



VLSI TECHNOLOGY, INC.

**CELL-BASED
DESIGN
USERS
GUIDE**

ASIC Division

\$15.00

VLSI TECHNOLOGY, INC.

CELL-BASED DESIGN USERS GUIDE

©1988 VLSI Technology, Inc. All rights reserved.
This document and the software that it describes are the proprietary and confidential property of VLSI Technology, Inc. ("VLSI") and Xidak Inc., for distribution and use only under license from VLSI and may not be copied without VLSI's written consent.

VLSI Technology reserves the right to make changes in the contents of this document without notice. VLSI Technology assumes no responsibility for any errors that appear in this document.

Mainsail is a trademark of Xidak, Inc. Bitpad is a trademark of Summagraphics, Inc. VAX and Vax/VMS are trademarks of Digital Equipment Corporation. Unix is a trademark of AT&T Bell Laboratories. EMBOS is a trademark of Elxsi, Inc. ROS is a trademark of Ridge Computers. Aegis is a trademark of Apollo Computer, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subdivision (b) (3) (ii) of the Rights In Technical Data and Computer Software clause at 52.227-7013 (48 CFR 252).

MACH 1000 and MACH Tools are trademarks of Silicon Solutions Corporation. HILO and HILO3 are trademarks of GenRad Incorporated.



CONTENTS

1	Introduction	1
	Purpose of Document	1
	Design Flow	2
	Design Types	2
	System Planning	3
	Design Entry And Simulation	4
	Physical Design	6
	Design Verification	8
	Implementation	9
	How To Use This Guide	9
	Cell-Based Elements	10
	Standard Cell Library	10
	Compiler Cell Library	11
	Megacell Library	12
	Datapath Element Library	14
	State Machine Elements	15

2	Design Types	17
	Introduction	17
	Turnkey Design	18
	User Logic Design	21
	User Design	23
	Joint Design	26
	Cost Considerations	29
	Design Reviews	29
3	System Planning	31
	Introduction	31
	System Partitioning	32
	Chip Economics	32
	The Die Size	32
	The Package	33
	Power Calculations	34
	Background Information	34
	Causes of Power Dissipation	34
	Completing the Power Calculations Worksheet	36
	Complete the Circuit Data Section	38
	Calculate the Internal Power Dissipation (Pint)	38
	Calculate the External Power Dissipation (Pext)	39
	Calculate the DC Power Dissipation (Pdc)	39

	Calculate the Total Power Dissipation (P _{tot})	39
	Write In The Package Type	39
	Write The Thermal Impedance	39
	Calculate the Junction Temperature (T _j)	40
	Partitioning Methods	40
	Random Logic Partitioning	40
	Building Block Partitioning	40
	Subsystem Partitioning	41
	Integration	41
	Chip Specification Package	42
	1.0 Device Description	43
	2.0 Electrical Specifications	44
	3.0 Test Features And Description	47
	New Design Information Form	48
	Preliminary Design Review	48
	Test Plan	49
	Statement Of Work	49
4	Design For Testability	51
	Designing For Testability	51
	General Guidelines	53
	Additional Circuitry vs. Long Simulations	53
	Translating A Board	53

	Provide Controllability	54
	Provide Observability	55
	Circuit Initialization	56
	Test Modes	57
	Use The Design-For-Testability Guidelines	57
5	Design Entry	59
	Creating Functional Blocks	59
	With Standard Cells	60
	With Compiler Cells	61
	With Megacells	62
	With Datapath Elements	62
	With State Machine Elements	64
	Schematic Entry	65
	Logic	65
	Borders	65
	Node Names	65
	Busses	65
	Signal Drive	66
	Clocks and Clock Buffers	66
	Metal Migration, Current Density and Parallel Buffers	67
	Parallel Logic	68
	Bus Repeater Cells For Three-State Nodes	69

	Weighting	69
	Weights on Clock Buffers	71
	Buffers	72
	Simulation And Test	72
	Pads	73
	Specifying Pad Placement in your Schematic	73
	Functional Blocks	74
	The Top Level Design	75
	Turnkey Design Markups	77
	Design Screen And Review	78
	Netlist Screening	78
	Design Review Program	80
	Pad Placement Form	86
	Bonding Diagram	86
6	Simulation	89
	Netlists For Timing Verification and Simulation	89
	Standard Cells, Compiler Cells and Megacells	89
	State Machine Elements	90
	Datapath Elements	91
	Flattening The Netlist	93
	Timing Verification	93
	Simulation	93

	Netlists	94
	Logic Simulation	95
	External Capacitances	95
	Asynchronous Logic	95
	Simulation Checks	95
	Timing Simulation	97
	Estimate Routing Capacitance	97
	Provide Output Load Capacitance	98
	Bidirectional Pins	98
	Worst Case Timing Can Hide Best Case Hazards	98
	Interconnect Capacitance Calculations	99
	Simulator	99
	Design Review Program	99
	Netlist Extractor	100
	Logic Compiler	101
	Fault Simulation	101
	Logic Design Review	102
7	Physical Design	103
	Chip Compiler	103
	Using The Chip Compiler	103
	Before You Begin	105
	Regenerate Your Netlists	105
	Check V6 Blocks	105
	Create Phantoms	106

	Flatten The Netlist	107
	Compile Your Cell	108
	Basic Operations	108
	Convert Your Layout	109
	Standard Cell Areas	109
	Large Chips	110
	Fixing UIs	110
	In The Composition Editor	110
	In The Layout Editor	111
8	Post-Physical Design Verification	113
	Back Annotation	113
	Post-Physical Design Simulation	114
	Complete The Physical Design	114
	Phantom DRC	116
	Phantom Netcompare	116
	Generate 1 MHz Test Vectors	117
	Super- Synchronous Simulation	
	Guidelines	120
	Test Vector Guideline Summary	120
	Checking Vectors	122
	How to Check Vectors	122
	Example	123
	Test Specification Form	123
	Prototype And Production Test	
	Specification Form	123

	Additional Guidelines For Test Program Generation	124
	Phantom Tape Out	124
	Final Design Review	125
9	Implementation	127
	Generate Test Program	127
	Merge Cell Layouts	128
	Physical Design Verification	128
	Design Rule Check	128
	Extract And Netcompare	128
	Visual Check	130
	Simulation	130
	CIF Tape Out	131
	Customer Package Marking Form	131
	CIF Tape Out	131
	Archive The Design	131
	Final Check	132
	Mask Generation	132
	Prototypes	132
A	VLSI Design Tools	133
B	Package Pin Inductances	143
C	Foundry Cells	147

D	Recommended Archive Documentation	155
Index	Index	158



CHAPTER 1

INTRODUCTION

This chapter describes the typical circuit elements and design tasks encountered in cell-based design using VLSI Technology's design tools and libraries.

Purpose of Document

This users guide describes the methodology for designing with:

- Portable library standard cells
- Compiler cells
- Megacells
- Datapath elements
- State machine elements

using VLSI Technology's software tools. Using the VLSI design tool system, these elements may all be combined on a single chip. By replacing LSI components with compiled cells, megacells, and datapaths implemented as compiled layout or

standard cells, and the remaining glue logic with state machine elements implemented as standard cells, a large system design can be reduced to a single chip.

For further information on using the software tools described in this guide, please refer to the corresponding VLSI tools manuals. A summary of the tools and the functions they perform can be found in Appendix A.

Design Flow

Design Types

VLSI offers four ways to design cell-based chips that allow you to participate as much or as little as you wish in the design cycle:

- **Turnkey** - You provide a VLSI Technology Center with your specifications and test vectors, from which the Technology Center engineers will design your chip.
- **User Logic** - You design the logic and turn it over to the Technology Center for physical implementation.
- **User** - You create both the logic and the physical design.
- **Joint** - You participate jointly with the Technology Center in every step of the development.

In all cases, it is the customer's responsibility to define simulation and test patterns.

Whether you choose a turnkey design or fuller participation, the general flow of the design cycle follows the stages outlined in this chapter.

System Planning

The first step in creating a cell-based design is to generate specifications and a system plan for your design (Figure 1). VLSI's Design Assistant tool can help you decide between alternatives at this stage.

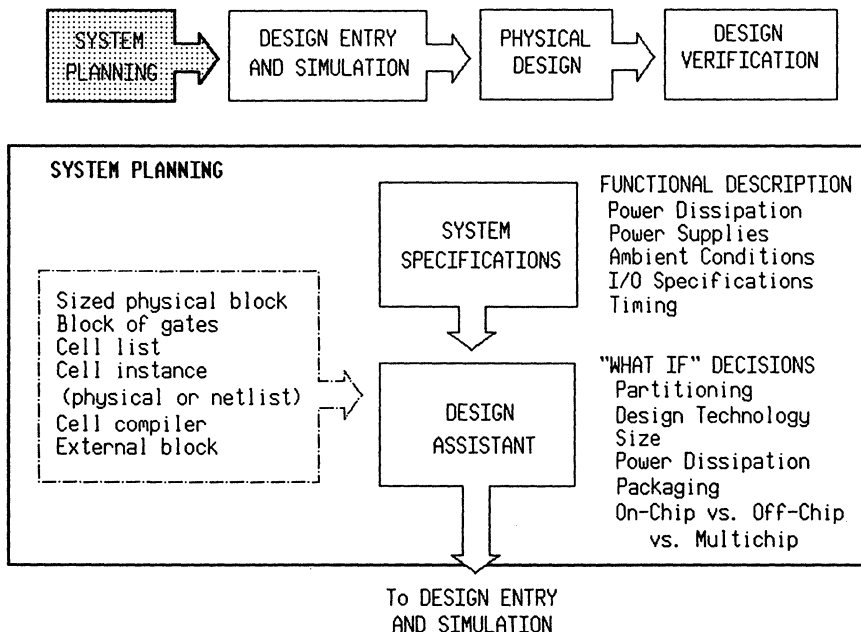


Figure 1. System Planning Flow

Design Entry And Simulation

The design logic is entered using the VLSI schematic editor (Figure 2). Some functional blocks, like compiler cells and state machine elements, need to be generated in the cell library window, VTICellLib, and then placed in the schematic.

The completed schematic is checked with the VLSI timing verifier and simulated using the mixed mode simulator, VTIsim. Test vectors are generated from the output of the simulator. Screen and review checks are performed where applicable, and optional fault simulation is available.

When the schematic is complete, there is a joint Customer/VLSI logic design review. If the design is acceptable, it goes on to physical layout design.

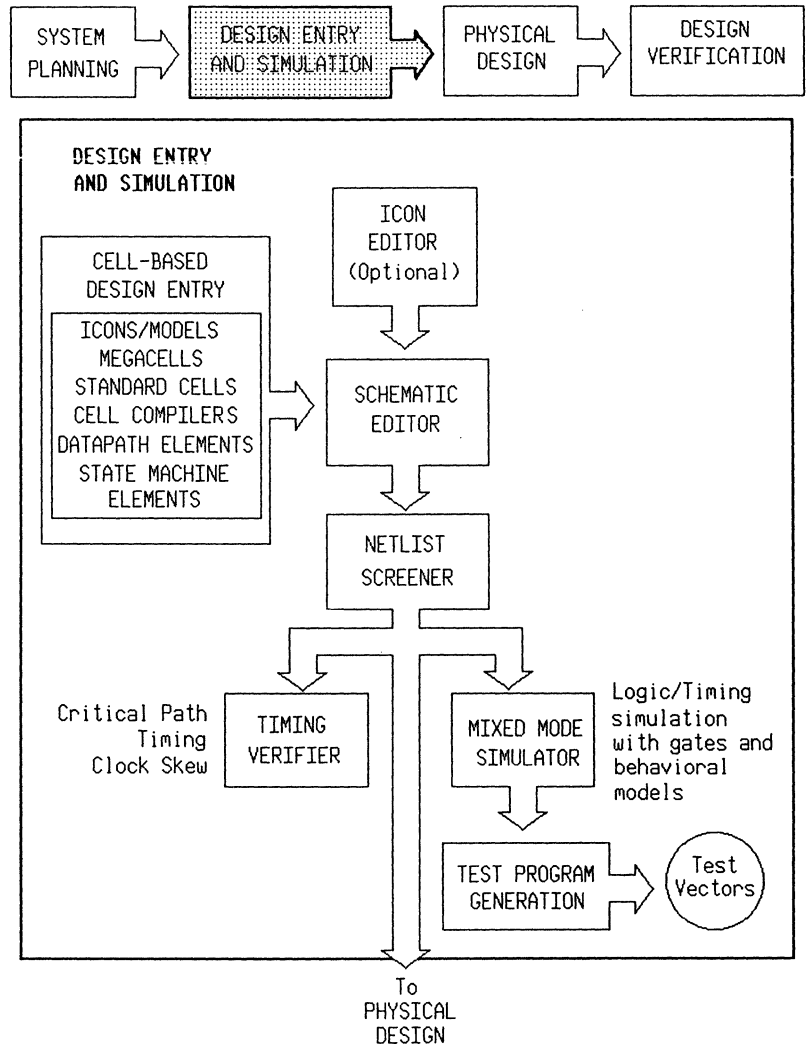


Figure 2. Design Entry And Simulation Flow

Physical Design

Compiled layout can be created using the cell library window, the composition editor or the layout editor. These arbitrary blocks, and any standard cell functional blocks, are represented as phantoms -- black boxes with connectors. These phantoms are assembled using VLSI's chip compiler. Placement and routing can be automatic, interactive, or a combination of both (Figure 3).

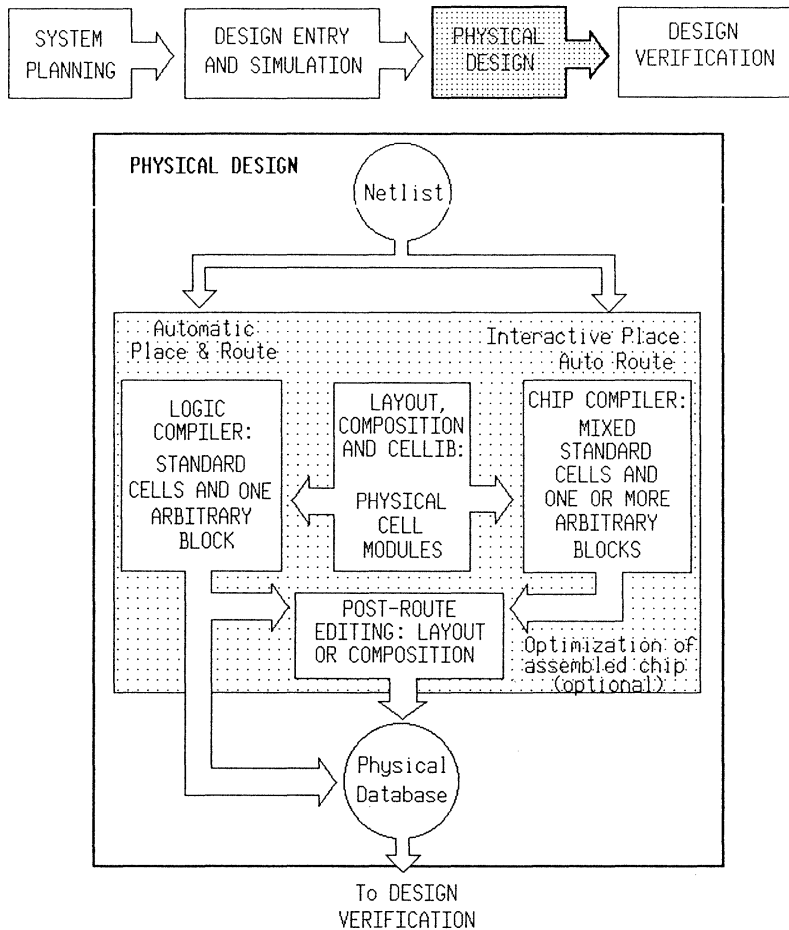


Figure 3. Physical Design Flow

Design Verification

A netlist extracted from the completed phantom physical database is compared to the schematic netlist using the netlist comparison program, to assure network consistency. The netlist is back-annotated with actual wiring capacitances and resimulated with VTIsim. Finally, a DRC verifies that the design conforms to physical design rules (Figure 4).

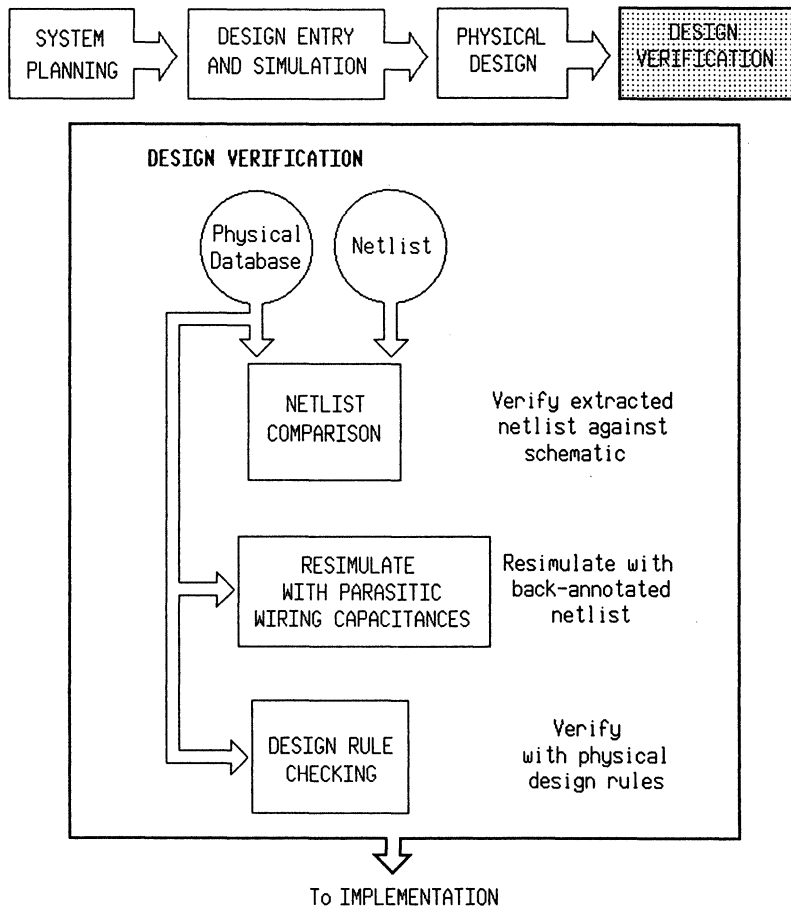


Figure 4. Design Verification Flow

When the layout is completed and verified, there is a joint Customer/VLSI final design review.

Implementation

If the finished design is approved, the phantoms are replaced with actual layout and the design is sent for full physical design verification, mask generation and prototype build. When the prototype is approved, the design can be transferred from pre-production status to full production.

How To Use This Guide

Chapter 2, **Design Types**, specifies which tasks are performed by the customer and which are performed by the VLSI Technology Center for each design type: Turnkey, User Logic, User, and Joint. When you have decided upon a design type, check the flowchart for your design type in Chapter 2. The blocks on the **User** side of the chart list the tasks for which you are responsible. In Chapters 3 through 9, read the explanation of those tasks that apply to your design type, and ignore those that do not. The chapters that explain the design cycle tasks are:

- Chapter 3 - System Planning
- Chapter 4 - Design For Testability
- Chapter 5 - Design Entry
- Chapter 6 - Simulation
- Chapter 7 - Physical Design
- Chapter 8 - Post-Physical Design Verification
- Chapter 9 - Implementation

Cell-Based Elements

This section describes the types of circuit elements and functional blocks that you can use in a VLSI cell-based design: standard cells, compiled cells, datapath elements, and state machine elements.

Standard Cell Library

The standard cell portable library offers a wide range of functions, including gates, buffers, flip-flops, multiplexers, decoders, adder/subtractors, synchronous counters, latches and various I/O functions. These cells are available in the 2 micron VSC10 series and the 1.5 micron VSC100 series; refer to the appropriate library manual for more information. The base library for each series contains a set of standard functions, and an extended library provides additional functionality with clock buffers, flip-flops, latches, synchronous counters, I/O pads, buffers, and crystal oscillators. The portable netlist obtained from a design can be implemented either as standard cells or in a gate array.

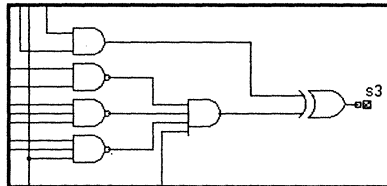


Figure 5. Standard Cells

Compiler Cell Library

The VCC compiler cell series consists of:

- RAM
- 2901 expandable datapath (2 micron only)
- Multiplier
- PLA
- ROM

These cells are available in the 2 micron VCC10 series and the 1.5 micron VCC100 series; refer to the appropriate library manual for more information. You can select from parameters such as speed, drive and number of bits to configure each compiler cell into the version you need to suit your design application.

NOTE: If you use these cells, you cannot implement your design as a gate array.

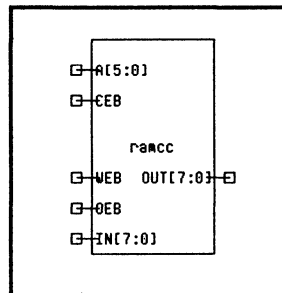


Figure 6. Compiler Cell

Megacell Library

Megacells are the building block equivalents of standard microprocessor peripherals. VLSI offers these megacells:

- M68C45 CRT controller
- M82C37 programmable DMA controller
- M82C50 asynchronous communications element
- M82C54 programmable interval timer/counter
- M82C59 programmable interrupt controller
- M82C84 clock generator/driver (2 micron only)
- M82C88 bus controller (2 micron only)
- M84C00 Z80 CPU
- M84C30 counter/timer circuit
- M84C40 serial I/O controller
- M85C35 serial communications controller

Most of these cells are available in both the 2 micron VMC10 series and the 1.5 micron VMC100 series; refer to the appropriate datasheet for more information. Beginning with V7R3, all megacell datasheets will be placed in one **Megacell Library Manual** binder.

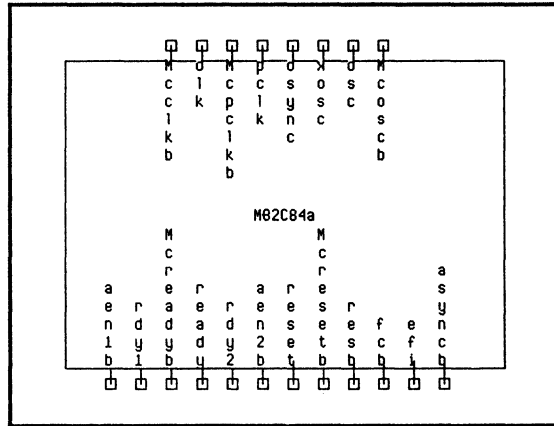


Figure 7. Megacell

State Machine Elements

A state machine is a block of logic whose outputs depend on its inputs and its internal state. Most cell-based logic is created by using cells; however, a state machine element is created from a text description. Specifications for the state machine are written in an easy-to-use high-level language that is based on Boolean equations as input. The portable netlist can be implemented either as standard cells or in a gate array. You can also generate a compiled PLA block from a state machine description.

