There are cases where other choices may be better. For example, another placement may be much better in the y direction, but due to a single long row be worse in total area. Often this can be fixed by moving cells out of the long row.

After deciding on a placement to work with, a good strategy may be to try the EVENUP followed by the D2 command, and to repeat the sequence until no furthur improvement in the x-direction is obtained. Optimize the y-direction with the OPTIM command. Any obvious cell movements to improve the routing can be made anytime in the sequence. The final command, however, should be OPTIM. The results can be SAVE'd if it appears to be a good candidate for further work. Then, the next '#BEST' placement can be tried.

10.2 TECHNIQUES FOR SUBSEQUENT EDIT SESSIONS

10.2.1 EVENING UP THE ROWS

The EVENUP command is quite effective in evening up the rows. However, due to the heuristic algorithm that is used, the rows are not as even as they can be. Sometimes, several rows will be even (but short), followed by several other even (but long) rows, followed by more even (again short) rows. This occurs because the EVENUP command can locally even up the rows, but has trouble deciding which cells to move from the long rows to the short rows without creating an excess of feeds. The best approach is to run the series: D2, EVENUP until no improvement can be made and then manually move some of the cells to evenup the rows.

An alternate and sometimes simpler technique is to find a large cell in a long row which has most of its connections going to the top of

the circuit. Move that cell to one of the first short rows at the top of the chip, fix its position, and run a D2. This will tend to drag the cells connected to that large cell out of the long rows into the short rows at the top in order to delete all the feeds which were created when the large cell was moved. This should be done for each group of short rows. An EVENUP at this point will be much more effective because the cause of long and short rows has been removed. The large cells should then be unfixed from their positions.

10.2.2 CRITICAL PATHS

In some circuits, it may be known beforehand that certain signals are sensitive to load capacitance. In others, these signals will not be known until the routing has been performed. In either case, the filename.CAP file should be periodically examined to see which signals are heavily loaded.

Once the critical paths are known, the TRACE command may be used to see where all the cells are for a given signal, and the appropriate edits made.

10.2.3 OPTIMIZING ROUTING

The OPTIM command will try to produce an optimal routing by exchanging adjacent cells if this results in a shorter total wire length. This technique can be improved through manual placement of cells.

If there are a pair of cells on the left side of the circuit, connected to each other, and each having a single connection to the right side of the circuit, neither will move to the right--each one will hold the other on the left side of the circuit. This is true whether the cells are in the same row or in different rows. Both will have to be manually moved to the right side of the chip.

A cell that is pulled to the right will not be exchanged with its neighbor to the right if the neighbor is pulled as strong or stronger to the right. This can effectively freeze the first cell (and any others to the left of it).

10.3 FEED CELLS

When feed cells are created, ZyPAR takes an average of all the x-coordinants in the row above and the row below the feed rows. This x-coordinate will be used as the location at which the feed is placed for all the feeds for that signal. After the feed is placed, it is treated just like any other cell. It may be moved by a PLACE command, or even fixed in position. Thus, total control is possible over feed thrus.

10.4 MANUAL PLACEMENT

A complete manual placement may be performed by typing the placement of every cell in the circuit into the filename.DMP file, and placing a RELOAD option in the filename.RL2 file. The format of the filename.DMP file is most easily explained by listing one. The row number is given first, then several lines of cell-name and orientation pairs. This is the order of the cells in the row, reading from left to right. An asterisk at the end of the line indicates that there are more cells for the row on the next line. The only cells which need not be entered in the filename.DMP file are the feed cells.

Once the filename.DMP file is created, ZyPAR may be run. It will read the library and filename.NTW file, then re-load from the filename.DMP file. From this initial placement, any of the normal edits may be made.

11 ZyPAR RULES

ZyPAR has two files which contain the rules to be followed for the chip. The first file, filename.RL1, contains the rules which are always true for the chip. The second file, filename.RL2, is used to communicate between edit sessions. Normally, these two files are transparent to the user. Only rarely will changes be necessary, and these should be done by an experienced user.