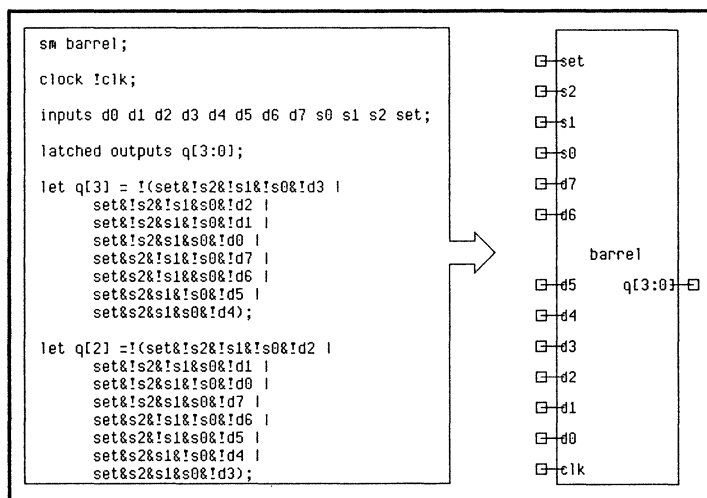


Figure 9. State Machine



CHAPTER 2

DESIGN TYPES

This chapter describes the design tasks performed by the user and VLSI for each cell-based design type.

Introduction

There are four basic types of design, depending upon how the design tasks are divided between VLSI Technology and the customer: turnkey, user logic, user, and joint.

When you have decided upon a design type, check the flowchart for your design type in the pages that follow. The blocks on the **User** side of the chart list the tasks for which you are responsible. In Chapters 3 through 9, read the explanation of those tasks that apply to your design type, and ignore those that do not.

The chapters that explain the design cycle tasks are:

- Chapter 3 - System Planning
- Chapter 4 - Design For Testability
- Chapter 5 - Design Entry
- Chapter 6 - Simulation
- Chapter 7 - Physical Design
- Chapter 8 - Post-Physical Design Verification
- Chapter 9 - Implementation

Turnkey Design

For a turnkey design, the customer supplies design specifications, logic diagrams and test vectors. VLSI is responsible for all phases of design, verification and test program generation from the vectors provided (Figure 10).

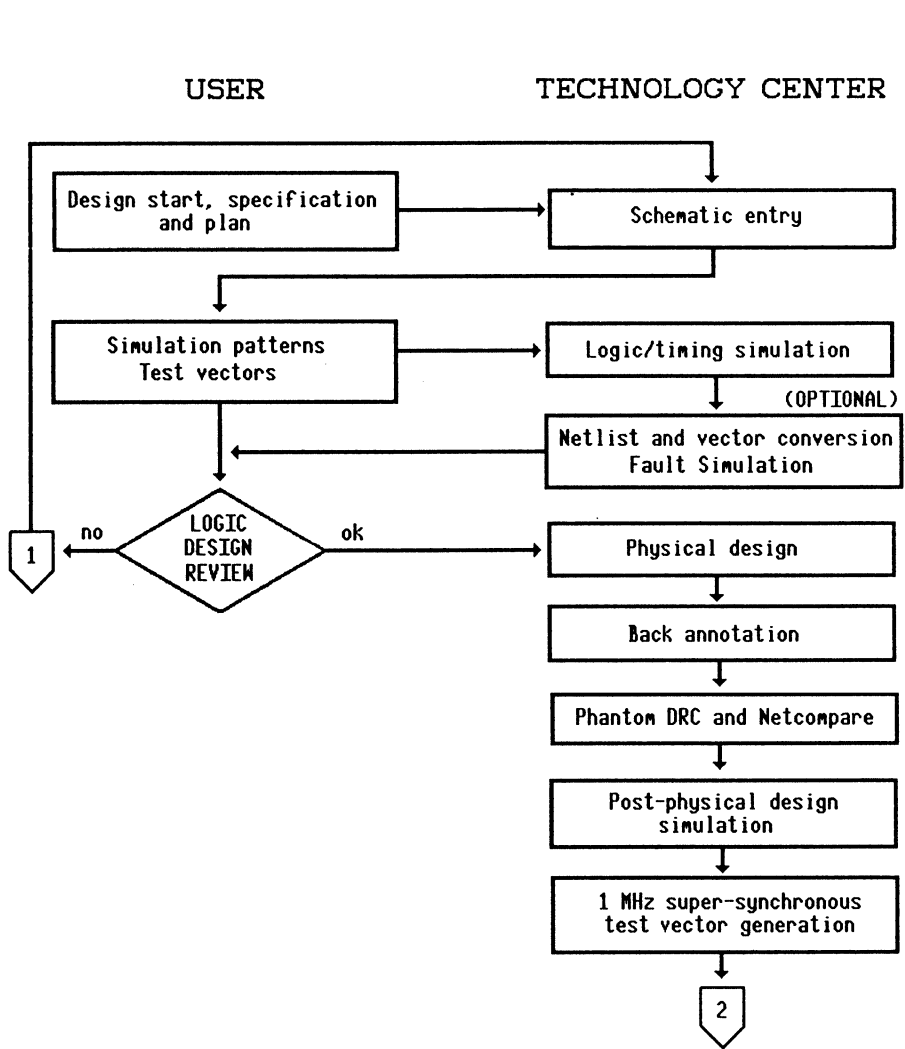


Figure 10. Turnkey Design

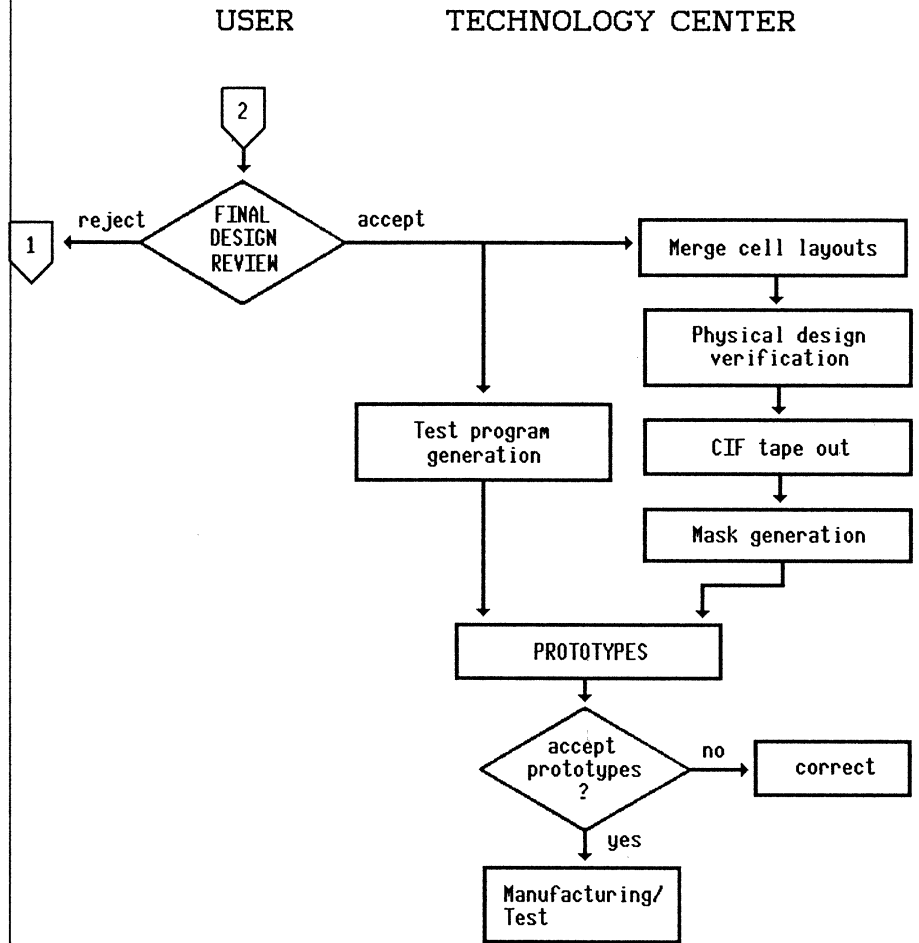


Figure 10. Turnkey Design (Cont'd)

User Logic Design

For a user logic design, the customer is responsible for schematic entry, logic/timing simulation, vector generation, 1 MHz super-synchronous test vector generation, and post-physical design simulation. VLSI is responsible for netlist and vector conversion, logic verification, and the physical design and verification tasks (Figure 11).

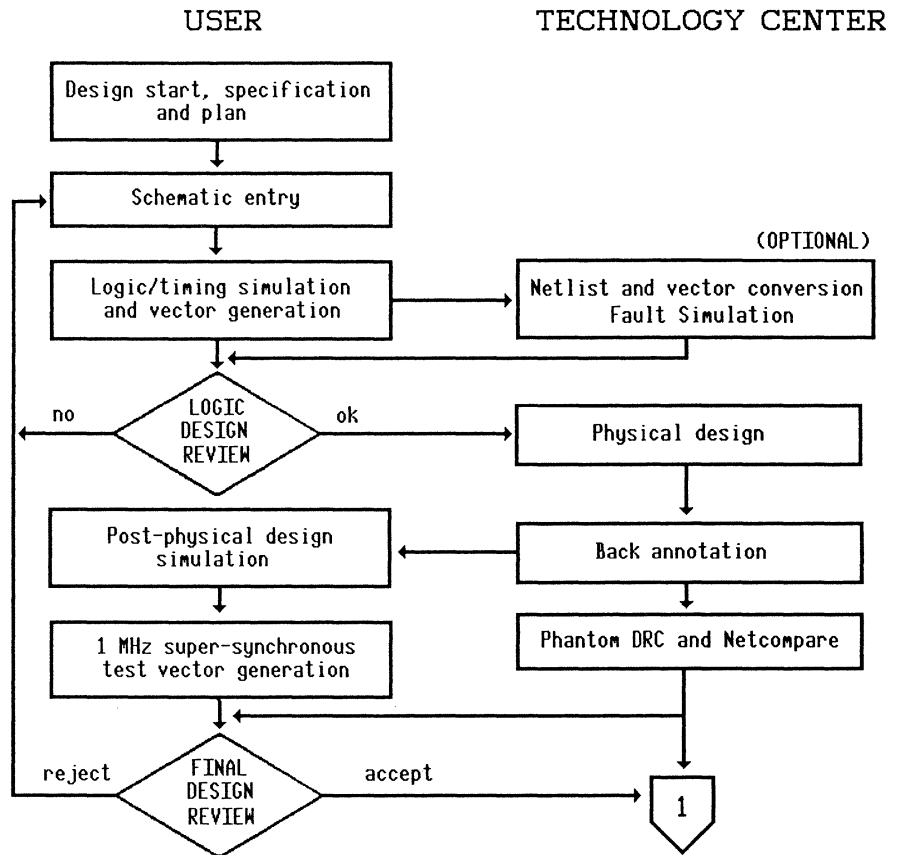


Figure 11. User Logic Design

USER

TECHNOLOGY CENTER

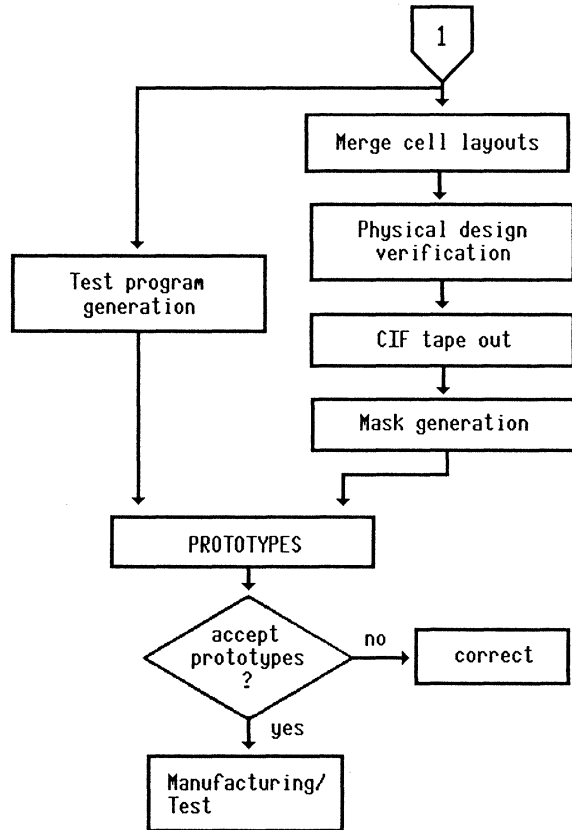


Figure 11. User Logic Design (Cont'd)

**User
Design**

For a user design, the customer is responsible for schematic entry, logic/timing simulation, vector generation, 1 MHz super-synchronous vector generation, physical design, phantom design verification (design rule check and netlist comparison), back-annotation and post-physical design simulation. VLSI is responsible for merging cell layouts, physical design verification, test program generation and CIF tape out (Figure 12).

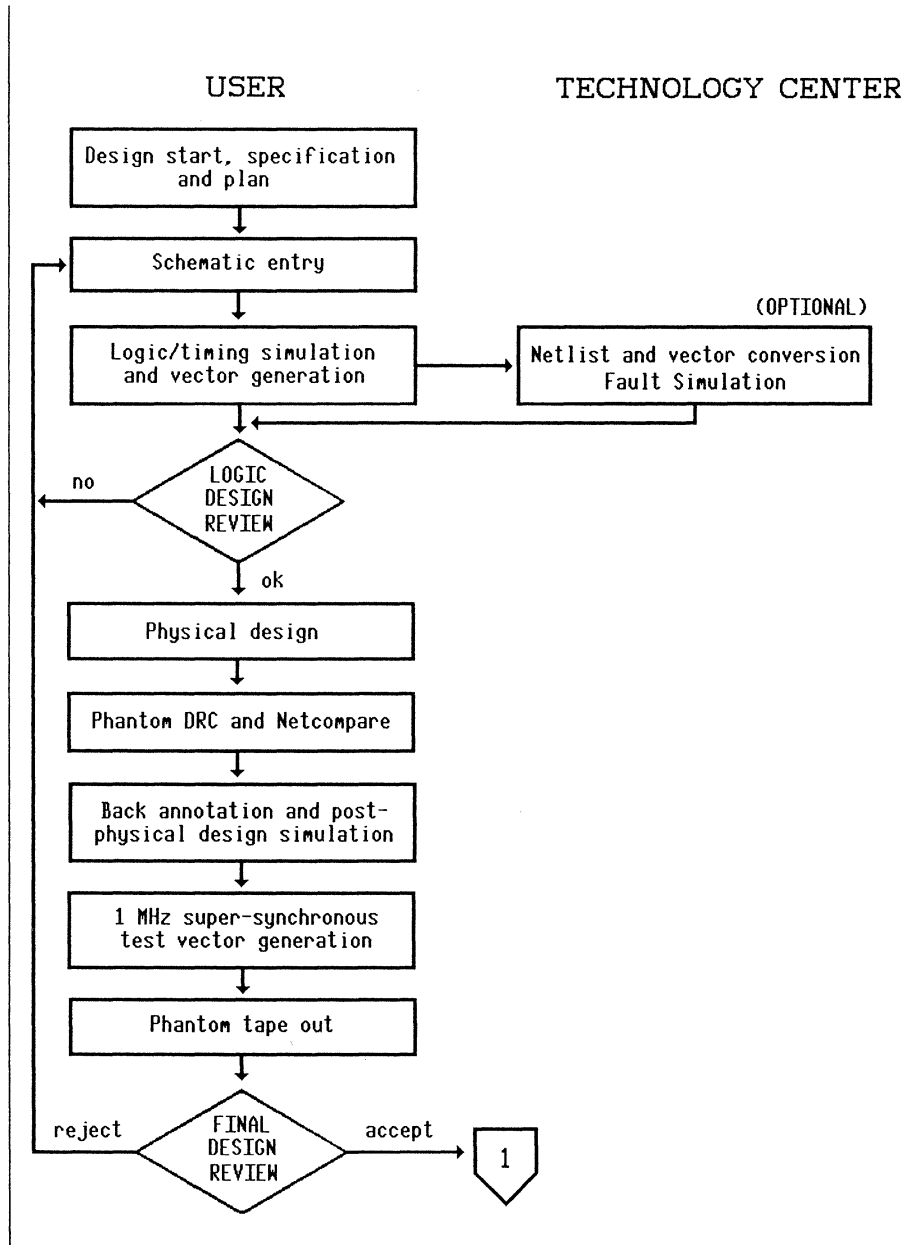


Figure 12. User Design

USER

TECHNOLOGY CENTER

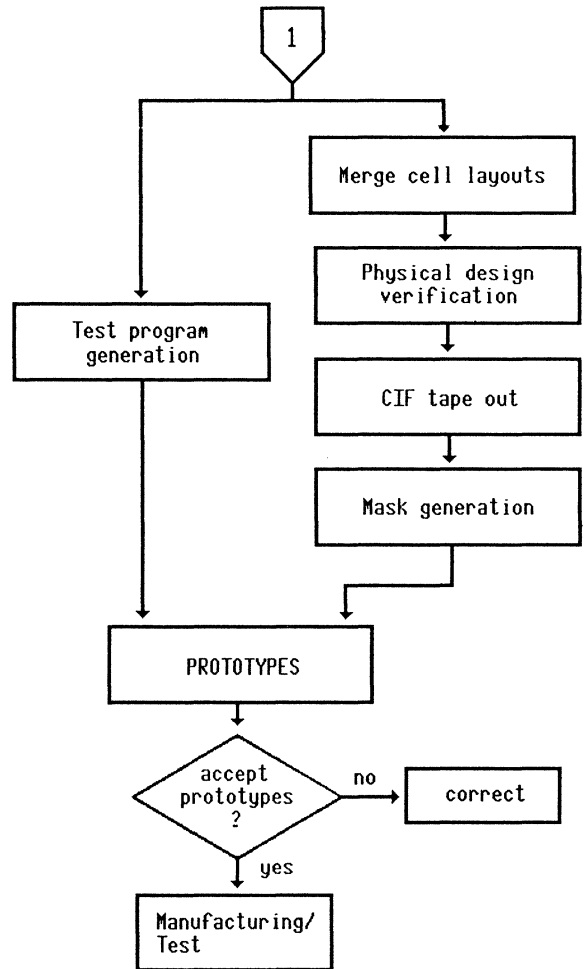


Figure 12. User Design (Cont'd)

Joint Design

For a joint design, the customer and VLSI work together on all phases of the design and share the responsibility for its success (Figure 13).

USER and TECHNOLOGY CENTER

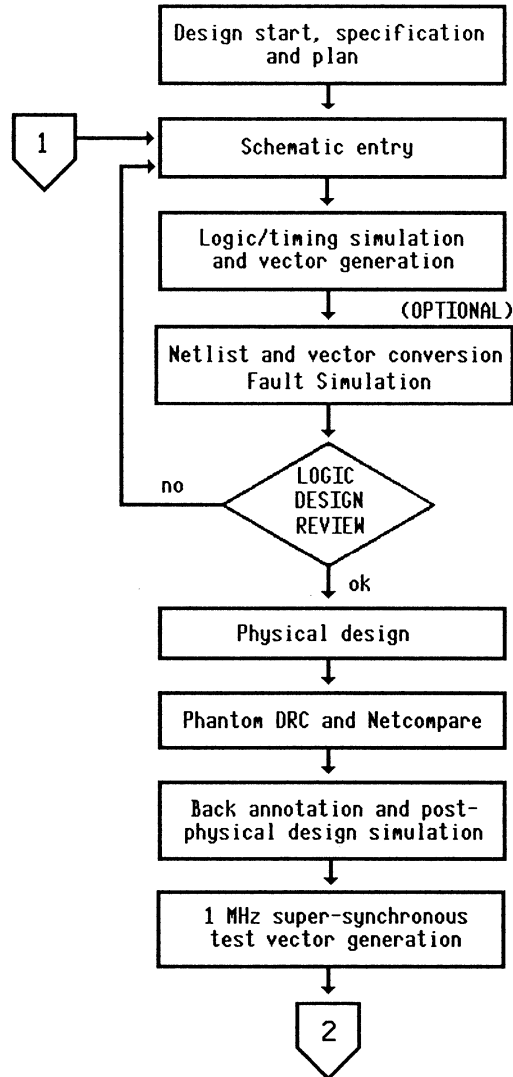


Figure 13. Joint Design

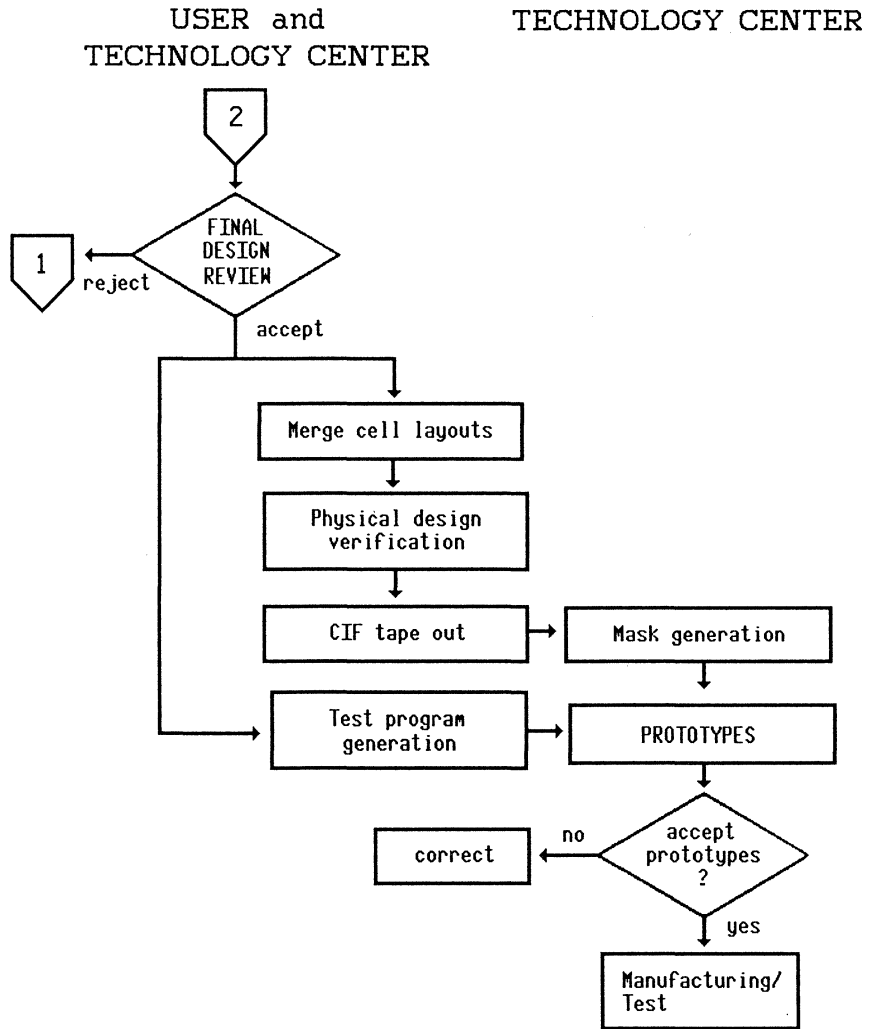


Figure 13. Joint Design (Cont'd)

Cost Considerations

As shown in Figure 14, the cost of the design goes down as the amount of user involvement increases:

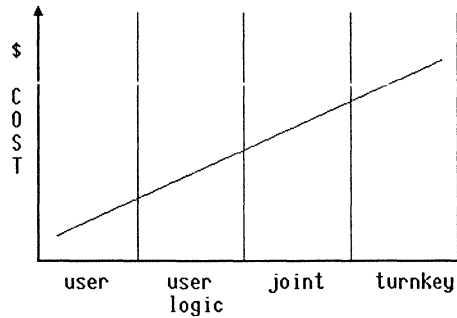


Figure 14. Cost vs. User Involvement

Design Reviews

In all cases, VLSI and the customer conduct two design reviews:

- Logic Design Review - after Logic/Timing Simulation
- Final Design Review - after Post-Physical Design Simulation

Both of the design reviews include careful investigation of the chip specification, design methodology, post-physical verification, critical paths, testability and packaging requirements.

CHAPTER 3

SYSTEM PLANNING

This chapter describes the design start, specification and planning task, including system partitioning, chip economics, and power calculations.

Introduction

ASIC designs are becoming increasingly complex, and it is now possible to integrate a complete digital system onto a single chip. The design engineer has to find the best solution for subdividing such a system into parts that can be integrated most efficiently. A complete system can be a one-chip ASIC solution, or various ASIC chips of different technologies mixed with standard parts.

System Partitioning

How you partition your system can depend upon:

- Total system cost
- Total and partial system performance, such as speed and power consumption
- Total system reliability, including logic design and packaging

The various approaches to system partitioning are determined by one or another of these objectives. Before subdividing a system, you need to consider the semiconductor economics, VLSI implementation approaches, and partitioning methods appropriate to your design.

Chip Economics

The Die Size

There is always a certain defect density on a wafer; the probability of having a defect increases with increasing die size. Reducing die size is the most important means of obtaining economic chip production, and the designer must handle this efficiently.

The design of a cell-based chip can directly influence its die size. It depends upon:

- The complexity of the logic: the number of gates
- The regularity and the amount of interconnect: the bus structure
- The number and size of I/O pads on the chip
- Performance requirements

The result can be either a pad-limited layout, where the size of the die is dictated by the padding, or a core-limited layout, where the die size is dictated by the core dimensions (Figure 15). Each of these layout types requires a corresponding I/O pad set. The VLSI chip compiler automatically selects the appropriate pads for your design layout.

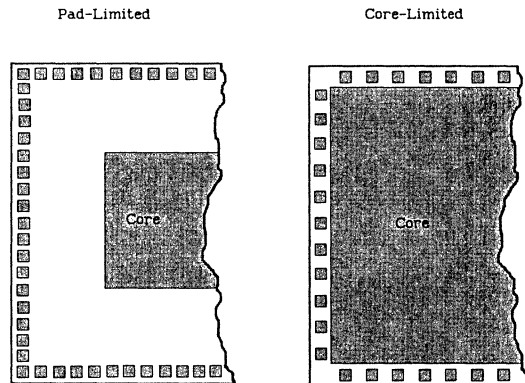


Figure 15. Core-Limited and Pad-Limited Chips

The Package

The package also contributes to the total device cost. The package type selected depends on the number of pins required, die size and estimated power dissipation. There are different lead frames for the same package; one must be chosen to match the die size and to ensure that the bonding rules are fulfilled. Using a specific package can also affect other objectives, such as cost, reliability, and printed circuit board mounting techniques. Prototype chips are normally delivered in ceramic packages, for shorter assembly times, while subsequent production devices may be assembled in less expensive plastic packages.

The semiconductor packages that VLSI Technology offers are listed in the **Semiconductor Package Selection Guide**.

This document gives the materials, dimensions, pin count, package width, VLSI bond form drawing number, pad size, minimum and maximum die sizes, thermal resistance, and other necessary statistics for each available package. VLSI is continually qualifying new sizes and types of packages; consult your Technology Center representative for package availability.

Power Calculations

Background Information

When you complete the **Power Calculations Worksheet** according to the instructions at the end of this section, you will be using typical power dissipation figures supplied by VLSI. These figures are based on the considerations discussed in this section.

Causes of Power Dissipation

The amount of heat generated within the silicon chip, known as power dissipation, is low in CMOS technology, compared to other technologies. Power dissipation causes temperature to rise, increasing a circuit's propagation delay. The three causes of power dissipation in CMOS technology are:

1. The charging and discharging of the internal capacitance of a circuit. Known as *AC power dissipation*, the charging and discharging -- switching -- of circuit capacitance is responsible for more than 90% of a circuit's total power dissipation. The power dissipation in a CMOS circuit is essentially a function of the frequency of the logic switching. The charging of a capacitor (C) to a voltage (V) through a P-channel device builds up a charge (CV) and stores energy (CV*V). This energy is later discharged through the

N-channel device which is paired with the P-channel device. When such switching takes place at a frequency (F), the resulting power dissipation can be expressed as $P=FCV^2$, where P is power dissipation.

2. DC current. There are two types of DC power consumption: static DC current, which flows through ON transistors; and leakage DC current, which continues to flow when transistors are OFF. DC current can be significant for output drivers if an external load current is present. You need to set up a test condition in which all static DC current can be turned off, so DC leakage (static I_{dd}) can be measured easily.
3. Transient currents. Transient currents occur when the P- and N- transistors switch from the HIGH to LOW state, or vice versa, in the period when:

$$V_{TH}(N) > V_{IN} < V_{DD}-V_{TH}(P)$$

where V_{TH} = input threshold voltage, N = N-transistor, V_{IN} = input, V_{DD} = supply voltage, and P = P-transistor. Transient currents are responsible for less than 10% of the total power dissipation.

The total power dissipation of a cell-based design can be estimated by adding the estimated power dissipation for the macros you have used in your design, and multiplying by a correction factor to account for the percentage of gates that switch simultaneously. Statistically, 0.20 (20%) has been found to be a useful correction factor, reflecting the percentage of simultaneously switching gates commonly found in cell-based designs, although typical applications vary from near zero to 50%. Your own estimate may be somewhat higher or lower, depending on the characteristics of your design.

The power dissipation for each VSC10 or VSC100 macro is given in the appropriate library manual. The maximum allowable power dissipation, of course, depends on the package and the cooling system used.

Another method of calculating a circuit's power dissipation is to use the Design Assistant, which supplies an estimate of the total power consumption in mW.

Completing the Power Calculations Worksheet

The **Power Calculations Worksheet**, illustrated in Figure 16, helps you calculate your circuit's junction temperature, which is the temperature of the die inside the package. It is also used to arrive at certain values required on the **AC/DC Specifications Form**. VLSI encourages you to include a copy of the worksheet in the initial signoff review, although it is not required. The information on the **Power Calculations Worksheet** is necessary for package selection; please contact the Technology Center about package feasibility if you omit this form.

Completing the **Power Calculations Worksheet** requires the following steps, described in detail in the text that follows:

- Complete the circuit data section.
- Calculate the Register Percentage (R).
- Calculate the Internal Power Dissipation (Pint).
- Calculate the External Power Dissipation (Pext).
- Calculate the Total Power Dissipation (Ptot).
- Calculate the Junction Temperature (Tj).

POWER CALCULATIONS WORKSHEET

DESIGN NAME _____

VLSI TECHNOLOGY PART NUMBER _____

MILLIWATTS/GATE $P = 0.020 \text{ mW/MHZ/GATE}$ AVG. OPERATING FREQUENCY (F) $F = \text{_____ MHZ}$ EST. FRACTION OF GATES SWITCHING
SIMULTANEOUSLY (TYPICALLY 0.20) $S = \text{_____}$ AMBIENT OPERATING TEMPERATURE $T_a = \text{_____ C}$ NUMBER OF GATES $G = \text{_____}$ NUMBER OF OUTPUT PINS $B = \text{_____}$ AVG. OUTPUT LOAD CAPACITANCE $C = \text{_____ PF}$

INTERNAL POWER DISSIPATION

 $P_{int} = P * F * S * G$ $P_{int} = \text{_____ mW}$

EXTERNAL POWER DISSIPATION

 $P_{ext} = 0.035 * F * B * .2 * C$ $P_{ext} = \text{_____ mW}$

DC OUTPUT POWER DISSIPATION SUM

 $P_{dc} = \text{_____ mW}$

TOTAL POWER DISSIPATION

 $P_{tot} = 0.001 * (P_{int} + P_{ext} + P_{dc})$ $P_{tot} = \text{_____ W}$

PACKAGE TYPE (INCL. # PINS) _____

THETA JA OF PACKAGE _____

C/W

JUNCTION TEMPERATURE

 $T_j = (P_{tot} * \text{THETA JA}) + T_a$ $T_j = \text{_____ C}$

REPORT GENERATED BY: _____

Figure 16. Power Calculations Worksheet

Complete the Circuit Data Section

Complete each part of the circuit data section as follows:

- **MILLIWATTS/GATE (P)**. Typical power dissipation in milliwatts/MHz/gate. The typical power dissipation for VLSI's VSC cells is provided in the datasheets in the **VSC10 Macro Library** and **VSC100 Macro Library** manuals.
- **AVERAGE OPERATING FREQUENCY (F)**. Write the circuit's operating frequency in megahertz in the blank.
- **AMBIENT OPERATING TEMPERATURE (Ta)**. Write the circuit's maximum ambient operating temperature in degrees centigrade in the blank.
- **NUMBER OF GATES (G)**. Write the number of gates required by the circuit in the blank.
- **NUMBER OF OUTPUT PINS (B)**. Write the number of external outputs the circuit contains in the blank.
- **AVERAGE OUTPUT LOAD CAPACITANCE (C)**. Estimate the output load capacitance using the input capacitance specifications for the interfacing chips and the interconnect capacitance. Write your estimate of the average, in picofarads, in the blank.

Calculate the Internal Power Dissipation (Pint)

Using the values you noted in the circuit data section, and the Register Percentage you calculated, solve the equation shown and write the result in the blank.

Calculate the External Power Dissipation (P_{ext})

Using the values you noted in the circuit data section, solve the equation shown, and write the result in the blank.

Calculate the DC Power Dissipation (P_{dc})

Add the individual DC currents of any output pads that drive DC loads and write the result in the blank.

Calculate the Total Power Dissipation (P_{tot})

Add the Internal Power Dissipation, External Power Dissipation and DC Power Dissipation and multiply the sum by .001. The result is the circuit's Total Power Dissipation in watts. Write the results in the blank.

Write In The Package Type

Write the type of package you have chosen, including the number of pins, in the blank provided.

Write The Thermal Impedance

Write the thermal impedance (Θ_{JA}) of the package you have chosen in the blank provided. This value is given in the **Semiconductor Package Selection Guide**, which you can obtain from the Technology Center.

The junction-to-ambient (JA) thermal resistance data is based on a 10,000 square mil die, with the board mounted in still air. The thermal resistance varies with the materials used, die size, process technology, air circulation, and heat dissipation characteristics of the device. Values listed in the package selection guide are meant to serve as guidelines and are believed to be on the high side. For larger dice, the values are typically lower.

**Partitioning
Methods****Calculate the Junction Temperature (T_j)**

Multiply the Total Power Dissipation by the thermal impedance (Θ_{ja}) of the package you have chosen. When the thermal impedance is given as a range, use the highest value in the range. Add the result to the Ambient Operating Temperature (T_a) you noted in the circuit data section. Write the sum in the blank. This is the estimated junction temperature in your circuit. The recommended limit is 150 degrees C.

Your implementation approach determines the partitioning method that you apply to a system: random logic partitioning, building block partitioning, or subsystem partitioning.

Random Logic Partitioning

In a total system that is built with standard parts of different complexities, the MSI and SSI parts normally form the glue logic between the LSI and VLSI parts. Random Logic Partitioning consists of putting these random logic parts into one or several ASIC chips. The total resulting chip complexity is low, and the pin count is high. A system partitioned this way, however, is not the most cost-efficient and space-saving solution, because the total IC count remains high. However, the design turnaround time is short compared to more complex solutions, especially if a gate array is used.

Building Block Partitioning

The next step towards a higher level of integration is to subdivide the system into separate functional building blocks. The glue logic remains implemented externally as standard parts or as other ASIC chips, but the main functional part of the system is implemented as an ASIC.

This approach improves cost efficiency due to a more complex design and a better gate-to-pin ratio. But the biggest advantage arises if the ASIC for the main functional part can be used as a general purpose proprietary IC for other systems in your company. In that case, additional glue logic and external standard parts are necessary to connect to other parts of this new system, but the "core" of the system stays the same. The cost decreases because, for multiple systems, Non-Recurring Engineering cost (NRE) is reduced, and the unit price of this ASIC drops due to a higher volume usage.

Subsystem Partitioning

Going to the highest possible level of integration results in a design that strictly matches the specific system requirements. All random and glue logic, plus functional building blocks and other customized parts, are integrated into one chip. The ASIC design loses its flexibility as a general-purpose IC but, for this specific system, the efficiency improves in terms of space savings -- due to a lower IC count and less interconnect on the printed circuit board -- along with higher reliability and lower power dissipation. The resulting ASIC can be regarded as a complete system or subsystem to be included into an even larger total system.

Integration

With the availability of various silicon compilation techniques, the designer has to decide whether high-complexity standard parts such as RAMs or ROMs should be partially or totally integrated. VLSI's large library of megacells makes the choice even more complicated. The decision to place these components on- or off-chip can be based on purely technical requirements, such as general system performance, or commercial reasons, such as fluctuating prices for standard parts.

The Design Assistant can help you to effectively partition your system and to floorplan the individual chips. It is used prior to the logic design of the individual chips. From a block-level description developed in the Design Assistant, you can obtain a high-level schematic for each of the chips.

The Design Assistant is also useful in intermediate design phases when only some of the pieces of the design are completed. For this scenario, the Design Assistant can be used to get a better estimate of chip size.

Refer to the **Design Assistant Manual** for more information.

Chip Specifica- tion Package

The **Chip Specification Form** contains guidelines for assembling your chip specification package. When this package is completed, it contains a detailed description of the proposed chip:

- Functional block descriptions
- Physical size and packaging
- Electrical and timing specifications
- Test features

A sample of this form can be found in the **Signoff Forms** section within this binder.

This section contains an explanation of each portion of the **Chip Specification Form**.

1.0 Device Description

To complete the Device Description section of the **Chip Specification Form**, include this information:

- **1.1 General Function And Features** - Attach a brief description of the device function and its application.
- **1.2 Logic Block Diagram** - Use an upper-level schematic, or attach separate drawings of each logic block.
- **1.3 Detailed Block Description** - Include a detailed description of the device function for each separate functional block.
- **1.4 Physical** - Make a rough estimate of the chip's size and confirm that the prototype and production packages are available. Package information can be found in VLSI's **Semiconductor Package Selection Guide**.

Complete and attach:

- **AC/DC Specification Form**
- **Customer Package Marking Form**

These forms can be found in the **Signoff Forms** section within this binder.

The **Pad Placement Form** is also included in the Chip Specification package; however, it is not necessary to complete this form until the schematic is finished.

2.0 Electrical Specifications

To complete the Electrical And Environmental Specifications section of the **Chip Specification Form**, include this information:

- **2.2.1 Clock Input Timing Table** - On a separate page, insert a timing parameter table for clocks. A sample table is shown in Figure 17.
- **2.2.2 Clock Input Waveforms** - On the same page or following pages, insert timing diagrams for the clock timing. A sample waveform is shown in Figure 17.

Clock input timing table:

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	NOTES
T _{cyc}	Clock cycle	100			ns	1
T _{pw}	Clock pulse width	40		60	ns	
Tr/Tf	Clock rise/fall			5	ns	

NOTES:

1. Maximum clock cycle time is limited by the type of logic used. Consult VLSI for details on specific limitations

Clock input waveforms:

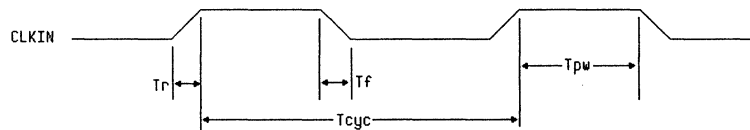


Figure 17. Clock Timing

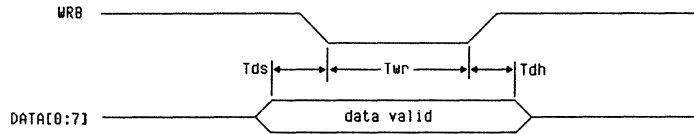
- **2.3.1 I/O Timing Table** - On a separate page, insert a timing parameter table for inputs and outputs. A sample table is shown in Figure 18.
- **2.3.2 I/O Timing Waveforms** - On the same page or following pages, insert timing diagrams for the inputs and outputs. Sample waveforms are shown in Figure 18.

I/O timing table:

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	NOTES
Tds	Data setup to WRB	10			ns	1
Tdh	Data hold from WRB	50	35		ns	
Twr	WRB pulse width	100		150	ns	
Tda	Data valid from RDB			35	ns	
Tdz	Data hi-Z from RDB	10		25	ns	

I/O timing waveforms:

DEVICE DATA BUS WRITE TIMING



DEVICE DATA BUS READ TIMING

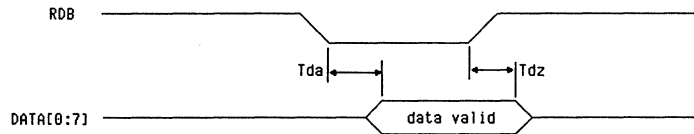


Figure 18. I/O Timing

- **2.4 AC Critical Path Description** - Create a drawing giving the input, cells on the path, output, capacitance load, and signal delays. A form that can be used for this purpose is shown in Figure 19.

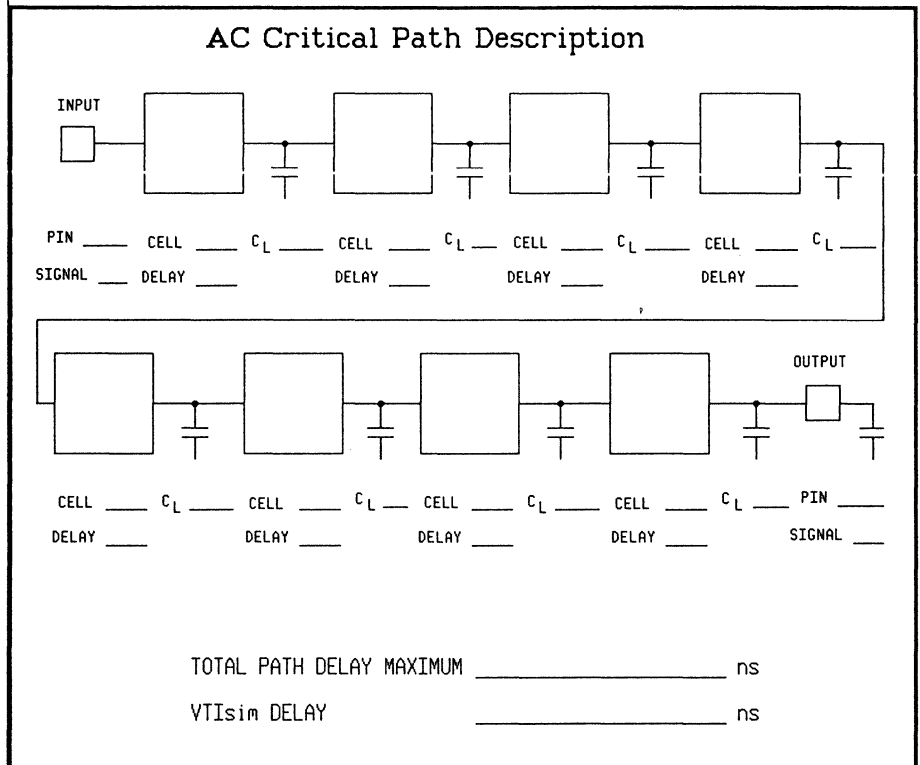


Figure 19. Critical Path Description

3.0 Test Features And Description

To complete the Test Features And Description section of the **Chip Specification Form**, attach separate pages with this information:

- **3.1 Testability Features** - Describe the test modes/ test logic strategies that will be used to enhance testability and test coverage. Include an overview of the test strategy for the device.

- **3.2 Implementation of Test Features** - Describe the partitioning of the test program. For each functional block, describe each test and which pins are of interest for each test block. Some pins may be multi-purpose and have various functions depending upon whether or not the device is in test mode.

**New
Design
Informa-
tion Form**

Include the **New Design Information Form** in your chip specification package for the preliminary design review. It helps you plan your usage of VLSI's standard cell and cell-based element libraries, and estimate the size of each functional block. It also provides the VLSI Technology Center with information and general specifications for your design, as well as an estimated schedule.

**Prelimi-
nary
Design
Review**

Using the **New Design Information Form** and the **Chip Specification Form** package as the initial specification for the proposed chip, conduct a preliminary design review. This review should carefully examine the proposed design methodology, architecture, AC and DC requirements, clock strategy, pinout and power pin requirements.

Test Plan

Based upon initial specifications in the **Chip Specification Form**, the Technology Center engineer or the customer can start work on a test plan. This plan specifies the overall test strategy, including how different sections of the chip are to be tested and what additional circuitry, if any, is necessary. Test strategies are discussed in Chapter 4, DESIGN FOR TESTABILITY.

Statement Of Work

When you have completed system partitioning and planning, create a chip specification. Determine the type of design you will be doing: turnkey, user logic, user, or joint. Fill out the **Statement Of Work** form, specifying responsibilities for each design task (Figure 20). Get the quote number from VLSI sales and write it in the **QUOTE #** blank.

STATEMENT OF WORK		Page 1 of 2	
CUSTOMER: VLSI Technology		DESIGN TYPE: User Logic	
		QUOTE #: 5847	
DESIGN STEP	RESPONSIBILITY: VLSI-CUST: BOTH	COMPL: DATE	COMPLETION SIGNOFF/ REFERENCE DOCUMENTS
CUSTOMER TRAINING Customer trained on VLSI design tools	X		VLSI CUST REF. DOC. _____
INITIAL SPECIFICATION ACCEPTANCE Agree on initial specification, design approach, and schedule		X	VLSI CUST REF. DOC. _____
DESIGN START Design work begins.	X		VLSI CUST REF. DOC. _____
SCHEMATIC CAPTURE Enter customer logic into VLSI tools with netlist ready for sim		X	VLSI CUST REF. DOC. _____
TEST PLAN Establish overall chip test philosophy and test vectors	X		VLSI CUST REF. DOC. _____

Figure 20. Statement Of Work

The **Statement Of Work** form is updated after each step has been completed. This form is a record of the design progress. Include it in your chip specification package for the preliminary design review.

CHAPTER 4

DESIGN FOR TESTABILITY

This chapter describes the concepts needed to design for testability in the system planning and schematic entry tasks.

Designing For Testability

Design for testability in integrated circuits refers to design approaches that enable you to test VLSI systems with minimal effort and maximum coverage. The key concepts in designing for testability are:

- Controllability -- The portion of circuitry to be tested must be easily stimulated from the inputs of the circuit.
- Observability -- The response of that portion of the circuitry must be easily observable from the outputs of the circuit.

In order to assure high fault coverage for the various types of functional blocks used in cell-based designs, VLSI Technology recommends these techniques:

- **Megacells** -- Use the high-coverage canned test programs provided with the megacell. Detailed instructions for using these programs are given in the **Megacell Test Development Methodology** application note, in the APPLICATION NOTES section of this binder.
- **Compiled Cells** -- Vector compilers that generate high-fault-coverage test vectors have been developed for several of the cell compilers. These compilers, described in the application note **Vector Compilers For Compiled Cells**, are currently supported only for VLSI Technology Centers. Please consult your Technology Center engineer.
- **Standard Cells, State Machine Elements, and Datapath Elements** -- Use the methods described in the application note entitled **Design-For- Testability Guidelines**, in the APPLICATION NOTES section of this binder.

The **Design-For-Testability Guidelines** apply to the whole design as a collection of interconnected function blocks, as well as to the standard cell or glue logic portions of your design. Using the suggestions in this application note, you can greatly reduce the amount of work involved in simulating and creating test patterns for some designs by adding a small amount of testability logic. For this reason, the question of circuit testability should be addressed at the very beginning of the logic design process.

For highly sequential circuits, testability issues can be complicated, and test logic can consume a significant portion of the complete logic design. Even for simple circuits, however, there are a few fundamental testability issues to keep in mind when designing a circuit.

General Guidelines

This section describes general design-for-testability techniques. Refer to the application note entitled **Design-For-Testability Guidelines**, in the APPLICATION NOTES section of this binder, for a detailed discussion of design-for-testability techniques.