Some of the rules are TRUE/FALSE options. If the option is specified without a TRUE/FALSE argument, it will be considered to be 'TRUE'. Or, a specific TRUE or FALSE may follow. For example:

```
STRIP

STRIP = TRUE

STRIP FALSE
```

Some options require an argument. These arguments may be an alpha-numeric string, an integer, or a real number. If the option requires a distance (numeric value) argument, they will be specified in ZyPAR units. For the Zy40000 library cells, each ZyPAR unit is one micron. For the Zy50000 series library cells, each ZyPAR unit is one-half micron. The option may be followed by an equal sign, '=', or by a blank. Examples:

```
METAL_WIDTH=6

NROW 5

CD_CELL = XAG1
```

11.1 ALIGN_CELL

Specifies the name of the cell to be used for alignment targets.

e.g: ALIGN_CELL = XAQ1

default: None. ALIGN_CELL must be specified.

11.2 CD_CELL

Specifies the name of the CD (critical dimension) cell.

e.g: CD_CELL = XAR1

default: None. CD_CELL must be specified.

11.3 CLUSTER_AREA

During the global placement algorithm, cells are gathered together into clusters. This parameter specifes the maximum size of a cluster as a fraction of the total active area of the chip. For example, CLUSTER_AREA = .20 would indicate that a cluster should grow no larger than 20% of the total cell area.

e.g: CLUSTER_AREA = .30

default: 0.20

11.4 CMOS

This indicates that the chip uses CMOS technology.

e.g: CMOS False

default: TRUE

11.5 CONTACT1_CELL

Specifies the name of the metal to polysilicon contact cell.

e.g: CONTACT1_CELL = XAK1

default: Blank.

11.6 CONTACT2_CELL

Specifies the name of the metal 2 to Metal 1 contact cell.

e.g: CONTACT2_CELL = XAK2

default: Blank.

11.7 CONTACT_SPACE

Specifies the minimum distance required from the center of one contact to the center of another contact. Distance is given in ZyPAR units.

e.g: CONTACT_SPACE = 20

default: None. CONTACT_SPACE must be specified.

11.8 DEBUG

Specifies that every debug option is to be turned on. This will create excessively large output to the terminal and output files.

e.g: DEBUG

default: FALSE

11.9 DEBUG_OPTIM

Turns debugging diagnostics on during the OPTIM command.

e.g: DEBUG_OPTIM

default: FALSE

11.10 DEBUG_PLACE

Turns debugging diagnostics on during global placement.

e.g: DEBUG_PLACE

default: FALSE

11.11 DEBUG_ROUTE

Turns debugging diagnostics on during routing of the circuit.

e.g: DEBUG_ROUTE

default: FALSE

11.12 DIFF_LAYER

Specifies the Calma GDS layer of the diffusion layer.

e.g: DIFF_LAYER = 12

default: None. DIFF_LAYER must be specified.

11.13 DIFF_OVERLAP

Specifies the amount, in ZyPAR units, that diffusion side connections are set back from the cell border. ZyPAR will create a small rectangle to connect these side connections for telescoping macro cells.

e.g: DIFF_OVERLAP = 3

default: None. DIFF_OVERLAP must be specified.

11.14 FEED_CAP

Specifies the capacitance of a feed thru cell. Units are in picofarads

e.g: FEED_CAP = .4500

default: 0 -- i.e. there is no capacitance.

11.15 FEED_NAME

Specifies the name of the feed thru cell.

e.g: FEED_NAME = XAG1

default: None. FEED_NAME must be specified.

11.16 FRACT_DIST

Specifies the maximum allowed distance between seeded cells while still having them in the same cluster during global placement. The distance is expressed as a fraction of the estimated side dimension of the final chip.

e.g: FRACT_DIST = .15

default: .25

11.17 GRID_SIZE

Specifies the default spacing in ZyPAR units between adjacent pins of a cell.

e.g: GRID_SIZE = 16

default: None. GRID_SIZE must be specified.