Additional Circuitry vs. Long Simulations

Designers are sometimes reluctant to add test logic because they are trying to minimize the logic. It is important to consider the cost of long simulations during the design cycle, and to make the appropriate trade-offs regarding testability logic at the beginning of the design phase. In some cases, as with encoding sequential systems, minimizing the logic may lead to asynchronous designs with no recognizable cycles, which cannot be tested.

VLSI Technology Center engineers are available to assist with testability considerations at the beginning of the design phase.

Translating A Board

If you are converting the contents of a large TTL printed circuit board to a single chip, avoid direct translation. Boards and chips have different controllability and observability characteristics.

Provide Controllability

You can provide direct access to the inputs of each functional block by using a variety of techniques: bus architectures, de-gating, scan paths, microcode, and built-in tests. The important idea is that the stimulus to control a particular block can be applied to the inputs without passing through lots of other circuitry before reaching the block. This block isolation technique is described in detail in **Megacell Test Development Methodology**.

As shown in Figure 21, signals must propagate through blocks 1 and 2 in order to stimulate functional block 3. If these blocks contain sequential circuitry, a large number of vectors will be required to apply the stimulus to the inputs of block 3.

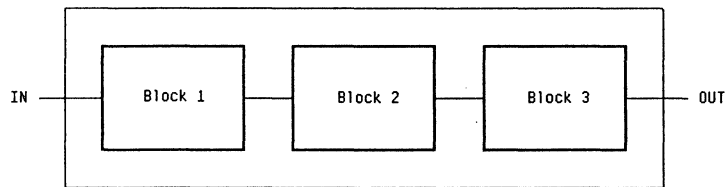


Figure 21. Controllability Problem

By allowing block 3 to receive its signals either from block 2 or directly from the circuit inputs, the stimulus can be applied directly to block 3 without being affected by blocks 1 or 2 (Figure 22).

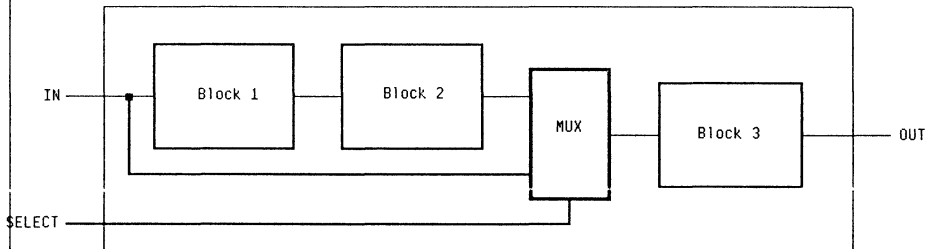


Figure 22. Stimulus From Inputs

Provide Observability

You can obtain direct access to the outputs of each functional block using a variety of techniques: bus architectures, de-gating, scan paths, microcode, and built-in tests. The important idea is that the output response values for a particular block can be observed at the outputs of the circuit without requiring the response values to propagate through lots of other circuitry before reaching the circuit outputs. This block isolation technique is described in detail in **Megacell Test Development Methodology**.

When trying to observe the response of functional block 1, the response values must propagate through blocks 2 and 3. If these blocks contain sequential circuitry, a large number of vectors will be required to observe the response from block 1 at the outputs of the circuit (Figure 23).

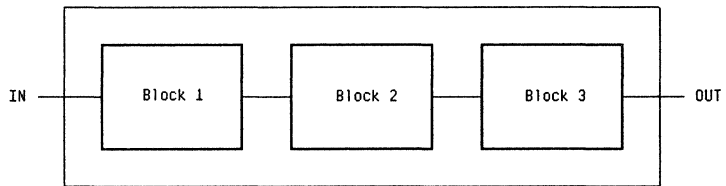


Figure 23. Observability Problem

By allowing block 1 to be directly accessed from the circuit outputs, as shown in Figure 24, the response values can be easily verified during test or simulation. These response values are not affected by blocks 2 or 3.

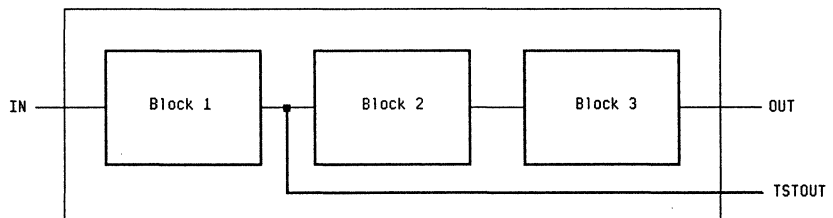


Figure 24. Response From Outputs

Circuit Initialization

You must be able to initialize the circuit to a known state. This allows most of the functional tests to be executed in an arbitrary sequence and provides for simple partitioning of the test vectors during test. Also, for a test program to be generated from a simulation trace file, none of the internal nodes of the circuit may be stimulated at any time during the simulation.

Even though a circuit may function correctly regardless of the initial state of storage elements upon power-up, this causes problems for the tester. For example, in order for the tester to be easily synchronized with the circuit, latches and memory elements must have a set/reset capability. By adding the reset capability, the circuit can be set directly into a known state, and the tester knows exactly how many vectors to apply to achieve the expected output.

Test Modes

If there are no I/O pins available for use as test mode input pins, it is often possible to use some illegal combination of inputs, or an input combination that cannot occur during the normal operation of the circuit, to place the circuit in test mode. On-chip test logic can be added to test portions of the circuit while it is operating in test mode. You can add simple bypass multiplexers to key inputs and outputs of the circuit, to directly access important internal nodes.

Use The Design-For-Testability Guidelines

This is a summary of the guidelines:

- Provide set/reset capabilities on latches or memory devices and provide the capability to initialize the circuit to a known state.
- Avoid asynchronous circuitry.
- Allow internal clocks to be bypassed.
- Allow counters, dividers, and other sequential circuitry to be bypassed.
- Provide access to RAMs, ROMs, and other functional blocks from the inputs and outputs of the circuit.

- Partition circuitry that has more than 8 bits of cascaded latches, such as 24-bit counters and 10-bit dividers.
- Avoid nesting of sequential circuitry.
- Provide access to allow redundant circuitry to be tested.
- Be aware of signals that fan out and reconverge at the inputs to a functional block.
- Allow analog circuits to be bypassed during testing of digital portions.

Detailed examples of each guideline are presented in the TESTABLE DESIGN GUIDELINES chapter of the **Design-For-Testability Guidelines**.



CHAPTER 5

DESIGN ENTRY

This chapter describes the schematic entry task for all types of cell-based elements, including design screening and review and completing the pad placement and bonding forms.

Creating Functional Blocks

This section describes how to create a functional block containing one or more of these elements:

- Standard cells
- Compiler cells
- Megacells
- Datapath elements
- State machine elements

You can place and interconnect instances of these blocks into higher-level schematics to create a hierarchical design, in accordance with your system partitioning plan. The top level

of the design hierarchy should contain all pads, pad drivers and a core block.

With Standard Cells

To create a functional block with standard cells, be sure you have the VSC10 or VSC100 standard cell library on your search path:

- VSC10 - pvsc010d
- VSC100 - pvsc100d

If you have no `vt1.bo` startup file, you can add the library to your search path by selecting the canned startup file for the library you want when you first bring up the tools. This is described in the **VTItools Fundamentals Manual**.

If you already have a startup file, you can bring up the tools, go to the cell manager window, and click `library` → `search path`. When the search path menu is displayed, add the library name to your search path as described in the **Cell Manager Manual**.

When the library you want has been added to your search path, bring up the schematic editor window and create a functional block composed of standard cells, or other types of cells, as described in the **VLSI Schematic Editor Manual**.

To make an icon of the functional block, click on `misc` → `make icon`. You can place this icon in a higher-level schematic. You can also use the VLSI icon editor to edit this icon or create a new one. If you do, be sure to:

- Bring up the `set up` property sheet while you have the schematic loaded in the schematic editor and type in the name of your icon cell.

- Click on when you are in the icon editor window, to match your completed icon cell to the functional block schematic.
- Make sure the node names on the new icon match the node names in your schematic.

You can make your own custom standard cells and use them as you would VLSI's standard cells. Refer to the application note entitled **Designing Custom Standard Cells** for more information.

With Compiler Cells

Compiler cells are placed in a schematic the same way as standard cells. Be sure the compiler cell library is on your search path as described for the standard cells. The libraries are:

- VCC10 - vcc010d
- VCC100 - vcc100d

You have to specify cell parameters in order to create a compiler cell:

1. Bring up the compiler cell library window, `cellLib`.
2. Find the type of compiler cell you want in the browser.
3. Click on either the `[tp1]` (template cell) or the `[pc1]` (parameter cell) for the cell type you want.
4. Click on .
5. When the prompt box appears, type in a name of your choice for the compiler cell to be created.

6. Click on to specify parameters for the cell, such as RAM size.
7. Click on → to make an [sc] schematic cell of the compiler cell in your working directory.
8. In the schematic editor, place the completed compiler [sc] cell in your schematics as you would a standard cell; a default icon is automatically generated by the system when the cell is placed.

With Megacells

Each megacell is in a separate library; you should have the names of the megacell libraries you want to use on your search path. The name of the library is the megacell name plus 010d for VMC10 or 100d for VMC100. For example, the 2 micron M82C54 megacell is in the m82c54010d library, and the 1.5 micron version is in m82c54100d.

In the schematic editor, place the megacell in your schematics as you would a standard cell.

With Datapath Elements

To create a datapath functional block:

1. Make sure that the 2 micron vdp010d or 1.5 micron vdp100d library is on your searchpath.
2. Using the schematic editor, create a specification schematic with datapath elements as described in the **Datapath Compiler Manual**.
3. Place a title block in the top level datapath schematic.
4. Click on .

5. Open a cellLib window and select the datapath template, [tp1]cdp, in the browser.
6. Click on . The system will prompt you for a parameter cell name; type in a name that is **DIFFERENT** from the name of your specification schematic.

IMPORTANT: The name for the model-schematic is derived from the [pcl] name you enter here. The model-schematic must have a different name from that of your specification schematic.

For example, if you name your specification schematic dpath, the resulting schematic cell is [sc]dpath. If you give the same name to the [pcl] cell, it is also applied to the resulting functional block schematic; so the new [sc]dpath overwrites the old, and your specification schematic is destroyed.

7. Click on . The system displays a parameter sheet. Type in the name of your specification schematic on the DP_specification line to cross-reference the parameter cell to your datapath specification schematic.
8. Click on → to create a functional block [sc] schematic cell.
9. Place the resulting functional block schematic cell in your schematics as you would a standard cell.

With State Machine Elements

To create function blocks with state machine elements:

1. Create the *filename.sm* text file containing the state machine specification.
2. Be sure that the `smachn000d` library is on your search path.
3. Bring up the cellLib window.
4. In the `smachn` library, select the cell called `[tp1]smachn` and click on .
5. When the system displays the prompt box, type in the name to assign to the parameter cell; this should be the same name as your state machine specification file. The system creates `[pc1]filename`.
6. Click on ; when the property sheet is displayed, type in the name of your state machine specification file, to cross-reference it to the `[pc1]` cell.

There are two ways you can create a schematic of your state machine element to place in a higher-level schematic. In the cellLib window you can load the `[pc1]` cell and use → to get an `[sc]` cell of your state machine circuit. Or:

1. In the schematic editor window, go to your working directory and select the state machine `[pc1]` cell.
2. Click the button where you want to place the state machine element; there will be no cell outline. The system automatically generates an `[sc]` cell and places it in your schematic.

Schematic Entry

Enter your schematic using the VLSI schematic editor, VTIschematic. Refer to the **VTIschematic Manual** for detailed instructions. This section contains information on general topics that may be of use when entering schematics.

Logic

Synchronous logic is preferred, because testers use cycles, and the circuit should be able to work under cycle constraints. If it is necessary to use asynchronous logic, follow the recommendations in Chapter 6, SIMULATION.

Borders

Use the same size border for most schematics. B size is generally large enough for all schematic pages. Place the latest change date, the project name, and the responsible engineer's name in the lower right corner of each schematic.

Node Names

The character strings VDD, PWR, VCC, BULK, VSS and GND are reserved system names, or have special meanings in one or more of the VLSI tools. Do not use them in node, instance or connector names in your schematics, or as part of a name, such as TOPVSS.

Busses

Use busses whenever possible; they are quicker to enter and easier to trace. However, do not bundle unrelated signals in order to make an arbitrary bus.

Signal Drive

To provide adequate signal drive:

- Buffer all internal signal outputs sufficiently.
- Internal signal rise/fall delays should be kept to a minimum; 10 ns or less is recommended. The 10 ns maximum rise/fall time for internal nodes is independent of chip clock speed. Extremely slow rise and fall times on internal nodes can lead to excessive DC current and may possibly disturb internal states.
- Clock signal rise/fall delay to internal cells should not exceed 5 ns.
- Be aware of output diffusion capacitance on three-state drivers; it is sometimes high because a number of drivers are driving the same node.
- Create a clock tree to adequately buffer clock inputs and to avoid clock skew to different parts of the chip. See the application note entitled **Power And Clock Distribution In Cell-Based IC Design**, in the APPLICATION NOTES section of this binder, for more details.

Clocks and Clock Buffers

All unbuffered flip-flops and latches must be driven by a clock buffer cell in order to minimize clock skew.

A clock buffer cell's output pins must drive **both** inputs on one or more cells, in order to reduce the amount of skew between output pins C and CN (Figure 25).

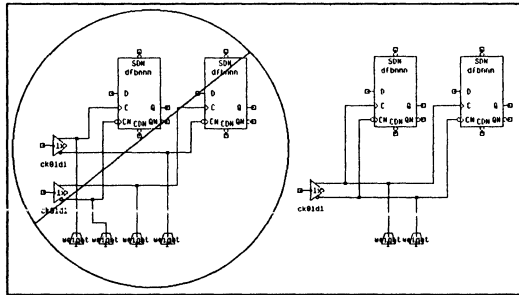


Figure 25. Clock Skew Problem

Do not scatter clock trees among many schematics as it makes visual checking of the schematics difficult. Try to keep the clock tree on a single sheet.

Use flip-flops and latches with internally buffered clocks.

Metal Migration, Current Density and Parallel Buffers

A design formula for metal migration, based on RMS current, that you can use to determine the number of fanouts a buffer can drive is:

$$B * L * F$$

where:

B = Buffer size; 1 for a 1X buffer, 2 for a 2X buffer, and so on.

L = The number of fanouts. The interconnect capacitance should be converted into the equivalent number of fanouts, and added to the value of L.

F = The frequency of the node in MHz.

The result of this calculation should stay below the current density figure of merit, which is:

- VSC10 - 1003
- VSC100 - 9034

As a general rule, use 1X drive per 3 loads, regardless of clock speed, as the flip-flop delay is the same at any frequency of operation.

Do not forget to include some estimate of interconnect capacitance when making these calculations.

Parallel Logic

It is bad design practice to parallel cells with dissimilar logic functions, as it results in contention. If this situation should occur in a design, the VLSI netlist screener produces an error message to indicate that a signal is driven by more than one type of cell, unless the cells are three-state (Figure 26).

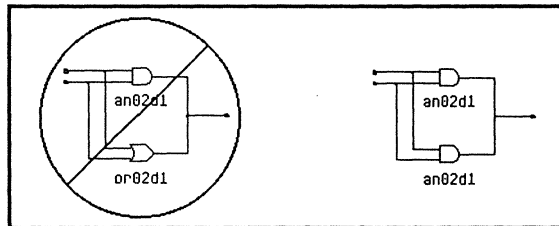


Figure 26. Parallel Cells

When a signal is driven by wired-OR cells whose input signals are not common, contention results. If this situation occurs in a design, the VLSI netlist screener produces an error message unless the cells are three-state (Figure 27).

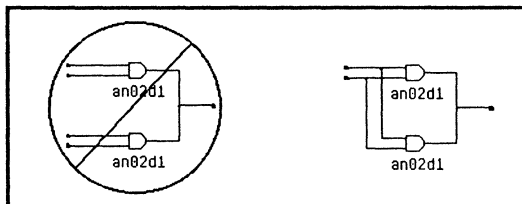


Figure 27. Common Inputs

Bus Repeater Cells For Three-State Nodes

Bus repeater cells prevent excess power consumption caused by floating three-state nodes driving other logic. The bus repeater cell, RP01D1, holds the last value that was driven onto the node. Only one bus repeater cell should be used for each three-state node. Bus repeater cells should not be used on nodes that are not three-state. If the circuit is such that the three-state node never floats, a bus repeater is not required.

Weighting

You can specify routing priority by using net weights in your schematic.

Net weights are used to guide the automatic place and route system. If a node is weighted, instances which share the node are placed together during place and route. Refer to the description of the WEIGHT cell in the **VSC10** or **VSC100 Portable Library Manual**. To weight a node, select the weight icon in the browser under **logicComp SYMBOLS**, and place it near the line you want to weight. Wire from the

weight icon to the selected line as you would any other cell (Figure 28).

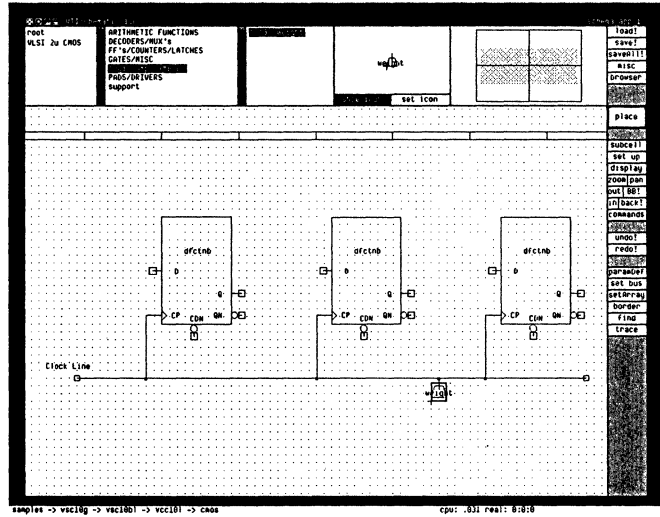



Figure 28. Weight Symbol

To set the value of the weight, click on **commands** → **SetParam**, then point to the weight and click  to bring up the weight property sheet (Figure 29).

Instance weight ub Parameters				
accept				
id	name	type	value	default
1	weight	direct	7	1

Figure 29. Weight Property Sheet

The value set in this box becomes the routing priority for that net. All nets have a default weight of 1. A net with a higher weight is considered more important than other nets at cell placement time. A net assigned a weight of 5 is considered 5

times as important as a normal net. Weights should be used sparingly; never use a weight greater than 10. Excessive use of weights will adversely affect routing time and efficiency.

Only one weight has to be attached to a net: everything traced in a single net by VTIschematic has the same routing priority. The entire net highlighted in the illustration below has the same priority (Figure 30).

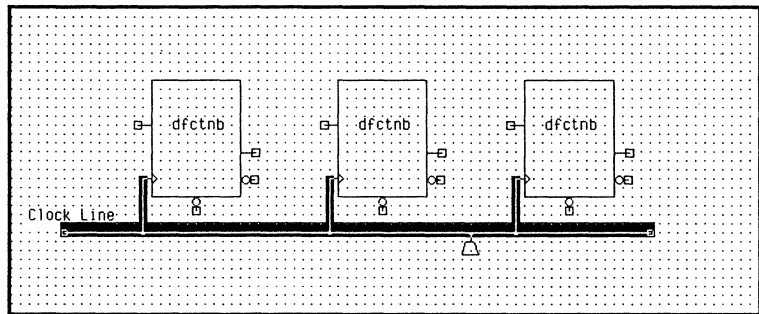


Figure 30. Weighted Net

Use weights to minimize wiring and parasitic capacitance on a node. Do NOT use weights to minimize chip area, since weights alter the overall wiring and usually result in increased die area. Never weight a node like RESET that runs all over the chip.

Weights on Clock Buffers

The clock buffer cell's output pins need to be weighted in order to minimize wire length and clock skew. The weight must be balanced, as shown in Figure 31.

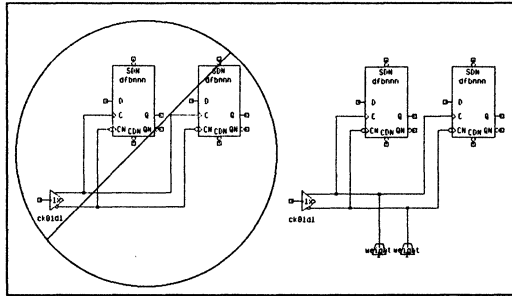


Figure 31. Weights On Clock Buffers

Buffers

Make sure that any line with a heavy load on it, such as a global RESET line, is properly buffered.

Simulation And Test

Add any circuitry necessary for efficiently testing the chip. Make sure that this circuitry does not interfere with normal operation.

Include circuitry so that the circuit is initialized to a known state for simulation, testing and actual operation.

Refer to Chapter 4, DESIGN FOR TESTABILITY, and the **Test Generation Guidelines** application note, for more information on designing testability into your circuit. In addition, the application note entitled **Megacell Test Development Methodology**, provides special guidance for megacell testing. Megacells, and some compiled cells, come with canned test programs, which must be integrated into the overall test program.

Pads

Use the minimum drive output drivers that meet your AC and DC requirements. Available driver output currents are 2, 4, 8, and 12 mA.

Determine the number of power and ground pins needed using the application note entitled **Power And Clock Distribution In Cell-Based IC Design**, in this binder. The power to the padding should be separated from the power to the core, if possible. Place PCVDD1 and PCVSS1 pads for core power pins. Place PCVDD2 and PCVSS2 pads for padding power pins. If padding and core power pins must be common, place PCVDD3 and PCVSS3 pads.

The number of VDD and VSS pads in the schematic should match the number used in the layout. This simplifies your netcompare results and design documentation.

This step should be completed early in the design cycle, either during system planning or design entry.

Specifying Pad Placement in your Schematic

Optionally, you can assign pad placement in VTIschematic by modifying the property sheet associated with a specific pad.

You can do this by clicking on → , clicking on the pad, and clicking to bring up the parameter sheet. The value assigned to PinNumber is used for the die pad number for this pad.

Put a # sign **before** the pad number. Any number not preceded by a # is interpreted as the package pin number, not the intended die pad number (Figure 32).

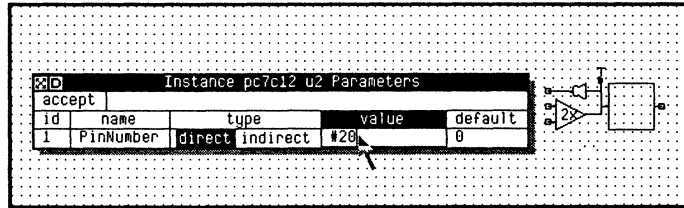


Figure 32. Die Pad Number Specification

In the chip compiler, die pad number 1 corresponds to the center pad on the top side of the padring. The pad number increases counterclockwise. The location of package pin number 1, however, varies depending upon the package you select; refer to the **Semiconductor Package Selection Guide**.

Functional Blocks

All instances that are to be placed into a given arbitrary block during the physical design of the chip should be placed in the same schematic. There are several reasons for this:

- In the chip compiler, this allows you to fix the placement of this block of standard cell logic, which may have critical timing, by using the hierarchical name in a wild card specification for a standard cell area.
- If you want to do a chip compilation on the block level to group standard cells and arbitrary cells, such as RAMs, ROMs, PLAs, megacells and compiled layout, then you should hierarchically group these cells in the schematic.

- If you are editing multiple arbitrary cells in the composition editor, these cells must be hierarchically grouped in order to use the chip compiler to place your composed cell with the rest of the logic.
- You can perform a sub-block netcompare on an arbitrary block when the physical hierarchy matches the schematic.