11.18 LIB_TYPE

Specifies the type of library. Possible answers are:

Zy4: Zy40000 Silicon gate CMOS

Zy5: Zy50000 Silicon gate HCMOS

e.g: LIBTYPE = Zy4

default: None. LIBTYPE must be specified

11.19 LOGO_CELL

Specifies the name of the Logo cell.

e.g: LOGO_CELL = XAZ1

default: None. LOGO_CELL must be specified.

11.20 METAL_CAP

Specifies the capacitance of metal in picofarads per square ZyPAR unit.

e.g: METAL_CAP = .0014

default: 0.0

11.21 METAL_CONTACT

Specifies the size of the metal square around a contact in ZyPAR units. For example, if a contact is 6 ZyPAR units, and the metal overlap is 2 ZyPAR units, METAL_CONTACT would be 10 ZyPAR units.

e.g: METAL_CONTACT = 10

default: None. METAL_CONTACT must be specified.

11.22 METAL_LAYER

Specifies the Calma GDS layer for metal.

e.g: METAL_LAYER = 7

default: None. METAL_LAYER must be specified

11.23 METAL_SPACE

Specifies the minimum metal to metal spacing in ZyPAR units.

e.g: METAL_SPACE = 6

default: None. METAL_SPACE must be specified.

11.24 METAL_SWIDTH

Specifies the width of metal side connections between telescoping macro cells in ZyPAR units.

e.g: METAL_SWIDTH = 8

default: None. METAL_SWIDTH must be specified.

11.25 METAL_WIDTH

Specifies the mimimum metal width in ZyPAR units.

e.g: METAL_WIDTH = 6

default: None. METAL_WIDTH must be specified.

11.26 METAL2_CAP

Specifies the capacitance of 2nd metal in picofarads per square ZyPAR unit.

e.g: METAL2_CAP = .0014

default: 0.0

11.27 METAL2_LAYER

Specifies the Calma GDS layer for 2nd level metal.

e.g: METAL2_LAYER = 28

default: 0

11.28 METAL2_SPACE

Specifies the minimum metal to metal spacing, in ZyPAR units, on 2nd level metal.

e.g: METAL2_SPACE = 6

default: 0

11.29 METAL2_WIDTH

Specifies the mimimum 2nd level metal width in ZyPAR units.

e.g: METAL2_WIDTH = 8

default: 0

11.30 MIN_CLUSTER

Specifies the mimimum number of clusters to create during global placement. ZyPAR will stop combining clusters when this number has been reached.

e.g: MIN_CLUSTER = 6

default: 10

11.31 NBEST

Specifies the number of best placements to be saved during global placement. Once this number of placements in each category has been reached, a new placement will be saved only if it is better than one of the ones already saved, and the old one will be deleted.

e.g: NBEST = 4

default: 5

11.32 NO_INIT

Specifies that no initial optimization techniques are to be performed during global placement.

e.g: NO_INIT

default: False. i.e. initial optimization will be performed.

11.33 NROW

Specifies the number of rows in the circuit.

e.g: NROW = 12

default: ZyPAR will determine the number of rows depending on the number of grids and the estimated active to routing ratio, RATIO.

11.34 POLY_CAP

Specifies the poly capacitance in picofarads per square ZyPAR unit.

e.g: POLY_CAP = .0012

default: 0.0

11.35 POLY_CONTACT

Specifies the size of the poly around a contact. For example, if the contact is 6 ZyPAR units, and the poly overlap is 1 ZyPAR unit, POLY_CONTACT would be 8 ZyPAR units.

e.g: POLY_CONTACT = 8

default: None. POLY_CONTACT must be specified.

11.36 POLY_LAYER

Specifies the Calma GDS layer for the polysilicon.

e.g: POLY_LAYER = 6

default: None. POLY_LAYER must be specified.

11.37 POLY_OVERLAP

Specifies the amount a poly side connection between adjacent telescoping macro cells must overlap the edge, in ZyPAR units.

e.g: POLY_OVERLAP = 6

default: None. POLY_OVERLAP must be specified.

11.38 POLY_RUN

Specifies the maximum horizontal poly run length in ZyPAR units. This will determine when horizontal runs must be made in metal, even if it is physically possible to make a run in poly under another metal.

e.g: POLY_RUN = 200

default: 0

11.39 POLY_SPACE

Specifies the mimimum poly to poly spacing in ZyPAR units.

e.g: POLY_SPACE = 6

default: None. POLY_SPACE must be specified.

11.40 POLY_SWIDTH

Specifies the width, in ZyPAR units, of poly side connections between adjacent telescoping macro cells.

e.g: POLY_SWDITH = 6

default: None. POLY_SWIDTH must be specified.

11.41 POLY_WIDTH

Specifies the mimimum poly width in ZyPAR units.

e.g: POLY_WIDTH = 6

default: None. POLY_WIDTH must be specified.

11.42 RATIO

Specifies the ratio of cell height to channel routing "width". This value is used by ZyPAR to estimate the chip size.

e.g: RATIO = 1.5

default: 0.5

11.43 RELOAD

Specifies that the circuit is to be reloaded from the filename.DMP file. This command is usually found in the filename.RL2 file, and is automatically generated when an edit session is terminated with a DUMP and EXIT command.

e.g: RELOAD

default: False

11.44 RESTART

Specifies that the circuit is to be RESTARTed from the previous routing. This command is usually found in the filename.RL2 file, and is automatically generated when an edit session is terminated with an EXIT command.

e.g: RESTART

default: False

11.45 RESTORE_LIB

Specifies that the library is to be read from the saved library file, filename.LBS, as opposed to being read from the ZyP library and local library files. Since filename.LBS is a binary file, it will be read much faster. ZyPAR will automatically place a RESTORE_LIB option in the filename.RL2 file at the end of every ZyPAR run except those containing a DUMP command or a QUIT command.

e.g: RESTORE_LIB

default: False

11.46 SCALE

Specifies that the scaling factor for the ZyPAR units to match the GDS units on Calma. For Zy4 library, SCALE = 2. For Zy5 library, SCALE = 1.

e.g: SCALE = 2

default: 2

11.47 SCRIBE_GAP1 THRU SCRIBE_GAP18

Specifies the gap between the active area and the inside edge of each scribe ring in ZyPAR units. There are up to 18 scribe rings which can be defined.

e.g: SCRIBE_GAP1 = 34

default: SCRIBE_GAP1 must be specified for the metal layer. All others are optional.

11.48 SCRIBE_LAY1 THRU SCRIBE_LAY18

Specifies the Calma GDS layer for each scribe ring. Several rings may be on the same layer.

e.g: SCRIBE_LAY1 = 7

default: SCRIBE_LAY1 must be specified for the metal layer. All others are optional.

11.49 SCRIBE_SIZE

Specifies the minimum width, in ZyPAR units, of the scribe area from the edge of the active area to the center of the scribe. This width will be rounded up to the next even mil (after shrink).

e.g: SCRIBE_SIZE = 34

default: None. SCRIBE_SIZE must be specified.

11.50 SCRIBE_WID1 THRU SCRIBE_WID18

Specifies the width, in ZyPAR units, of each scribe ring.

e.g: SCRIBE_WID1 = 15

default: If not specified, and a GDS layer has been specified for this ring, the ring will extend to the center of the scribe.