The Top Level Design

The top level of the design hierarchy should contain all the I/O pads, any required pad drivers and a core block (Figure 33). Place a connector on the external outputs of all pads. Only VDD, VSS and I/O pad signals should have connectors on them at the top level. The VLSI netlist screener will report an error otherwise.

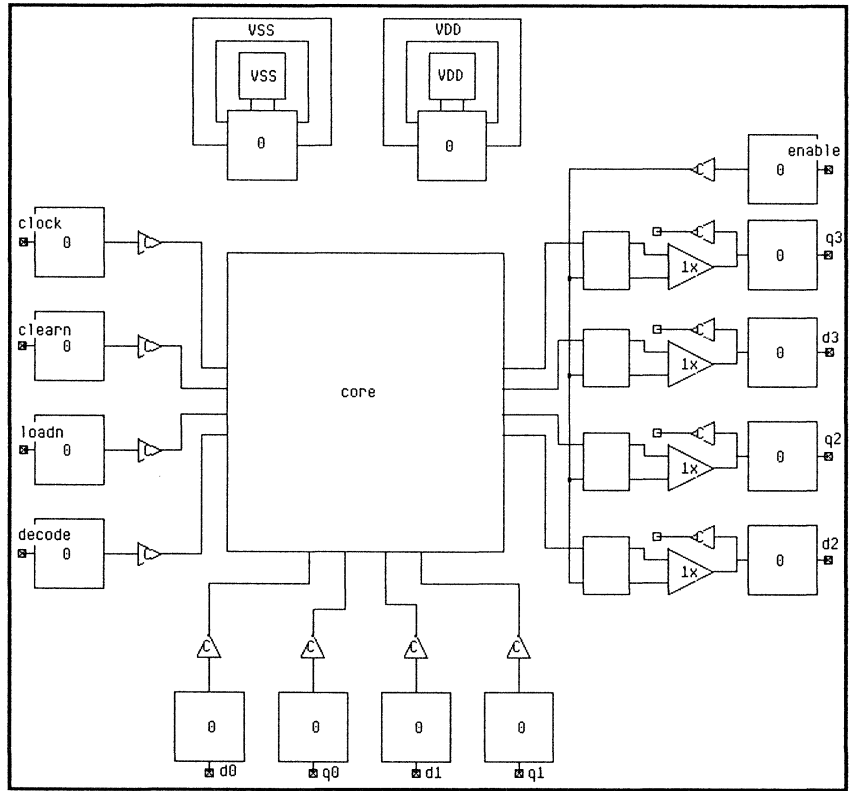


Figure 33. Top Level Of Design

Three-state, bidirectional, open drain and open source pad control signals should be explicitly named (Figure 34); these signals must be watched in the 1 MHz simulation used for test vector generation. Refer to the **VTIschematic Manual** for the syntax of legal signal names. Also, all three-state, bidirectional, open drain and open source output buffers must be driven by a pad driver. The output pad's NGATE and PGATE pins must be connected to a single driver cell's NGATE and PGATE pins respectively.

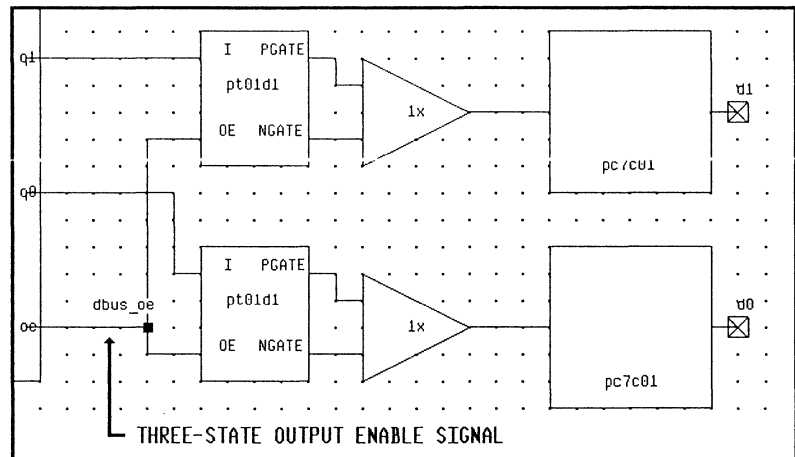


Figure 34. Pad Drivers

For designs created with standard cell libraries prior to the VSC10/VSC100 series, use TTL, Schmitt Trigger or CMOS level shifters where appropriate. Doublecheck that level shifters are placed correctly, although input pads or output buffer cells that are not correctly connected to a level shifter are flagged by the netlist screener. Refer to the VLSI **Netlist Screener Manual** for more information on these checks.

Turnkey Design Markups

In turnkey designs, if VLSI schematics are derived from customer schematics, the Technology Center engineer should “yellow line” the VLSI and customer schematics to make sure that they are logically equivalent. This is a node-by-node comparison; each node on one schematic that matches a node in the other schematic is highlighted in yellow. When the yellow-line is completed, nodes that are missing (not highlighted) in either schematic are investigated.

Design Screen And Review

The screen and review checks should be performed at various checkpoint stages during the logic design. When you want to check a functional block, or the top-level schematic, create an [hns] netlist as described in the **Schematic Editor Manual**.

Netlist Screening

Run VTIscreen to identify simple errors and potential problems in the logic design. Refer to the **VTIscreen Manual** for more details.

When you run VTIscreen to read the [hns] netlist created by the schematic editor, it:

- Calculates the internal cell and I/O pad utilization, and prints the results in a summary report.
- Provides a Design Statistics report listing information on the number of nets and FromTos -- connections between two pins -- in your design.
- Provides a Design Complexity report listing gate equivalents per cell.
- Checks for certain design and connectivity errors and other potential problems in your design.
- Prints the report to your terminal as well as producing a disk file, [scr], containing the entire report.
- Optionally writes a Logical Design Structure file, [lds], to be used by VLSI's design review or vector conversion programs.

An example follows.

```
VTIscreen 1.2
VTIscreen> read [hns]latch8
VTIscreen> screen
```

VTIscreen - latch8 design.

```
*****
*                               VTIscreen Utilization Summary for latch8                               *
*                               *                               *                               *
* Number of Input Pads                               0 *
* Number of Output Pads                              0 *
* Number of Bidirectional Pads                       0 *
* Number of VDD                                       0 *
* Number of VSS                                       0 *
* Number of VDD Pads                                  0 *
* Number of VSS Pads                                  0 *
* Number of VDD Core                                  0 *
* Number of VSS Core                                  0 *
*
* Number of Gate Equivalents Used                    30.00 *
*
* Number of Non-Primitive Blocks Used                 1 *
*****
Press return to continue
```

Design Statistics:

```
Number of Nets           : 1
Average Number of Pins per Net : 9.0
Maximum Number of Pins per Net : 9
```

Design Complexity:

Cell Name	Number Occurrences	Number Gate Equivalents Per Cell	Total Gate Equivalents
lanfnn	8	3.50	28.00
ni01d3	1	2.00	2.00
Totals:	9		30.00

*** Summary of design warnings and errors ***

WARNING> The following signals do not drive any cells:

```
u2.Q      u3.Q      u4.Q      u5.Q      u6.Q      u7.Q
u8.Q      u9.Q
```

ERROR> The following primary input signals should go thru input buffers:

```
Y7      Y6      Y5      Y4      Y3      Y2      Y1
Y0      GN
```

ERROR> The following primary output signals should go thru output buffers:

P7 P6 P5 P4 P3 P2 P1
P0

End of screener check

A summary of the reports, warnings, and errors provided by VTIscreen can be found in the **VLSI Netlist Screener Manual**.

To create an [lds] file for the design review or vector conversion program to read, use the **save** command. Before creating an [lds] file, you should remove any errors that the **screen** command finds. Only after your design is free of errors should you proceed to use the design review program.

Design Review Program

VLSI's design review program assists standard cell designers in analyzing and simulating their designs, by generating:

- A pre-route or post-route design report
- A post-route back-annotation file

The design report contains a design summary which gives the number of cells, transistors, equivalent gates, inputs, outputs, bidirectionals, and pins, as well as capacitance, power dissipation and average frequency. There can be up to eleven tables following the design summary:

- Internal node timing verification
- Output pad timing verification
- Standard cells used
- Macro inputs connected to VDD
- Macro inputs connected to VSS

- Macro inputs unconnected
- Macro outputs unconnected
- Signal name/alias cross-reference
- Ports of customer-designed macros
- Worst rising ramp delay
- Worst falling ramp delay

An example is:

```
VTI> util review
```

```
*****
*****
**          **
**   A == Pre route design report (.RPP)      **
**   B == Post route design report (.RPA)     **
**          **
**   C == Post route back annotation file (.PST) **
**          **
**   Q == Quit the execution                  **
**          **
*****
*****
```

```
REVIEW: a
```

```
Enter design name: latch8
```

```
unable to open LATCH8.OFF
```

```
all off chip loads set to 0
```

```
Is this a gate array design? (Y/N/Q): n
```

```
Is this a standard cell design? (Y/N/Q): y
```

```
Average Frequency = percentage1 * frequency1 + percentage2 * frequency2 ...
```

```
Enter a percentage (%) : 100
```

```
Enter a frequency (MHz): 1
```

```
Average frequency = 1 MHz
```

```
SELECT A DELAY MODE:
```

```
1 : MINIMUM = MAXIMUM * 0.36
```

```
2 : TYPICAL = MAXIMUM * 0.6
```

```
3 : MAXIMUM = DATA BOOK INDUSTRIAL
```

- 4 : MINTYP = (MINIMUM+TYPICAL)/2
- 5 : TYPMAX = (TYPICAL+MAXIMUM)/2
- 6 : MAXMIN = (MAXIMUM+MINIMUM)/2
- 7 : DEFINE = VDD PROCESS TEMPERATURE

Enter a delay mode: 6

Do you want to write the timing list? (Y/N) : y

Select a sorting method for timing list (CR = signal):

- SIGNAL = by signal names MACRO = by macro types
- FI = by number of fan-ins FO = by number of fan-outs
- GATECAP = by gate capacitance PREMET = by predictive metal capacitance
- PRER = by predictive rise time PREF = by predictive fall time
- CANCEL = no sorting

Response (? for valid responses): premet

Should the list be sorted in ascending or descending order? (A/D) : d

Should the capacitance be in pf or unit load? (PF/UL) : pf

Do you want to write the list again with another sorting method? (Y/N) : n

=====
OUTPUT FILE: LATCH8.RPP
=====

```

*****
*
*                               VLSI DESIGN SUMMARY
*                               (PRE PLACE AND ROUTE)
*
*   DESIGN NAME:                latch8
*   USER   NAME:                USER       SOURCE:                hn1
*   DATE:                   24 AUGUST 1987   TIME:                16:21:33
*
*****DESIGN STATISTICS*****
*
*   NUMBER OF MACRO TYPES          2   NUMBER OF MACROS USED          9
*   NUMBER OF TRANSISTORS         120  NUMBER OF EQUIV GATES         30
*
*   NUMBER OF PRIMARY INPUTS      9   NUMBER OF PRIMARY OUTPUTS     8
*   NUMBER OF PRIMARY BIDIRS      0   AVAILABLE PADS USED          17
*
*   MACRO INPUTS TO VDD           0   MACRO INPUTS TO VSS          0
*   NO. OF UNC. MACRO INPUTS      0   NO. OF UNC. MACRO OUTPUTS    8
*
*   NUMBER OF PINS                 9   NUMBER OF NETS                1
*   AVERAGE PINS/NET              9.00  MAXIMUM PINS/NET              9
*
*   TOTAL GATE CAP LOAD(PF)       1.12  TOTAL METAL CAP LOAD(PF)      .60
*   TOTAL OFF CHIP LOAD(PF)       .00   TOTAL DIFFUSION LOAD(PF)     1.52
*   TOTAL LOAD(PF)                 3.24  AVERAGE FREQUENCY(MHz)      1.00
*   POWER DISSIPATION(mW)         .10
*
*****
*
*                               DELAY IMPLEMENTATION FOR TIMING VERIFICATION:
*
*                               MINIMUM = MAXIMUM * 0.36
*                               MAXIMUM = DATA BOOK INDUSTRIAL
*                               MAXMIN = (MAXIMUM + MINIMUM) / 2
*
*****

```

TABLE 1 STANDARD CELLS USED

MACRO NAME	OCCURENCES	TRANSISTORS / MACRO	TRANSISTORS TOTAL	MACRO TYPE
1 lanfnn	8	14	112	INTERNAL
2 ni01d3	1	8	8	INTERNAL
TOTAL	9		120	INTERNAL

TABLE 2 PORTS OF CUSTOMER DESIGNED MACROS

```

=====
MACRO NAME:      latch8
  INPUTS :
                GN
                Y0
                Y1
                Y2
                Y3
                Y4
                Y5
                Y6
                Y7

  OUTPUTS:
                P7
                P6
                P5
                P4
                P3
                P2
                P1
                P0
=====
    
```

TABLE 3 MACRO OUTPUTS UNCONNECTED

MACRO	PORT	INSTANCE
1 lanfnn	Q	u2
2 lanfnn	Q	u3
3 lanfnn	Q	u4
4 lanfnn	Q	u5
5 lanfnn	Q	u6
6 lanfnn	Q	u7
7 lanfnn	Q	u8
8 lanfnn	Q	u9

TABLE 4 OUTPUT PAD TIMING VERIFICATION WITH MAXMIN DELAY

```

=====
TYPE = TYPE OF THE MACRO DRIVING THE SIGNAL
ULCAP = OFF CHIP LOAD CAPACITANCE [IN UNIT LOAD]
PFCAP = OFF CHIP LOAD CAPACITANCE [IN PICO FARAD]
RISE = BASE RISE DELAY + RISE LOAD FACTOR * OFF CHIP LOAD [IN NANO SECOND]
FALL = BASE FALL DELAY + FALL LOAD FACTOR * OFF CHIP LOAD [IN NANO SECOND]
=====

```

TOP	SIGNAL	NAME	TYPE	ULCAP	PFCAP	RISE	FALL
P7		LANFNN		.0	.0	1.4	1.0
P6		LANFNN		.0	.0	1.4	1.0
P5		LANFNN		.0	.0	1.4	1.0
P4		LANFNN		.0	.0	1.4	1.0
P3		LANFNN		.0	.0	1.4	1.0
P2		LANFNN		.0	.0	1.4	1.0
P1		LANFNN		.0	.0	1.4	1.0
P0		LANFNN		.0	.0	1.4	1.0

TABLE 5 INTERNAL NODE TIMING VERIFICATION WITH MAXMIN DELAY
(sorted by predictive metal capacitance in descending order)

```

=====
TYPE = TYPE OF THE MACRO DRIVING THE SIGNAL
FI = NUMBER OF DRIVING PORTS
FO = NUMBER OF DRIVEN PORTS
GCAP = GATE CAPACITANCE THE SIGNAL DRIVES [IN PICO FARAD]
MCAP = METAL CAPACITANCE THE SIGNAL DRIVES [IN PICO FARAD]
RISE = BASE RISE DELAY + RISE LOAD FACTOR * (GCAP + MCAP) [IN NANO SECOND]
FALL = BASE FALL DELAY + FALL LOAD FACTOR * (GCAP + MCAP) [IN NANO SECOND]
=====

```

TOP	SIGNAL	NAME	TYPE	FI	FO	PREDICTIVE			
						GCAP	MCAP	RISE	FALL
u10.Z		NI01D3		1	8	.6	.6	2.4	2.4

TABLE 6 WORST RISING RAMP DELAYS

```

=====
RAMP = LOAD FACTOR * (GATE CAP + METAL CAP) [IN NANO SECOND]
RECOMMENDED RAMP DELAY LIMIT = 10 NS; ramp values = data book industrial
=====

```

TOP	SIGNAL	NAME	TYPE	RAMP
u10.Z		NI01D3		1.9

TABLE 7

WORST FALLING RAMP DELAYS

RAMP = LOAD FACTOR * (GATE CAP + METAL CAP) [IN NANO SECOND]
 RECOMMENDED RAMP DELAY LIMIT = 10 NS; ramp values = data book industrial

TOP	SIGNAL	NAME	TYPE	RAMP
u10.Z		NI01D3		1.9

The post-route design report also contains an Actual vs. Predicted Wirelength Comparison report.

Information on usage and examples of design review checks can be found in the **VTireview Manual**.

Pad Placement Form

When you have finished design entry, complete the **Pad Placement Form** and attach it to the **Chip Specification Form**.

Bonding Diagram

Complete a bonding diagram for prototype and production packaging. To generate the bonding diagram, plot the full chip CIF at the same scale as the blank bond form for the desired package; this is usually 20X. Bond form part numbers and package information are given in the **Semiconductor Package Selection Guide**. The bond form documentation for the package you select gives the scale at which the bond form was plotted. Place the plot in the center of the blank form and draw wire connections between the package pins and the die pads.

A good general procedure to follow when generating your bonding diagram is:

1. Mark the low inductance pins on the bond form. Package pin inductances for many of the available VLSI packages are given in Appendix B.
2. Bond VSS and VDD first, using the low inductance pins.
3. Bond the outputs in a distributed manner, without bunching them.
4. Bond the inputs.

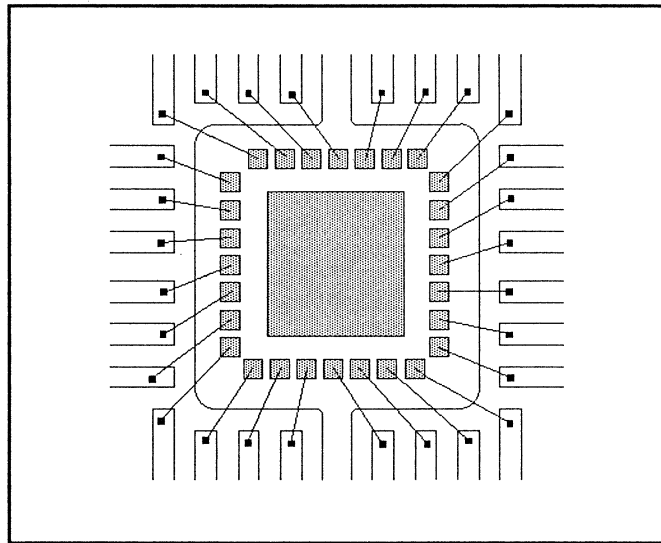


Figure 35. Sample Bonding Diagram

When bonding, use these common-sense guidelines:

- Bond from pad to pin.
- Keep angles to a minimum.
- Don't cross over pads, core, or other wires.



CHAPTER 6

SIMULATION

This chapter describes the logic/timing simulation, fault simulation and Logic Design Review tasks.

Netlists For Timing Verifi- cation and Simulation

You will need netlists of all your functional blocks and the top-level design in order to run the timing verifier and the simulator. This section describes how to create netlists for the various types of cell-based elements.

Standard Cells, Compiler Cells and Megacells

When you have the schematic loaded in the schematic editor, click on → or to create an [hns] hierarchical netlist. Load this netlist into the simulator. While the simulator is loading your file, it is flattened to an [fns] netlist file.

State Machine Elements

You can initially simulate a state machine element without creating a netlist. Create a [pcl] parameter cell, as described in the DESIGN ENTRY chapter, then bring up the simulator window and load the [pcl] file. Set the clock and inputs, and run the simulation.

To generate a portable netlist for standard cell implementation, open a cellLib window and load the state machine [pcl] file. Click on → to generate a file named [hns]pclName_P.

You can place the state machine element in a higher-level schematic, generate an [hns] netlist for that schematic, and load the [hns] into the simulator. In this case, VTIsim will simulate the state machine with a behavioral model. The model does not model delays, loads, or drives accurately, but it is quicker than simulating at gate level. To use the portable netlist [hns] you created instead of the behavioral model, make a switch cell, as shown in Figure 36.

On/Off	Src Typ	Src Name	Src Pcl	Inst	Dest Typ	Dest Name	Dest Pcl
on	mde	smachn	counter	*	hns	counter_P	

Figure 36. State Machine Switch Cell

Bring up the switch cell editor, click on , and click on the **add** box to create a new switch cell. Click on the **Src Typ** box until it displays **mde**. Click on the **Src Name** box and type in **smachn**. Click on the **Src Pcl** box and type in the name of your state machine [pcl] cell. Click on the **Dest Typ** box and type in **hns**. Click on the **Dest Name** box and type in the name of your netlist file, *pclName_P*.

When you bring up the simulator window, type in `set switch pclName` before you load your chip. When it finds the state machine model, it will replace it with the compiled netlist.

Datapath Elements

Datapath elements can be implemented as standard cells with a portable library netlist, or as compiled layout with a layout model netlist. Which netlist you use in simulation depends upon how you implement your datapath.

To generate a portable netlist for standard cell implementation, open a cellLib window and load the datapath [pcl] file. Click on → to generate a file named [hns]pclName_P.

To generate a layout model netlist for compiled layout implementation, open a cellLib window and load the datapath [pcl] file. To generate a file named [hns]pclName_L, click on → .

To use the netlist you generated, you need to create a switch, as shown in Figure 37; refer to the state machine section for details. For the portable netlist, your switch should contain:

- Src Typ of mde
- Src Name of mdp
- Src Pcl of pclName
- Dest Typ of hns
- Dest Name of pclName_P

Switches									
accept	add	del	move						
On/Off	Src Typ	Src Name	Src Pcl	Inst	Dest Typ	Dest Name	Dest Pcl		
on	mde	mdp	counter	*	hns	counter_P			

Figure 37. Portable Netlist Switch Cell

As shown in Figure 38, your switch for the layout model netlist should contain:

- Src Typ of mde
- Src Name of mdp
- Src Pcl of *pclName*
- Dest Typ of hns
- Dest Name of *pclName_L*

Switches									
accept	add	del	move						
On/Off	Src Typ	Src Name	Src Pcl	Inst	Dest Typ	Dest Name	Dest Pcl		
on	mde	mdp	counter	*	hns	counter_L			

Figure 38. Layout Model Netlist Switch

When you bring up the simulator window, type in `set switch pclName` before you load your chip. When it finds the datapath model, it will replace it with the compiled netlist.

Flattening The Netlist

When you load a hierarchical netlist [hns] into VTIsim, it automatically flattens the netlist down to the transistor level, into an [fns] form that it can use. To save time, you can use the HNL utility to flatten your netlist. Using the non-graphic shell environment or a VTItterminal window running `shell!`, type utility HNL to the VTI> prompt. Then type `flatten` to get into the `flatten` subsystem. Finally, flatten your [hns] file using options `simflatten`. For more information, please see the **VTItools Utilities Manual**.

Timing Verifi- cation

After the schematic is entered and checked, it is useful to check critical paths for timing verification. VLSI's timing verifier allows you to determine if your design functions properly at your clock speed, and that all critical paths meet their specified timing. This step is optional. Refer to the **VLSI Timing Verifier Manual** for more information on timing analysis.

Simulation

Simulation generally takes much longer than the schematic entry phase of the design, so you should allow enough time for a thorough simulation. Use VLSI's mixed mode simulator, VTIsim, to run your simulations as described in the **VTIsim Manual**. VTIsim creates waveform plots and a trace file, [trc].

There are two kinds of simulation:

- System Simulation -- Both Logic and Timing simulations; exercise the circuit as if it were in the system. There are no restrictions on simulation timing.

- Test Generation Simulation -- Exercises the circuit functionally. Since the results are used to generate test vectors, this simulation must subscribe to VLSI's super-synchronous guidelines.