11.51 SHRINK

Specifies the amount by which the circuit is to be shrunk. This parameter will not cause ZyPAR to shrink any coordinates. It is only used to determine where the center of the scribe should be.

e.g: SHRINK = .86

default: 0.0

11.52 TEKTRONIX

Specifies that ZyPAR is being operated from a Tektronix 4113 or 4115. This enables all the Tektronix commands. Care must be taken if this rule is true when operating ZyPAR from a terminal other than a 4113-- if a Tektronix command is given, the terminal response is undetermined.

e.g: Tektronix

default: False

11.53 VDD_LAYER

Specifies the Calma GDS layer on which VDD layer comes out on the cells.

e.g: VDD_LAYER = 7

default: None. VDD_LAYER must be specified.

11.54 VDD_LOC

Specifies the location at which VDD comes out of the side of cells for which a location is not explicitly given. The location is specified in ZyPAR units from the bottom of the cell.

e.g: VDD_LOC = 220

default: None. VDD_LOC must be specified.

11.55 VSS_LAYER

Specifies the Calma GDS layer on which VSS is connected to the cells.

e.g: VSS_LAYER = 7

default: None. VSS_LAYER must be specified.

11.56 VSS_LOC

Specifies the location at which VSS is to be connected to on cells for which no explicit location has been given. The location is expressed in ZyPAR units from the bottom of the cell.

e.g: VSS_LOC = 12

default: None. VSS_LOC must be specified.

11.57 WHRATIO

Specifies the target width to height ratio of the chip. If the number of rows ("NROW" option) is not specified, the number of rows will be determined from this ratio "WHRATIO" and the total active area in the cell.

e.g: WHRATIO = 1.3

default: 1.0

11.58 ZYPLIB

Specifies that the ZyP artwork library will be used.

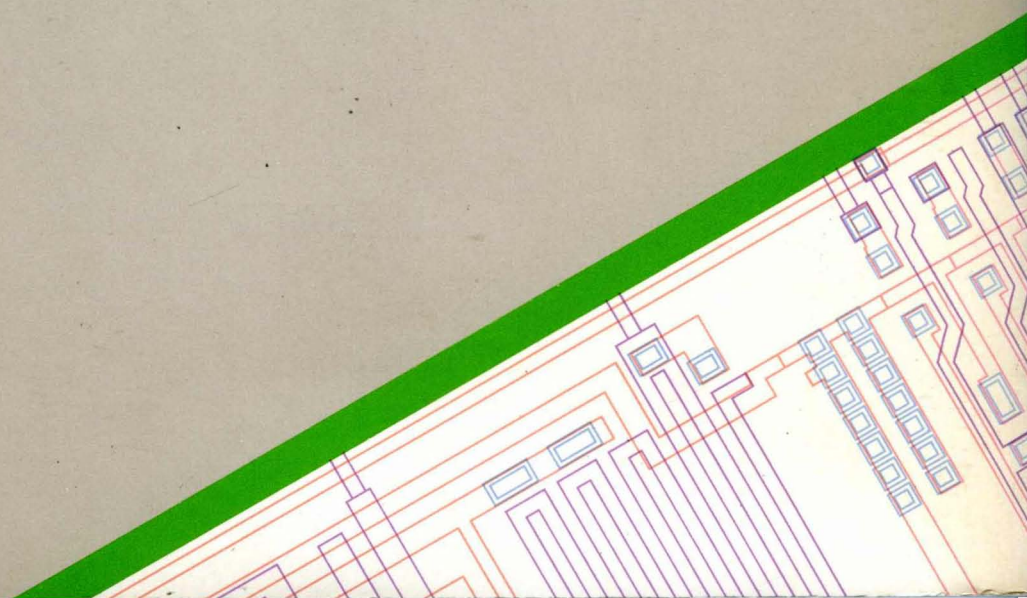
e.g. ZYPLIB

default: TRUE

ZyMOS

PRODUCING THE STANDARD IN CUSTOM VLSI©

477 N. Mathilda Ave. Sunnyvale, California 94086 (408) 730-8800



Zy002-3/85-2K-CAT