Maximum clock speed for the test generation simulation for cell-based designs is 1 MHz. If the chip is intended to run at a higher speed, run system simulations at this higher speed and then adjust the test to 1 MHz to create test vectors. Test vector generation is covered in the POST-PHYSICAL DESIGN VERIFICATION chapter in this manual, and in the **Test Generation Guidelines**.

Normally 10,000 vectors are sufficient to test a chip. If more than 10,000 vectors are required, extra charges may be applied. Check with the Technology Center to determine the extent of these charges.

Create a set of at-speed timing and test program engineering simulations, where the pre-route performance of the chip is predicted by simulating with predicted wire capacitances. Create the simulation driver using the VLSI mixed-mode simulator for each of the sub-blocks, as described in the **VTIsim Manual**, then create a test driver for the entire chip. Document the results of this simulation.

Netlists

There is no interlock mechanism to ensure that the hierarchical netlist (HNL) files used for your simulation are up to date. Periodically delete all HNL files and recreate new ones by loading the top-level schematic into the simulator, especially before you start the physical design, and before final logic simulation. Re-simulate and re-verify this final netlist.

NOTE: Avoid deleting your state machine and datapath netlist files, as they take considerably longer to regenerate.

Logic Simulation

External Capacitances

Include external capacitances in all your simulations. These are the off-chip loads for full chip simulation, and the output loads for lower level simulations.

Asynchronous Logic

By default, VTIsim is run only in slow (worst case) mode. If you use asynchronous logic, run it in fast (best case) mode as well as slow mode. This can be approximated by using the command `set simparms delayFactor .25` in VTIsim. For any critical timing questions, Spice simulation is recommended. In addition, contact the Technology Center for help in test program development for asynchronous logic.

Simulation Checks

Be sure to use the following checks at the logic/timing simulation stage, as well as earlier in the design cycle:

- Static Checks -- Perform static checks in VTIsim on the top-level [hns] files. Record the results in a log file. Static checks include `check inputs`, `check outputs`, `check pushpull`, `check usage`, `show power`, `show syn *vdd*`, and `show syn *vss*`.
- Toggle Checks -- Use the `set toggles` command. Run the full chip simulation. Use the `show toggles` command. Examine the nodes for any that have not been toggled. If necessary, add test vectors and iterate.

The toggle check does not report interior model nodes. To obtain detailed fault coverage information, see the **Fault Simulation** section, in this chapter.

- Internal Node Load Checks -- Load the top-level [hns] into VTIsim. Execute the `show cap *` command. Investigate any internal node with unusually high capacitance; usually, any capacitance greater than 2 pF should be checked. Add extra buffers or use higher drive buffers where necessary. This check may result in lengthy listings. The netlist screener can also be used; this provides a more detailed analysis of loading and ramps.
- Clock Tree Check -- Use the timing verifier to check the clock tree. If this is not possible, use the simulator to examine the clock tree, making sure that it is balanced and that all clock polarities are correct. Visually examine the schematics. Load the [hns] file into the simulator. Start at a clock input and proceed down the tree using the `show after` and `show synonym` commands and looking for anomalies. Watch for polarity errors that may be missed by simulation.
- Schematic Check -- Review the schematics with the Technology Center engineer.

Timing Simulation

Timing simulation allows you to determine if your design functions properly at your specified clock speed and that all critical speed paths meet their specified timing.

Estimate Routing Capacitance

Use estimated routing capacitance while simulating your design. To add estimated routing capacitance to your netlist, use the proper `simparms` command and parameters in the simulator. Chip compilation has many variables for routing and is an iterative process. It is vital to estimate performance at the schematic simulation stage to save on expensive and time-consuming chip compilation iterations, especially for circuits over 16,000 devices/4,000 gates, or for circuits that have strict critical paths.

There are special simulation parameters that can be set in the simulator to estimate interconnect capacitance; they are the default capacitance and incremental capacitance. An example is:

```
set simparms defCap 0.05
set simparms incCap 0.20
```

The system default is `defCap .010` and `incCap 0.0`. The incremental capacitance is in units of pF per functional pin. Using our example, if an output (1 pin) goes to five inputs (5 pins), the interconnect capacitance -- in addition to the gate and overlap capacitances which are included in all functional model instances -- is going to be:

$$0.05 + (1 + 5) * 0.20 = 1.25 \text{ pF}$$

Provide Output Load Capacitance

Include the external load capacitance that the output buffers will see in your design application. To load your output buffers, use the `set capacitance` command in VTIsim. The following example loads output RAS with 50pF and output CAS with 150 pF. Default units for the simulator are picofarads.

```
VTIsim> set capacitance 50 RAS
VTIsim> set capacitance 150 CAS
```

The `set capacitance` command overrides any previous capacitance value assigned or computed for a node. If the load on RAS is currently 5 pF, and you give the `set capacitance 50` command, the load on RAS becomes 50 pF, not 55 pF. If you wish to increase the existing load, `set capacitance` to the total value desired.

Bidirectional Pins

ON BIDIRECTIONAL PINS, USE set charged. The `set input high` and `set input low` commands override anything the pad tries to do, and can hide contention or design problems.

Worst Case Timing Can Hide Best Case Hazards

All of VLSI's portable library models use worst case process timing in the models. You should analyze your circuit for race conditions caused by best case processing. One possibility is to use the VTIsim `set simparms delayfactor n` statement, where *n* is the best case derating factor for process, temperature or voltage factors. Derating factors are given in the **General Information** section of the standard cell library manuals. If necessary, contact the VLSI Technology Center for assistance.

Interconnect Capacitance Calculations

This section describes how interconnect capacitance is calculated by the mixed-mode simulator, the design review program, the netlist extractor, and the logic compiler.

Simulator

If a post-route capacitances file, [pst], is loaded, VTIsim uses the capacitance value given in the node's `capacitance` statement. If there is no [pst] file, interconnect capacitance is predicted using this formula:

$$C = A + N * B$$

where:

`C` = interconnect capacitance

`A` = the `defCap` value given in the `set simparms defCap` statement.

`N` = The number of model and gate pins on the node, including the pin driving the node.

`B` = the `incCap` value given in the `set simparms incCap` statement.

The `defCap` and `incCap` values are normally set by loading [sim]predcap, which is a one-line simulation command file that does a `set simparms` command. There is a different [sim]predcap in each technology's logic design library.

Design Review Program

The formula used by VTireview to calculate post-route interconnect capacitance is:

$$C = L1 * CM1 + L2 * CM2$$

where:

C = The interconnect capacitance in fF.

$L1$ = The length of the metal1 wire in microns.

$CM1$ = The capacitance per micron for metal1 in fF.

$L2$ = The length of the metal2 wire in microns.

$CM2$ = The capacitance per micron for metal2 in fF.

$L1$ and $L2$ are read from a wire length file, [wr1].

For standard cells, the capacitance-per-micron factors are found by multiplying the hardcoded gate array capacitance factors by the standard cell wire widths divided by 4. For gate arrays, $CM1$ and $CM2$ are hardcoded to 0.255 and 0.171, respectively. These values may be altered by defining new values in your `vt1.bo` startup file, as shown in this example:

```
define gaCapPerLength1  0.274
define gaCapPerLength2  0.186
```

For predictive wire lengths, the calculation is:

$$L1 = L2 = 100 + 250 * N$$

where N is the number of macro pins on the node.

Netlist Extractor

The extractor calculates capacitance by multiplying the areas and perimeters of wires by the plate capacitance and sidewall capacitance values in the [etf] file. It also adds a metal1/metal2 overlap capacitance.

Logic Compiler

The logic compiler uses the same plate and sidewall capacitance factors as the extractor, but adds a factor to these to account for overlap capacitance, instead of calculating the exact overlap. The factors used are in the [tch] file, documented by comments in the file.

Fault Simulation

Fault simulation provides an automated algorithmic measure of a test program's ability to exercise nodes in your circuit. VLSI Technology Centers, as a service, offer assistance in performing fault simulation.

Using your design netlist and simulation vectors, VLSI runs either the GenRad HILO3 software simulator or the Silicon Solutions MACH 1000 hardware accelerator, and provides you with a detailed report of the results, as generated by VLSI's fault simulation post processor. The report includes fault coverage and counts of detected, not detected, and undetectable faults. Additionally, VLSI provides a file of undetected faults, a cross-reference file, a netlist screen file, and the original fault simulator output file.

Assistance in analyzing the results is also provided to help you determine if the fault coverage is acceptable; otherwise, additional vectors are needed. VLSI can also assist you to enhance your fault coverage.

The steps in performing a fault simulation are:

1. Translate the netlist to the fault simulator format using VTIexchange.
2. Translate vectors to the fault simulator format using VTIVector.
3. Perform fault simulation.
4. Run VLSI's fault simulation post processor to generate a more useful results file.
5. Review the results of the fault simulation with the customer. Make any necessary changes to test vectors or logic to improve the coverage.

Fault simulation is an optional step.

**Logic
Design
Review**

When simulation is complete, the files and completed forms listed on the **Logic Design Review Checklist** are assembled for the joint Customer/VLSI logic design review. If everything is satisfactory, the customer and VLSI engineers sign the **Statement of Work** for all completed milestones.



CHAPTER 7

PHYSICAL DESIGN

This chapter describes the physical design task using VLSI's chip compiler and other physical design tools.

Chip Compiler

Using The Chip Compiler

VLSI's chip compiler works with standard cell and arbitrary functional blocks. An arbitrary block does not follow the predefined standard cell layout rules, while a standard cell block contains standard cells of fixed height. An arbitrary block can be a compiled cell, a megacell, custom layout, a datapath element implemented as compiled layout, or a state machine element implemented as a PLA. A standard cell block can be a standard cell subcircuit, or a datapath or state machine element implemented as standard cells.

When you load your netlist into the chip compiler, it places your arbitrary blocks as separate functional blocks, and initially puts all your standard cells in one block (Figure 39). After the initial placement, you can divide the standard cells up into a number of different areas, and move the various standard cell blocks and arbitrary blocks around until you get a good floorplan.

You can convert your design to composition editor [cp] format, or layout editor [ly] format, to connect any incomplete routing and do additional manual editing.

Before You Begin

Regenerate Your Netlists

To ensure that your netlists match your most current schematic, delete all HNL files and recreate new ones from your top-level schematic. Avoid deleting your state machine and datapath netlist files, however, since they take considerably longer to regenerate.

Check V6 Blocks

Before loading your netlist into VTChipComp, you should check any arbitrary blocks that were created with Version V6 of VLSI's tools.

While VTChipComp uses gridless block routing, connectors must be spaced so that a via can be placed in front of any two adjacent connectors without causing a design rule violation. For CMN20A (VSC10), for example, this means a center-to-center separation of 8 lambda or more for connectors with a width of 4 lambda or less. For connectors that are wider than 4 lambda, the minimum center-to-center separation is larger. VTChipComp only allows connectors on two routing layers, metal1 and metal2 for CMN20A, and it does not support corner connectors.

Unfortunately, the VLSI Tools Version V6 compiler cells and megacells do *not* obey these spacing rules. The V6 version of VTLogicComp also does *not* produce cells that obey these rules.

Use `other` → `check macro cell!` in VTLayout or VTCompose, or `checkMacroCell` in Utility ChipComp to

check all arbitrary blocks to make sure the connectors obey the VTIchipComp spacing rules. Any violation of the spacing rules is reported.

If you started a chip design in V6, you need to rerun the cell compilers in V7 or manually fix all the connector spacing problems. The megacells and compiler cells have been fixed for Version V7, and VTIllogicComp and VTIchipComp in V7 produce cells that obey these spacing rules.

Create Phantoms

Using the non-graphic shell environment or a VTITerminal window running `shell!`, use utility `mcpconv` to generate `[mcp]` phantoms for all of your arbitrary blocks before loading your netlist. An arbitrary block is any functional block that you implemented as compiled silicon. This utility is not necessary for megacells, which already have `[mcp]` cells, or compiled cells, which automatically generate the appropriate `[mcp]` cells.

An example of an `mcpconv` session, from a VTITerminal screen running `shell!` or from the shell environment, is:

```
VTI> util mcpconv
Enter input physical cell (default type is [cif]): addchip
Reading input physical cell.
Writing output [mcp] cell.

VTI>
```

When there are multiple physical implementation types for a cell available, VTIchipComp takes an `[mcp]` cell rather than a `[cif]` (Caltech Intermediate Form), `[ly]` (layout) or `[cp]` (composition) cell. The `[mcp]` cells load into memory faster, so using them makes your netlist load faster. Generating the `[mcp]` also provides you with the elements you need to verify your design with phantom extract, DRC and netcompare.

NOTE: VLSI supplies a corresponding [mcp] for each megacell.

VTIchipComp does *not* automatically regenerate [mcp] cells if they are out of date. If you change the layout of your arbitrary block, you need to regenerate your [mcp] cells.

Flatten The Netlist

If the netlist you load into the chip compiler is a hierarchical netlist, usually [hns], the chip compiler uses the HNL flattener to flatten the netlist. Then the flattened netlist is loaded. If the netlist is a flattened netlist, usually [fns], then the netlist will simply be loaded.

However, a netlist that has been flattened for the simulator cannot be used by the chip compiler, because it deletes items that do not appear at the transistor level, such as the **weight** schematic and the power and ground pads. To provide a flattened netlist for the chip compiler, either load the [hns] into the chip compiler, which will flatten it appropriately, or use the HNL utility with special switches, as described in the GUIDELINES chapter of the **Chip Compiler Application Note**.

The flattener does not flatten the netlist all the way down to the transistor level. Instead, it stops flattening a cell if it is in the standard cell library or is listed in a file called **place.cls**. List the names of all your arbitrary blocks, and any custom cells that are not in the standard cell library, in this file. Use VTItext or a system text editor to create a file named **place.cls**, and list the arbitrary cell names, one per line, as shown in the example that follows.


```

c8255      # arbitrary block
it02d2a    # custom standard cell
shift4
counter4

```

NOTE: If you do not use the flattener -- that is, if you load an [fns] file -- the place.cls file is not read, and the chip compiler will not know about your arbitrary blocks.

Compile Your Cell

Basic Operations

The basic sequence of operations to compile a cell is:

- Create phantoms for your arbitrary blocks.
- Load the netlist.
- Set the general floor plan options.
- Run block placement to place all blocks.
- Manually modify the floor plan.
- Evaluate the floor plan.
- Set desired seed placements.
- Run initial placement to place standard cells, connectors, and pads.
- Run placement improvement on the initial placement of the standard cells; a maximum of three passes should be sufficient.
- Manually modify the standard cell placement.

- Run the router to connect up the standard cells in each standard cell area.
- Draw routing guidances and set desired widths for power nets and special signal nets.
- Use the `checkRouteBK` command to review the block routing.
- Run the router to connect all the blocks.
- If the netlist contains I/O pads, run the padding generator.
- Convert the finished cell to layout or composition cells.

Complete details on these operations are given in the **Chip Compiler Manual** and **Chip Compiler Application Note**.

Convert Your Layout

You need to convert your design to a physical layout format, either `[ly]` for the layout editor or `[cp]` for the composition editor. When the design is in a physical format, you can finish any incomplete routing, and do any additional manual editing that is required. Also, the design must be in a physical layout format in order to verify it with DRC and netcompare.

Standard Cell Areas

Convert standard cell areas to `[ly]` (layout) rather than `[cp]` (composition) format. Most standard cell areas are too big to be converted to `[cp]` format. The composition editor works best with about 20 instances per cell. It can be used for cells with 100 to 200 instances. Most large standard cell areas exceed this limit.

Large Chips

For large chips, use the **save on convert** option in the chip compiler's `set up` property sheet. This option causes those cells being converted to [ly] format to be directly written to the disk without creating a cell in memory. This reduces the memory utilization and speeds up the conversion process for big chips.

Another possibility for large chips is to use `utility chipComp` in `VTIshell` to do the conversion. This reduces the memory required and makes the conversion run faster. The **save on convert** option defaults to `on` when you use `utility chipComp`.

If most of the Unimplemented Interconnect (UIs) in a large design are in the padding, convert the core to [ly] and the padding to [cp], and correct the routing in the composition editor.

Fixing UIs

In The Composition Editor

You can use either the layout editor or the composition editor to fix any incomplete routes. If you convert your standard cell areas to [ly] cells, then you can convert your core cell to [cp]. However, just using `route` → `all UIs!` and `compact` will not produce a working chip. Most of the UIs occur because there is no room to route them. You need to move wires and add jogs to create enough room to route these UIs.

Compacting large chips in the composition editor may be slow or even impossible, depending upon your hardware. For core-limited designs, compacting in the composition editor pulls all the pads to the bottom left corner.

Use the composition editor check connectors, check node names and check constraints commands to look for disconnects, design rule violations and nodes that are shorted together.

In The Layout Editor

In the layout editor, your UIs are converted to lines on the TEXT layer. If you fix UIs in the layout editor, be sure to use → to check the area around each UI you fix before running a DRC on the whole chip.

IMPORTANT: Any time you alter the physical design or manually connect wiring, you must check the changes with DRC and netcompare.



CHAPTER 8

POST-PHYSICAL DESIGN VERIFICATION

This chapter describes the back-annotation and post-physical design simulation, phantom DRC and netcompare, 1 MHz test vector generation, phantom tape out, and Final Design Review design tasks.

Back Annotation

In the chip compiler, back-annotate your design with routing capacitances as described in the GUIDELINES chapter of the **Chip Compiler Application Note**. This is done by loading your design and using the `misc` → `make PST!` command, which generates a `[pst]` cell containing the interconnect capacitances resulting from the routing wires. It also includes statistical allowances for fringing capacitance and other effects, for a more accurate simulation.

Post-Physical Design Simulation

Be sure that you have a netlist that is correctly flattened for simulation purposes. Refer to the **Netlists For Timing Verification and Simulation** section, in the SIMULATION chapter, for a discussion of simulation netlists.

Load your schematic netlist into the simulator, then load the [pst] file using the command `load filename.pst`. Resimulate the design using the back-annotated capacitances. Examine all of the critical paths carefully at this time.

Use the `show cap *` command. Investigate any nodes that have become heavily loaded due to routing capacitance. If necessary, modify the schematics and redo the place and route as necessary. Pay close attention to the clock tree and critical paths. Using `show cap *` may result in a very large file. This can be especially cumbersome if the line limit in the simulator is kept at its default value of 100 lines.

Complete The Physical Design

Before verifying the physical design, you need to generate the artwork for the logo, device number and revision level, trademark and copyright, layer numbers, and any other text you need to put on the final artwork. These should be added before you run the final netlist comparison and DRC.

In the graphic tools, bring up the cellLib window. In the browser, select the VCC10 or VCC100 library and the FOUNDRY category within it (Figure 40). Generate compiler cells for these elements, as described in the compiler cell library manual:

- CLOGO1 - The VLSI logo on the metall layer; an [1y] cell.

- CFNDEV - The device number and revision letter; a [tp1] cell. Obtain this number from the Technology Center.
- CTMARK1/CTMARK2 - The VLSI trademark in metal1 or metal2; an [1y] cell.
- COPYR1/COPYR2 - The copyright symbol and year in metal1 or metal2.
- CFNLAY - Layer numbers; a [tp1] cell.
- CTEXT - A text generator; a [tp1] cell.

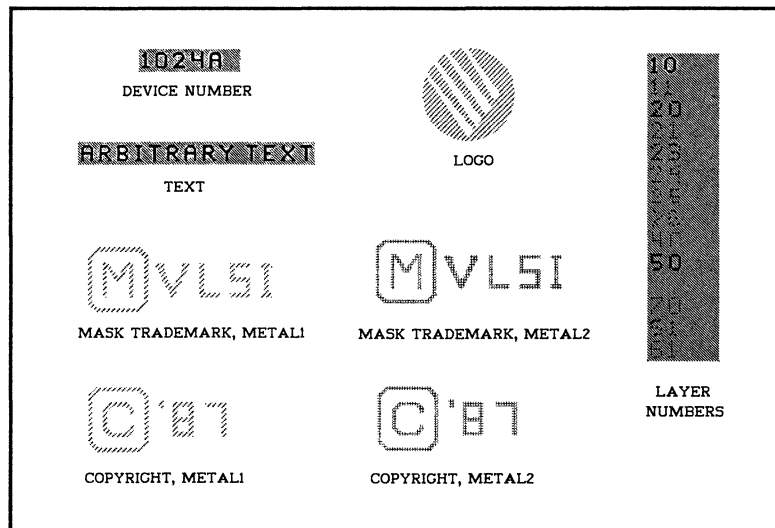


Figure 40. Foundry Artwork

Appendix C contains the data sheets for the CFNDEV, CFNLAY and CTEXT elements.

Output [cif] cells for the compiler cells you want. In the composition editor, place them in your top-level design at least 10 lambda from other geometries. Place the copyright line above CTMARK. Visually check all the placements.

Output the top level design from the composition editor as a [cif] cell.

Phantom DRC

After the physical design is completed, generate CIF for your design. The cells will be represented as phantoms: black boxes with connectors. The actual physical layout for these cells will be merged in later by the VLSI Technology Center engineer. Use the [cif] cell as input to the design rule checking program, DRC. You can run DRC from the shell environment, or in a graphic terminal window. Refer to the **DRC Manual** for detailed information on how to run a DRC and check the output.

Be sure that there are [mcp] cells for any datapath elements you have included in your design. For correct results, the phantom DRC needs an [mcp] cell rather than a [cif] cell when checking datapath blocks. The DRC automatically uses the [mcp] cell if it is present.

Phantom Net- compare

The netlist comparison program compares the netlist extracted from the physical layout of your design to the corresponding schematic netlist. VLSI's netlist extractor generates a netlist from your physical layout cell -- [cif], [ly] or [cp] -- that includes all the interconnect and capacitances. Refer to the **VTIextract Manual** for detailed information on extracting a netlist from physical layout.

For the schematic netlist, use an unflattened original netlist that refers to any datapath elements using the model schematic, `[mde]mdp(cellName)`.

In both netlists, the cells will be represented as phantoms: black boxes with connectors. Bring up the netcompare graphic window and run a phantom netcompare as described in the **Netlist Comparison Manual**.

Generate 1 MHz Test Vectors

All the vectors that were developed during logic and timing simulation were developed for at-speed simulation. In order to generate a prototype test program that can be used to deliver tested prototypes, you must create a set of low-speed, synchronous simulation vectors that can be converted into a test program.

The application note entitled **Test Generation Guidelines** describes VLSI's requirements for developing test vectors. This application note should be read and understood before you generate your test vectors. This section provides a short synopsis of the test vector requirements; you should refer to the application note for more information.

VLSI Technology provides a vector check utility that verifies that your test generation vectors adhere to the super-synchronous guidelines described in the **Test Generation Guidelines** application note. This utility is described in the **VTIcheck** manual.

To create the simulation vectors, prepare a standard 1 MHz test program driver file and run a simulation to generate either a dynamic or report-on-change tabular `[trc]` trace output file. Regular report-on-strobe tabular format does not have the necessary timing information in it. This trace file is used to create the test program for the specified tester.

There are three ways to create the required [trc] file, as shown in Figure 41:

- Use VTitest to drive VTIsim as described in the **VTitest Manual**.
- Use VTIsim commands entered manually or from a command file.
- Write a Mainsail .MS file to drive VTIsim as described in the **VTIsim Manual**.

In all cases, the resulting output [trc] file is used by VTIVector, run by a VLSI Technology Center, to create the test program.

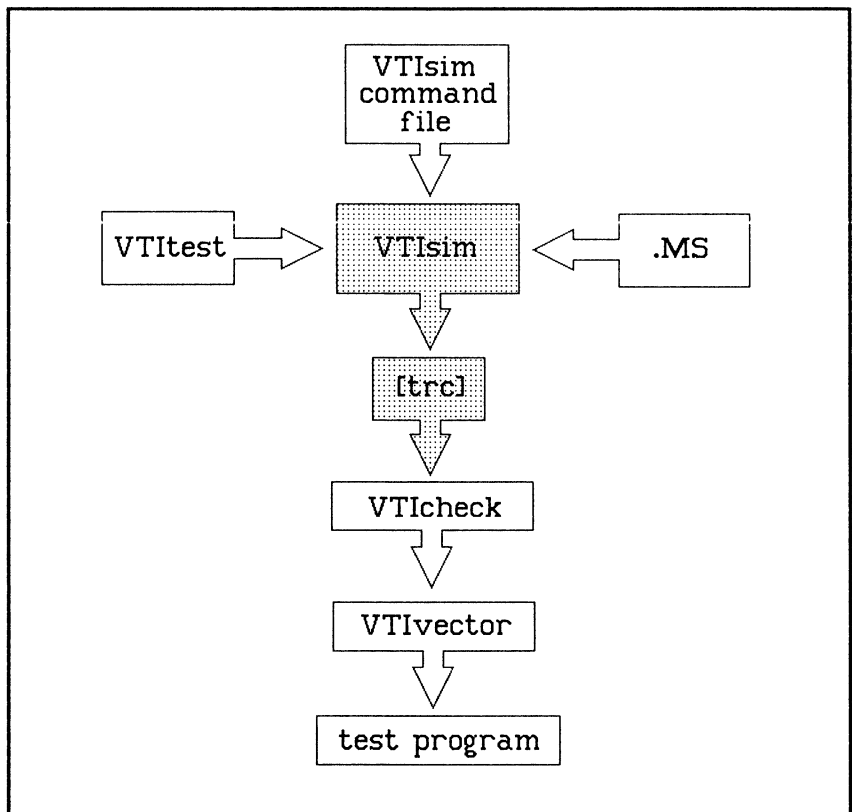


Figure 41. Trace File Generation

VLSI uses your simulation trace file, [trc], produced by VTIsim, and the [lds] file produced by VTIscreen or VTIcheck to generate the correct test vectors. An additional file used by VTIvector to re-verify that the vectors are synchronous is the [itg] file, produced by the vector checker, VTIcheck.

If you want your test program to contain a critical path check, an optional Critical Path Check [cpc] file may be specified.

**Super-
Syn-
chronous
Simulation
Guidelines**

See the application note entitled **Test Generation Guidelines** for more information on creating the [itg] and [cpc] files.

The super-synchronous simulation guidelines described in the application note **Test Generation Guidelines** are summarized in this section. These are the simulation guidelines used by VLSI Technology's vector checker, VTlcheck.

Test Vector Guideline Summary

- Run your simulation at 1 MHz.
- Limit the number of required timing generators to 6 or fewer.
- For a given input, all transitions must occur at the same edgetime throughout the simulation.
- Edgetimes for input transitions must be multiples of 100 ns, and must be between 0 and 800 ns.
- No two timing generators can share the same edgetime assignment.
- Inputs cannot change format during the simulation.
- Bidirectional nodes and bidirectional control nodes may have only one transition per cycle.

- If a bidirectional node transitions during the same cycle as its control signal changes, the control signal must transition before the bidirectional node.
- Avoid race conditions between paths.
- Include all external signals and all control signals on bidirectional and three-state outputs in your trace file.
- Submit only one trace file and limit your simulation to 10,000 cycles or less.
- Do not submit simulation vectors if there are any setup or hold violations, or any bidirectional conflicts.
- Begin your simulation with all internal signals in the unknown state.
- Design your circuit so that it can be initialized to a known state, and begin your simulation with vectors that initialize the circuit.
- Never force the values of internal signals during simulation.
- If you have any open drain or three-state outputs with pullups, simulate them with a rise time of 50 ns.
- Make each output go both HIGH and LOW during the first 1000 cycles.
- Submit an Input Timing Generator file.
- If you want critical paths checked, submit a Critical Path Check file. Critical paths cannot be checked within the first 5 cycles.

- If your design includes a block for which VLSI Technology supplies a test program, submit a Test Block Map file.

Checking Vectors

VLSI provides a vector checking utility, VTiCheck, that scans a set of trace vectors for adherence to VLSI's super-synchronous vector guidelines. These guidelines ensure that the vectors are easily convertible to a variety of testers. While checking the trace file for conformance to super-synchronous vector guidelines, VTiCheck creates the utility files that are required to convert the vectors.

VTiCheck requires only a trace file and netlist for inputs, and produces not only an error report, VER, but also a complete trace definition file, TDF, and input timing generator file, ITG as well.

How to Check Vectors

The VTiCheck utility examines vectors for conformity to an expected timing format. The timing format can be specified explicitly, by providing an ITG or TDF file, or the timing information can be derived from the trace vectors as they are checked.

If an lds file, output from VTIScreen, or an HNS netlist is supplied, VTiCheck creates a TDF file. Otherwise, a TDF file, either with or without explicit timing information, must be supplied as input. If a TDF file without timing information is supplied, the TDF file is modified to include timing information.

If for some reason, such as the unavailability of a netlist, it is not possible to create an lds file, the vector checker can still be used, but you must create a TDF file by hand. The syntax of the TDF file is given in the **VTIcheck Manual**.

Example

Be sure that your trace and lds files are available, then call VTIcheck by typing `utility check`, as shown in this example:

```
VTI> utility check
VTIcheck> input [trc]counter
VTIcheck> netlist [hns]counter
VTIcheck> itg [itg]counter (optional)
VTIcheck> duration 1000 (defaults to 1000 ns)
VTIcheck> source vlsi (defaults to VLSI)
VTIcheck> output counter
VTIcheck> create
#####
####          No Errors - Vectors are Super Synchronous          #####
#####
VTIcheck> quit
```

The `[ver]`, `[itg]` and `[tdf]` outputs all share the same cellname, given with the `output` statement.

Test Specifi- cation Form

Prototype And Production Test Specification Form

When the 1 MHz test vectors have been successfully generated, the customer and VLSI complete and approve the **Prototype And Production Test Specification Form**. Complete additional forms if there is more than one critical path to be tested.

Additional Guidelines For Test Program Generation

- Submit a Pad Placement Form before fabrication begins.
- Include a brief functional description for each device pin in the Pad Placement Form.
- When performing final simulations that will be used to test the device, do not place stimulus values on internal nodes with `check init`.
- Signal names used during final simulations must be identical to the signal names defined on the Pad Placement Form.
- Submit only one top-level trace file from the final simulation to the test engineer.

Phantom Tape Out

When the physical design and verification are completed, create a tape of the layout database in [cif] format, with the phantoms, and send it to the Technology Center. The Technology Center engineer reviews the package and then, upon customer approval, merges in the cell layouts and completes the physical design verification and the test program generation.

**Final
Design
Review**

When the post-physical design simulation is complete, the issues listed on the **Final Design Review Checklist** are reviewed jointly by the customer and the VLSI design engineer, to ensure that all possible problems have been resolved. The Tape Out Checklist at the end of the form ensures that all the data required to create prototypes has been submitted.

The **Performance Approval Form** is also completed at this time, and signed off by the customer and VLSI. This form authorizes mask generation and prototype production.

Finally, the customer and VLSI engineers sign the **Statement of Work** for all milestones that have been completed.

CHAPTER 9

IMPLEMENTATION

This chapter describes the implementation tasks in the cell-based design cycle, including test program generation, layout merge, physical design verification and simulation, CIF tape out, and final check.

Generate Test Program

All the test vectors developed using VTitest or the mixed-mode simulator should follow the 1 MHz guidelines. These vectors are then converted to a Sentry test program using VTivector at the VLSI Technology Center. If the design uses megacells, the Technology Center integrates the canned megacell test programs with the test vectors provided by the customer, as described in the **Megacell Test Development Methodology** application note. If there are any asynchronous designs requiring special test programs, they are developed manually or by using VTitest with the help of the Technology Centers.

Merge Cell Layouts

This step is performed by the Technology Center engineer, who uses the PHMERC utility to merge the layouts of the standard cells into the phantom structures. The `stdphy` library must be on the search path before running PHMERGE. PHMERC accepts either `[cif]` or `[ly]` files, and creates a file called `design_merge.cif`. This is the final, complete CIF database which is used to generate masks. It is very important to perform the full physical design verification on this database.

Certain compiled blocks are represented by a composite CIF with a model built in, for simulation purposes, because a transistor-level simulation of the block would be too time-consuming. The Technology Center engineer must be sure to have the actual full CIF for these blocks, rather than the composite/model file.

Physical Design Verifica- tion

Run the *standard* design verification programs on the full physical layout as described for the *phantom* DRC, extract and netcompare. Refer to the POST-PHYSICAL DESIGN VERIFICATION chapter for more information.

Design Rule Check

Perform DRC, using the merged physical layout, and iterate until the results are correct. Record the results in a log file.

Extract And Netcompare

Extract the netlist from the `[cif]` file using VTextract, as described in the POST-PHYSICAL DESIGN VERIFICATION chapter, and perform a netlist comparison against the schematic that was used for the full simulation. Refer to the **Netlist Comparison Manual** for complete information on performing a full physical netlist comparison.

In a full transistor-level netcompare, the schematic netlist only contains references to models (`mde`). In order to compare the interconnect within the cells, you need to switch in the flattened extract netlist, `fne`, for each `mde` referenced in the schematic netlist. This creates an equivalent transistor-level schematic. Although the netlists for layout and schematic are the same for standard cells, this netcompare checks for any problems created by wiring over a cell in the layout or composition editor. The netcompare catches any shorts in the CIF file that result from this wiring.

Do not use an `hnc` netlist from the composition editor for this netcompare. The netlist is based upon the center line of the wire and does not contain wire width information. For example, if two wires are 3 microns apart, and each wire is 10 microns wide, the `hnc` represents them as separate, but the CIF shows a short.

Do not use the flattener utility to generate the `fne` by switching a logical phantom (`lph`) to an `fne`, as the flattener drops the pad node name in the pads.

Repeat the netcompare until the results are correct. Record the results in a log file.

Supply the starting identifications manually; do not use `auto`, which can hide VDD and VSS shorts and opens.

The netcompare should result in 0 errors. However, there may not be 100% identification, and all unidentified nodes must be examined and understood. For example, compiled cells, such as RAMs, often contain large numbers of internal nodes that cannot be identified.

Netcompare should report that the physical netlist has the same number of VDD and VSS nodes that the chip is to have.

The number of schematic VDD and VSS nodes is not significant. If netcompare results indicate that any layout changes need to be made, the verification loop has to be repeated.

Visual Check

Make a large plot of the final CIF and review it visually.

Simulation

Load the top level [fne] file into VTIsim and perform static checks. Record the results in a log file. Static checks include:

- check inputs
- check outputs
- check pushpull
- check usage
- show power
- show syn *vdd*
- show syn *vss*

If there are compiled instances in the chip, static checks may result in lengthy listings.

Use the VTIsim commands `set power high` and `set power low` to set power and ground at the pads *only*: do not use the wild cards `*vdd*` or `*vss*`. Using the exact pad names rather than wildcards prevents power shorts and opens from being masked. By driving the power and ground

only at the pads, power and ground bus connectivity is thoroughly checked and floating wells are flagged.

CIF Tape Out

Customer Package Marking Form

Complete the **Customer Package Marking Form** and attach it to the **Chip Specification Form**. Be sure to sign the form.

CIF Tape Out

Review the physical layout DRC and netcompare logs, [hns] static checks, [fne] static checks, CIF plot, specification documentation, tooling form and CIF tape out form with the Technology Center engineer. Create a tape of the full chip CIF file and send it to the Technology Center.

Send the test vectors, including the [tst], [vec] and [s10] or [s20] files, to the VLSI test engineer. If you used VTItest, the pindef section of the VTItest [tst] file should have the same pinout as the chip. If you have not already done so, sign the *Test Program Done* portion of the **Statement of Work**.

Archive The Design

After Tape Out, archive the database according to instructions from the Technology Center System Administrator. The Administrator is given an archive tape, and the VLSI engineer keeps a second archive tape, along with a copy of the specifications and a complete set of the schematics. Appendix D gives a recommended outline of project tape and binder documentation for archiving.

**Final
Check**

Review the prototype packaging with the VLSI Product Engineer. Inform the Product Engineer and Test Engineer about any unusual requirements for chip testing. If VTitest was used, and the chip pinout is not identical to the **pinDef** section, notify these engineers.

**Mask
Generation**

The VLSI Mask Layout department generates masks from the final database after all physical design verification is complete and customer approval is obtained.

Prototypes

Prototypes are delivered to the customer in the prototype package specified. The customer evaluates the prototypes and, when functionality is approved, fills out the **Prototype Approval Form** and returns it to the Technology Center. Production can begin after this milestone is completed.

APPENDIX A

VLSI DESIGN TOOLS

The descriptions below summarize the functions of each of the VLSI design tools used in the cell-based design cycle. Each description is meant as an overview of that particular tool; for a more detailed description, refer to the appropriate users guide.

Cell Compiler - VTICellLib

The VLSI Cell Compiler is the interface to the VLSI Cell Compiler Library, and to the Datapath and State Machine compilers. With VTICellLib, you can specify parameter values and display parameter properties for compiler cells and timing models. Supported compiler cells include RAM, ROM, PLA and Multiplier. Outputs from the Cell Compiler are the physical representation (CIF) of the parameterized cell, an icon of it for use in the schematic editor, and a behavioral model of it for simulation in VLSI's mixed mode simulator.

Chip Compiler - Mixed Standard Cells and Multiple Blocks

The Chip Compiler is an integrated arbitrary block and standard cell placement and routing system. It provides a highly automated methodology for chip assembly. The system

allows you to place arbitrary blocks interactively, and to fill in the regions between those blocks with standard cells. Analysis aids are included to evaluate the efficiency of the chip's floorplan on the basis of interconnect routing density.

The Chip Compiler automatically sizes the defined standard cell regions to properly accommodate all the standard cells and their interconnections as they are listed in the design. After automatic placement of the standard cells, these regions and the interconnections between them are routed using a compaction algorithm to minimize chip area.

You can specify wire widths and routing guidance for interconnection paths with rough diagrams within the floorplan. The Chip Compiler follows this guidance to help optimize the floorplan and minimize the routing area for critical paths, such as power or highly propagated signals.

If the padding is specified in the design's schematic, the Chip Compiler automatically assembles it around the chip. The system evaluates the area of the design and selects the proper pad type based on the pitch for a core-limited or pad-limited design.

Composition Editor for Post-Route Editing - VTIcompose

The Composition Editor is a graphical chip assembly tool used for placement and routing of arbitrarily-sized rectangular blocks. As is the case with the Chip Compiler, each block can be one of several types, or a combination of custom cells and/or cells from the VLSI standard cell library, cell compiler library, or megacell library. Routing can be done manually or automatically from a netlist.

Once the design is placed and routed, you can use the VTIcompose compactor to minimize silicon area consistent

with the design rules of a given technology. The compaction process automatically displays the critical paths limiting further compaction, allowing you to optimize the design by rearranging blocks and interconnects with an extensive set of manual editing commands. VTIcompose can generate CIF for mask fabrication, and a hierarchical netlist, back-annotated with interconnect capacitances, for resimulation of the design.

Datapath Compiler

The Datapath Compiler is a silicon compiler used to help create repetitive portions of a design. It can generate the design in either a netlist format based on the VLSI Portable Library, or in a high-density layout form. The Datapath Compiler is good at implementing logic that is applied to each signal in a bus, such as in the execution unit of a computer. It is not restricted to computer design, however, and is useful in many areas of system design.

A datapath is specified by drawing a schematic using elements from the datapath library. This schematic is used as input to the compiler to tell it what elements are required and how they are to be interconnected. As is the case for other VLSI compilers, a parameter cell must be created to specify other necessary options. Functional units and bus interconnections are normally the specified word width; however, special operators can be used to reduce the number of bits. Arbitrary inter-bitslice routing is provided using the same special operators.

The Technology Center engineer can add cells to the datapath compiler in any of three ways: schematic only for implementation in either standard cells or as a gate array; layout, following VLSI's layout practices to add cells compatible with the existing cells from the datapath library; or layout to new rules using the compiler just to assemble the new cells.

With a specification schematic and a parameter cell, you can create a gate-level netlist suitable for gate array or standard cell implementation, a CIF implementation of the datapath, or a gate-level simulation and timing analysis model of the CIF implementation. An interface to the VLSI Design Assistant allows you to make fast tradeoffs when planning and partitioning a system containing datapaths.

Design Assistant

The Design Assistant evaluates the different implementation technologies such as gate array, standard cell, full custom and silicon compilation to identify the most efficient and effective method in which to implement a chip. Feasibility studies for different chip designs can be implemented at whatever level of design detail is available. Through the use of artificial intelligence algorithms, the system provides a list of design alternatives rationalized by power, size, and packaging requirements.

Design Rule Check - VTIdrc

VLSI's design rule checker verifies that geometric layouts conform to fabrication process constraints. For ease in editing, the errors can either be printed textually, plotted, or highlighted in the Layout Editor, along with the physical layout, for edit-in-place design corrections.

Exchange Netlist Formats - VTlexchange

VTlexchange is a program that translates netlists from one format to another. It has a screening utility to identify simple errors and other potential problems in your logic design. VTlexchange can read VLSI's HNS and FNS formats, GenRad's HILO, Mentor's MIF, Silvar-Lisco's SDL, and Tegas. VTlexchange can write VLSI HNS and FNS, HILO, Silicon Solutions' MACH 1000, SDL, Tegas, and Merlyn's VR.

Extract Netlist - VTlextract

VTlextract extracts a transistor-level netlist from the physical database. The extracted netlist contains transistor sizes, node names and accurate interconnect capacitances. It can be used as input to the Mixed-Mode Simulator, or compared to the netlist produced by the Schematic Editor with the Netlist Comparison program to verify circuit performance.

Fault Simulation - HILO3 and MACH 1000

Fault simulation is performed with either the GenRad HILO3 or the Silicon Solutions MACH 1000 software. Using these packages, the Technology Center runs either a complete fault simulation, which covers all of the faults in the fault set, or an incremental fault simulation, rerunning the fault simulation after incorporating additional test vectors.

The report includes fault coverage and counts of detected, not detected, and undetectable faults. Additionally, VLSI provides a file of undetected faults, a cross-reference file, a netlist screen file, and the original fault simulator output file.

Icon Editor - VTlicon

The icon editor is a symbolic editor which allows you to create custom symbols, called icons, to represent gate-level, transistor-level or higher-level cells in the schematic editor.

Logic Compiler for Standard Cells and One Arbitrary Block - VTllogicComp

The VLSI logic compiler provides a means of implementing a schematic netlist into a physical structure consisting of placed and routed standard cells and, optionally, a single arbitrary block. The schematic is loaded into the VTllogicComp window where it is automatically placed and routed. The Logic Compiler can either generate an entire chip, with pads; or a functional block that can be used with other functional blocks, such as compiled cells or megacells, to implement a

larger chip. Using the Logic Compiler, you can control the placement of critical paths by weighting nets or seed-placing cells. You can also control I/O placement and the shape of the functional block.

VTIlogicComp is optimized for double-metal routing. Second layer metal is routed over cells, based on automatically generated blockage masks. You can generate a back-annotated netlist that contains interconnect capacitances, and resimulate it to verify circuit performance. Functional verification can also be accomplished through the use of behavioral models.

This tool has been replaced by the Chip Compiler in V7.

Netlist Comparison - VTInetComp

VTInetComp takes the netlists produced from any two of the VLSI Tools and compares them to verify that they match. For example, a netlist from VTIschematic and one from the Chip Compiler can be compared to see if the physical layout does, indeed, represent the original logical data.

Review Program - VTireview

VLSI's Design Review program assists the designer of gate array or standard cell blocks in analyzing and simulating circuits. It offers the following major functions:

- Generates a pre-route or post-route design report.
- Generates a post-route back-annotation file which includes parasitic wiring capacitances.

Schematic Editor - VTIschematic

The schematic editor allows you to create, view, edit, annotate and plot hierarchical schematic diagrams of standard cells, megacells, gate arrays, compiled cells and custom cells. Its

output is a hierarchical netlist that can be simulated or netlist-compared for design verification.

Screeners - VTIscreen

The netlist screener is a program that identifies simple errors and other potential problems in the logic design. It also provides useful information about cell usage. VTIscreen reads the netlist of the design and performs the following calculations:

- Calculates the internal cell and I/O pad utilization and prints a summary report.
- Provides a Design Statistics report listing information on the number of nets and FromTos - connections between two pins - in the design.
- Provides a Design Complexity report listing gate equivalents per cell.
- Checks for certain design and connectivity errors and other potential problems in the design.
- Optionally writes a Logical Design Structure [lds] file to be used by VLSI's design review or vector conversion programs.

Simulator - VTIsim

VTIsim can simulate a mixture of gate-level, transistor-level and behavioral-level models, which simplifies design verification against pre-defined specifications. With VTIsim, you can simulate interactively with immediate textual and graphical feedback, or simulate in batch mode. Waveforms can be displayed during the simulation and/or plotted from a trace file produced by the simulation.

State Machine Compiler

The State Machine Compiler is a logic synthesizer that provides a fast and simple means of generating the control and state logic of a chip. The state machine is specified in a high-level, state-transition language. You write equations for each state to generate outputs and to change to a new state. If you want to check the specifications, the compiler accelerates the process by simulating the state machine directly from its specifications, without taking the time to create a final netlist.

When the design is ready, the compiler translates the state machine specifications into an internal gate form which it can implement either as a netlist for a gate array or standard cells, or as a PLA layout.

Test/Simulation Description Language - VTitest

VTitest allows you to describe the physical characteristics, timing information and expected response values for a circuit. VTitest uses a high-level test description language that automatically translates input information into commands that can be executed by VTIsim. The output can be used to generate a complete test program for the Sentry series of IC testers.

Timing Verifier - VTItv

The VLSI timing verifier is intended for a designer who is designing a medium-to-high performance part. No knowledge of transistor-level design is required to use this tool. Since static timing analysis is performed, no input or case vectors are given and no assertions have to be added to the circuit to be verified.

The timing verifier is intended to work with synchronous systems: systems that are composed of alternating levels of storage elements and combinatorial logic. All clocks must be

explicitly declared; the verifier is not able to derive clocks from counter chains. It is also not able to verify the correctness of self-clocked or data-clocked logic.

The timing verifier should be used to find potential timing problems in the design after the design has been entered with the schematic editor and its primary functionality has been verified with simulation. Using the verifier to find and rectify timing problems decreases some of the simulation burden. Among the checks made by the timing verifier are:

- Whether the delay along a given path is within specified tolerance.
- If there are any set up and/or hold violations.
- Whether the clock skew between any two clock signals is within a specified tolerance.
- Which are the critical paths in the circuit.
- Cycle time of the circuit.
- Delay along any specified paths.
- Minimum period for any clock.
- "Worst" paths: the paths with the greatest or least delay.

Vector Conversion Program - VTIVector

VTIVector is a general-purpose vector conversion and comparison program. Using VTIVector, a designer can automatically convert a simulation trace file into the input driver format of another simulator, or create test vectors for any of the tester formats VTIVector supports.

Vector Checker - VTlcheck

VTlcheck checks 1 MHz test simulation vectors for conformity to VLSI's super-synchronous vector guidelines for creating prototype test programs.

APPENDIX B

PACKAGE PIN INDUCTANCES

This appendix lists the package pin inductances for the semiconductor packages offered by VLSI Technology.

S/B PKG.	Pin Inductance - S/B Packages			
	Low (0 - 5 nH)	Medium Low (5 - 10 nH)	Medium (10 - 15 nH)	High (15 - 20 nH)
22-pin S/B	2, 3, 7-10, 13, 14, 18-21	1, 4, 5, 11, 12, 15, 16, 22	6, 17	
24-pin S/B	2-11, 14-23		1, 12, 13, 24	
28-pin S/B 250x250	4-11, 18-25	2, 3, 12, 13, 16, 17, 26, 27	1, 14, 15, 28	
28-pin S/B 310x310	6-9, 20-23	1-5, 10-19, 24-28		
28-pin S/B 350x350	4-11, 18-25	1-3, 12-17, 26-28		
40-pin S/B	8-13, 28-33	2-7, 14-19, 22-27, 34-39	1, 20, 21, 40	
48-pin S/B	10-15, 34-39	4-9, 16-21, 28-33, 40-45	1-3, 22-27, 46-48	
64-pin S/B		10-23, 42-55	2-9, 24-31, 34-41, 56-63	1, 32, 33, 64 (23.4 nH)

CPGA PKG.	Pin Inductance - CPGA Packages			
	Low (0 - 5 nH)	Medium Low (5 - 10 nH)	Medium (10 - 15 nH)	High (15 - 20 nH)
68-pin CPGA	9, 26, 43, 60	All pins except 9, 26, 43 and 60		
84-pin CPGA	1, 22, 43, 64	All pins except 1, 22, 43 and 64		
120-pin CPGA		1, 2, 4, 5, 7-26, 28, 29, 31, 32, 34, 35, 37-56, 58, 59, 61, 62, 64, 65, 67-86, 88, 89, 91, 92, 94, 95, 97-116, 118, 119	3, 6, 27, 30, 33, 36, 57, 60, 63, 66, 87, 90, 93, 96, 117, 120	
180-pin CPGA	23, 68, 113, 158	1, 10, 11, 16-18, 22, 24, 28, 31, 46, 63, 64, 66, 70, 71, 91, 100, 101, 106-108, 112, 114, 118, 121, 136, 153, 154, 156, 160, 161	2-9, 12-15, 19-21, 23, 25-27, 29, 30, 32-45, 47-62, 65, 67, 69, 72- 82, 84-90, 92-99, 102-105, 109-111, 115-117, 119, 120, 122-135, 137-152, 155, 157, 159, 162-172, 174-180	83, 173

PDIP PKG.	Pin Inductance - PDIP Packages			
	Low (0 - 5 nH)	Medium Low (5 - 10 nH)	Medium (10 - 15 nH)	High (15 - 20 nH)
20-pin PDIP	2-9, 12-19	1, 10, 11, 20		
40-pin PDIP 180x180	8-13, 28-33	2-7, 14-19, 22-27, 34-39	1, 20, 21, 40	
40-pin PDIP 200x200	6-15, 26-35	2-5, 16-19, 22-25, 36-39	1, 20, 21, 40	
40-pin PDIP 260x266	2-19, 22-39	1, 20, 21, 40		

PLCC PKG.	Pin Inductance - PLCC Packages			
	Low (0 - 5 nH)	Medium Low (5 - 10 nH)	Medium (10 - 15 nH)	High (15 - 20 nH)
28-pin PLCC	All pins			
44-pin PLCC	1-4, 9-15, 20-26, 31-37, 42-44	5-8, 16-19, 27-30, 38-41		
68-pin PLCC		All pins		
84-pin PLCC		All pins		

PKG.	Pin Inductance - LLCCC and LDCCC Packages			
	Low (0 - 5 nH)	Medium Low (5 - 10 nH)	Medium (10 - 15 nH)	High (15 - 20 nH)
44-pin LDCCC	1-5, 8-16 19-27, 30-38, 41-44	6, 7, 17, 18, 28, 29, 39, 40		
68-pin LLCCC		All pins		
84-pin LLCCC	1, 21, 22, 43, 44, 64, 65, 84	2-10, 13-20, 23-31, 34- 42, 45-52, 55-63, 66- 73, 76-83	11, 12, 32, 33, 53, 54, 74, 75	

APPENDIX C

FOUNDRY CELLS

CFNDEV - REV 1.1
 CMOS STANDARD DEVICE IDENTIFICATION NUMBER
 TEXT CELL

DESCRIPTION

CFNDEV is a text generator cell, used to place the device number of a particular chip on selected mask levels. The device number (typically four digits, with an optional one or two character revision/device variation extension) and the magnification (size of the text) are parameters to the cell.

PARAMETERS

PARAMETER NAME	PARAMETER DESCRIPTION	DEFAULT VALUE	RANGE MIN/MAX
FNDEV_device_number	4 digit device number, 0-2 character revision/variant	0	N/A
FNDEV_magnification	Size of text. (Width, in lambda, of geometries within text.)	2.L	1 / **
FNDEV_draw_all_layers	See note (1)	FALSE	N/A

** No real range.

Note (1) - When FNDEV_draw_all_layers is FALSE, the text is drawn on PWELL, NWELL, P diffusion, N diffusion and METAL1 layers. When FNDEV_draw_all_layers is TRUE, the text is drawn on all layers except the passivation layer, CG. Text is never drawn on the passivation layer.

CELL SIZE

For FNDEV—magnification = 2.0L,

WIDTH = 90 lambda, HEIGHT = 20 lambda

NOTES: The abutment box for the cell is 3 lambda beyond the maximum size of the actual geometries. There is an EXCLUDE layer around the entire cell. This EXCLUDE layer prevents the DRC program from checking the internal geometries (text) and thus prevents false DRC errors.

USING THE DEVICE NUMBER CELL:

The parameters to this cell are 1) the device number of the circuit, and 2) the width of the geometries used to generate text on the mask layers.

The cell assumes that the device number is 4 digits, optionally followed by up to 2 revision letters. This format is the VLSI standard. The device number field is padded by blank characters to make the length 6 characters.

CFNLAY - REV 1.1

CMOS MASK REVISION LETTER TEXT CELL

DESCRIPTION

CFNLAY is a text generator cell, used to place the revision letter of each mask layer within a particular circuit. This cell is used to track changes that affect only one or a few mask layers. Unlike other CMOS cell compilers, this cell requires that the engineer knows which physical masks are affected by changes in drawn layers. A chart illustrating drawn layers and their corresponding physical layers is included to make this easier.

PARAMETERS

PARAMETER NAME	PARAMETER DESCRIPTION	DEFAULT VALUE	RANGE MIN/MAX
FNLAY_magnification	Size of text. (geometry's width within text, in lambda)	2.	1 /**
FNLAY_uses_metal2	TRUE: Output mask revision for metal 2 and via. FALSE: No metal 2 or vias used in this circuit. If other cell compilers are used, this must be TRUE.	TRUE	N/A
FNLAY_mask_10_rev	Diffusion mask revision	*	N/A
FNLAY_mask_11_rev	Field Dope mask revision	*	N/A
FNLAY_mask_20_rev	Vtn mask revision	*	N/A
FNLAY_mask_21_rev	Vtp Adjust mask revision	*	N/A
FNLAY_mask_23_rev	P-well mask revision	*	N/A
FNLAY_mask_24_rev	N+ diffusion implant mask revision **	*	N/A
FNLAY_mask_25_rev	P+ diffusion implant mask revision	*	N/A

PARAMETER NAME	PARAMETER DESCRIPTION	DEFAULT VALUE	RANGE MIN/MAX
FNLAY_mask_26_rev	N-well mask revision	*	N/A
FNLAY_mask_40_rev	Polysilicon mask revision	*	N/A
FNLAY_mask_50_rev	Contact mask revision	*	N/A
FNLAY_mask_51_rev	Via mask revision	*	N/A
FNLAY_mask_60_rev	Metal 1 mask revision	*	N/A
FNLAY_mask_61_rev	Metal 2 mask revision	*	N/A
FNLAY_mask_70_rev ++	Oxide passivation mask revision ++	*	N/A

NOTE: * - means no revision letter, implying first silicon mask.
 ** - derived from p+ diffusion
 ++ - text on mask layer 70 is outputted on the top metal layer.

USING THE MASK REVISION CELL:

Any alphanumeric character is acceptable as a parameter value.

The size of the cell is 62 by 267 lambda. Note however that the abutment box for the cell is 3 lambda beyond the maximum size of the actual geometries. Also note that there is an EXCLUDE layer around the entire cell. This EXCLUDE layer prevents the drc program from checking the internal geometries (text), and thus prevents false drc errors.

The particular masks used to fabricate VLSI's CMOS circuits are derived from the drawn layers -- a given drawn layer may influence several masks. The table below should be consulted to determine which mask or masks are affected by a change to a particular drawn layer:

Drawn layer	MASK layer number													
	10	11	20	21	23	24	25	26	40	50	60	51	61	70
P-well		*			*									
N-well		*						*						
P diff	*			*		*	*							
N diff	*		*											
poly									*					
cont										*				
metal1											*			
metal2													*	
via												*		
res		*			*			*						
pad														*

CTEXT - REV 1.1 TEXT GENERATOR

DESCRIPTION

CTEXT generates user-specified text strings. The drawn layer of the text and the size of the text geometries are cell parameters. This cell is used to imprint miscellaneous text on a drawn layer.

Note only one layer at a time can be imprinted with text and only one layer can be specified in the text—layer parameter. If no layer is specified, the default layer is imprinted with the text string specified. If more than one layer requires text, this cell can be compiled again with the next layer and text string to be imprinted specified.

PARAMETERS

PARAMETER NAME	PARAMETER DESCRIPTION	DEFAULT VALUE *	RANGE Min/Max
TEXT_STRING	Text string to be placed on a mask.	n/a **	n/a
TEXT_MAGNIFICATION	Size of letters and numbers	2.0 +	n/a
TEXT_LAYER	Layer name	CM +	n/a

* Cell specific parameters are specific to the cell and should be changed to match the design if the default value does not meet the design criteria. These values are in lambda units unless noted otherwise.

** If space characters (blanks) are needed between words in the text string, mark them with the character (underline). This character generates a blank.

+ CM for metal, CND for N diffusion, CPD for P diffusion, text for text layer and CP for Poly. Text on passivation (CG) layer is illegal. The compiler does not generate a cif for the CG layer.

APPENDIX D

RECOMMENDED ARCHIVE DOCUMENTATION

This appendix lists the recommended project files and documents to be included in a project's archive package.

Pre-Design Documents

- Quote from Design Center to Sales*
- Quote from Sales to Customer*
- Copy of the contract between VLSI Technology and the Customer, if applicable*
- Customer's logic and/or block diagram*

Device Specifications: final versions, signed off

- VCxxxx specification*
- Packaging specification*
- Marking specifications*

- Bonding diagrams*

Device Schematics

- Hierarchy tree of the design*
- Top-level schematic and schematics of all submodules

Logic Simulation

- Driver files and simulations of all schematics, where applicable
- Driver files for extracted device simulations
- Tabular and/or timing output logic verification simulation
- Tabular and/or timing output device verification simulation
- ROM/PLA/PAL code files
- Logic design/verification customer signoff

Transistor-level circuit simulations

- Simulation schematics
- All pertinent ASPEC or HSPICE simulation input/output files

Physical Design

- Plots of all custom CIF/layout cells used in the design

- A complete copy of the physical library used to build the device
- Top-level data base and Extract files
- Calculation data
- Physical simulation customer signoff

Post-Design Documents

- Design schedules, history, and statistics*
- Billings forecast/actual worksheets*
- VTItest/VTIvector output for test program generation
- Special considerations, problems, explanations, etc.*
- Directory of archive tape contents*
- Project archive notes and checklist form*

Prototypes

- Prototype approval
- Prototype and production test waivers
- Characterization results

* Required to be in binder in hardcopy form

INDEX

- A**
- AC power dissipation 34
 - AC/DC Specification Form 43
 - aliases 81
 - ambient temperature 38
 - arbitrary blocks
 - definition 103, 106
 - from V6 105
 - list file 107
 - archive the completed design 131
 - asynchronous
 - designs 127
 - logic 65, 95
 - at-speed simulation 94
- B**
- back-annotation
 - file 80
 - in chip compiler 113
 - bidirectional pins
 - set capacitance 98
 - block diagrams 43
 - bond form 86
 - bonding diagram 86
 - borders 65
 - buffers 72
 - clock 66
 - clock, weights on 71
 - fanout for, metal migration 67
 - bus repeater cells 69
- C**
- busses 65
 - bypass multiplexers 57
- C**
- capacitance
 - default interconnect 97
 - estimated routing 97
 - external 95
 - fringing 113
 - incremental interconnect 97
 - interconnect
 - calculated by design review program 99
 - calculated by logic compiler 101
 - calculated by netlist extractor 100
 - calculated by simulator 99
 - interconnect calculations 99
 - internal node check 96
 - output 98
 - output diffusion 66
 - output load 38
 - post-route 113
 - to minimize 71
 - to set 98
 - to set for bidirectional pins 98
 - cell compilation 108
 - cell library window 61
 - cell manager 60
 - cell-based elements 1
 - cellLib 114
 - cellLib window 61

CFNDEV 115
 CFNLAY 115
 checkjlist
 final design review 125
 checklist
 logic design review 102
 chip compiler 97, 103
 chip manager 60
 Chip Specification Form 42
 chipcomp utility 105, 110
 CIF
 cell 116
 cell, final 124
 database, final 128
 tape out 131
 CIF tape out 131
 circuit initialization 56
 clock
 input timing 44
 input waveforms 44
 skew 66
 clock buffers 66
 weights on 71
 clock skew 66
 clock tree check 96
 CLOGO1 114
 compile a cell 108
 compiler cells
 create functional block with 61
 fault coverage 52
 library 11
 composition cell 109
 composition editor 105
 connectivity errors, screener check 78
 connector
 at top level 75
 names 65
 rules 105
 contention 68, 98
 controllability 51, 54
 COPYR 115
 copyright 114
 core block 75
 core-limited layout 33

D

cp cells 105, 109
 CPC file 119
 critical path 123
 Critical Path Check file 119
 Critical Path Description 46
 critical paths 114
 CTEXT 115
 CTMARK 115
 custom cells 61, 81
 Customer Package Marking Form 43, 131
 datapath elements
 create functional block with 62
 fault coverage 52
 library 14
 datapath model schematic 117
 DC current leakage 35
 default capacitance 97
 defCap 97, 99
 delay
 clock input rise/fall 66
 rise/fall 66
 worst 81
 delay factor, to set 98
 derating factors 98
 Design Assistant 42
 design complexity report, screener 78
 design cycle 2
 design entry and simulation flow 4
 design report 80
 design review program 80
 design reviews 29
 design statistics report, screener 78
 design summary 80
 design tasks 9
 design types 2
 design verification flow 8
 DRC
 window 111
 DRC, phantom 116

- DRC, physical 128
drive, signal 66
drivers
 pad 73
 simulation 117
dynamic trace file 124
- E** economic considerations 29, 32
editing layout 105
elements, cell-based 1, 10
estimated routing capacitance 97
external capacitances 95
extract netlist 116
extract, physical 128
- F** fanouts for metal migration 67
fault coverage 52, 101
fault simulation 101
fault simulation post processor 101
final check 132
final design review 29, 125
final design review checklist 125
flattened netlist
 for chip compiler 107
 for post-physical design simulation 114
 for simulation 93
 for simulator 89
floating wells 131
floorplan 103
flow
 design entry and simulation 4
 design verification 8
 implementation 9
 physical design 6
 system planning 3
fne 129
fns
 for chip compiler 107
 netlist 89
foundry artwork 114
- frequency, operating 38
fromTos 78
functional blocks
 creating 59
 placement 74
 test 48
- G** gates required 38
generate test program 127
guidance, routing 104
guidelines
 design for testability 57
 testability 53
- H** hierarchical designs 59
hierarchical netlist 89
HILO3 101
hnc 129
HNL
 files 105
 keeping current 94
 to make 89
 utility 93, 107
hns
 for chip compiler 107
 for datapath layout model netlist 91
 for datapath portable netlist 91
 for screening 78
 for state machine portable netlist 90
 netlist 89
- I** I/O pads 75
icon, to make 60
impedance, thermal 40
implementation flow 9
incCap 97, 99
incremental capacitance 97

initialization
 circuit 56
 initialization, circuit 72
 input connections 80
 instance names 65
 integration 41
 interconnect capacitance 99
 internal nodes
 buffering 66
 load check 96
 simulating 56
 testing 57
 timing 66
 unidentified in netcompare 129

J joint design 2, 26
 junction temperature 40

L large chips 110
 layer numbers 114
 layout cell 109
 layout editor 105
 lds file 78, 119, 122
 lds file, to create 80
 level shifters 77
 library
 add to search path 60
 compiler cells 11, 61
 datapath 62
 datapath elements 14
 megacell 12, 62
 standard cell physical layout 128
 standard cells 10, 60
 load capacitance, output 98
 loads, off-chip 95
 logic design review 29, 102
 logic design review checklist 102
 logic simulation 93, 95
 logical design structure file 78
 logo 114

lph 129
 ly cells 105, 109

M MACH1000 101
 macro library 36
 Mainsail driver file 118
 make HNL 89
 manager window 60
 manual routing 105
 mask generation 132
 mcp cell 106
 mcp cell for datapath DRC 116
 mcpconv utility 106
 megacell phantom cell 106
 megacells
 create functional block with 62
 fault coverage 52
 library 12
 testing 72
 merge phantoms 124
 metal migration 67
 mixed-mode simulator 93
 model schematic, datapath 117
 model-schematic output
 for compiler cell 62
 for datapath 63-64
 models, simulation 98
 modes, test 57
 MS file 118
 multiplexers, bypass 57

N names
 explicit 76
 for bidirectional nodes 76
 for open drain nodes 76
 for open source nodes 76
 for three-state nodes 76
 node 65
 signal in final simulation 124
 net weights 69

netcompare
 phantom 116
 physical 128

netlist
 extractor 116
 for screening 78
 for simulation 89
 regenerate for physical design 105
 schematic editor creates 89
 screener 68, 77-78
 to simulate compiler cells 89
 to simulate datapath elements 91
 to simulate megacells 89
 to simulate standard cells 89
 to simulate state machine elements 90

New Design Information Form 48

node names 65
 explicit 76

O

observability 51, 55
 off-chip loads 95
 operating frequency 38
 options simflatten 93
 output buffers 76
 output drivers 73
 output load capacitance 38, 98
 output pins required 38
 output portable netlist
 for state machine 90

P

package
 production 43
 prototype 43
 selection 33
 size estimate 43

pad drivers 73, 76
 pad placement 73
 Pad Placement Form 43, 124
 pad-limited layout 33
 padding
 generated by chip compiler 104

pads
 drive 73
 in top level design 75
 placement, specifying location 73
 power 73

pages, schematic 65

parallel logic 68

parameter cell
 for compiler cell 61
 for datapath 63
 for state machine 64
 used to simulate state machine 90

parameters
 compiler cells 61
 pad specification 73

parasitic capacitance 71

partitioning
 building block 40
 random logic 40
 subsystem 41
 system 31

PC boards, translating 53

pcl cell
 for compiler cell 61
 for datapath 63
 for state machine 64
 used to simulate state machine 90

performance approval form 125

phantom
 definition 6, 116
 DRC 106, 116
 extract 106
 merge 124
 netcompare 106, 116
 tape out 124
 to create 106

phantom merge 128

phmerg utility 128

physical design flow 6

physical layout formats 109

pinouts 131

pins required, output 38

place.cls file 107

portable netlist
 for datapath simulation 91
 for state machine simulation 90

post-route design report 80

power
 to core 73
 to padding 73
 power calculations 34
 Power Calculations Worksheet 38
 power dissipation 34, 36
 DC 39
 external 39
 internal 38
 total 39
 pre-route design report 80
 predictive routing capacitance 97
 preliminary design review 48
 priority, routing 69
 production package 43
 prototype
 approval form 132
 delivery 132
 package 43
 prototype and production test specification form
 123
 pst cell 113

Q quote number, VLSI 49

R repeater cells 69
 reserved names 65
 reset for storage elements 57
 review program 80
 rise/fall time, suggested 66
 RMS current 67
 routing 104
 capacitance 71
 estimated capacitance 97
 guidance 104
 incomplete routes 110
 manual 105
 priority 69
 UIs 110
 rp01d1 69

S save-on-convert option 110
 schematic cell
 datapath 63
 for compiler cell 62
 for datapath 63
 state machine 64
 schematic editor 65
 make netlist 89
 schematic pages 65
 scr file 78
 screener report file 78
 screening netlist 78
 search path, to add to 60
 Semiconductor Package Selection Guide 34,
 86
 Sentry test program 127
 set capacitance 98
 set simparms delayfactor 98
 set/reset for storage elements 57
 setParam for pad placement 73
 signal drive 66
 signal naming 76
 simflatten 93
 simparms 97, 99
 simulation 52
 1 MHz 94
 checks 95
 clock tree check 96
 driver file 117
 fault 101
 full physical design 130
 logic 93
 static checks 95
 test generation 94
 timing 93
 toggle checks 95
 simulator 93
 skew, clock 66
 standard cell
 areas 109
 block, definition 103
 library 10
 list file 107

T

standard cells
 create functional block with 60
 custom 61
 fault coverage 52
 startup file 60
 state machine elements 15
 create functional block with 64
 fault coverage 52
 Statement Of Work 49
 static checks 95, 130
 stdphy library 128
 storage elements, testing 57
 switch cell
 for datapath simulation
 layout model netlist 92
 portable netlist 91
 for state machine simulation 90
 synchronous logic 65
 system partitioning 31
 system planning flow 3

 tape out checklist 125
 tasks, design cycle 9
 TDF file for timing verification 122
 temperature
 ambient 38
 junction 40
 template cell
 for compiler cell 61
 for datapath 63
 for foundry 115
 for state machine 64
 test
 circuitry 72
 driver 94
 generation simulation 94
 modes 57
 patterns 52
 plan 49
 program generation 127
 vectors submitting 131
 test vectors
 submitting 124

testability
 definition 51
 features 47
 logic 52
 text vectors
 generate 117
 thermal impedance 40
 theta ja 40
 three-state nodes 69
 timing
 clock input 44
 I/O waveforms 45
 parameter table 45
 simulation 93, 97
 verification 80
 verifier 93
 worst case 98
 timing format, VTIcheck 122
 title block
 for datapath schematic 62
 for schematic pages 65
 toggle checks 95
 top level of design 75
 tpl cell
 for compiler cell 61
 for datapath 63
 for foundry 115
 for state machine 64
 Trace Definition File 122
 trace output file 117
 trace vectors 122
 trademark 114
 transient current 35
 translating a board 53
 trc file 117, 119
 turnkey design 2, 18, 77
 types of designs 2

U

UIs
 fixing 110
 unidentified nodes 129
 unimplemented interconnect 110
 user design 2, 23

user logic design 2, 21
 utility chipcomp 105, 110
 utility HNL 93, 107
 utility mcpcnv 106
 utility phmerg 128
 utilization summary, screener 78

V

VDD/VSS
 connections 80
 drive at pads 130
 nodes 129
 vector
 check 122
 limit 94
 vector error report 122
 visual check 130
 vti.boa file 60
 VTIcheck 122-123
 VTIexchange 102
 VTIextract 116, 128
 VTIschematic 65, 73
 VTIscreen 78
 VTIsim 93, 118
 VTItest 118, 131
 VTIVector 102, 118

W

waveforms
 clock input 44
 I/O 45
 weight icon 69
 weighting 69
 window DRC 111
 wiring capacitance 71
 worst case timing 98

Y

yellow-lines 77

[

[mde]mdp 117





VLSI TECHNOLOGY, INC.

VLSI Technology, Inc.
ASIC Division
1109 McKay Drive
San Jose, CA 95131
408-434-3100
TLX: 278807
FAX: 408-263-2511