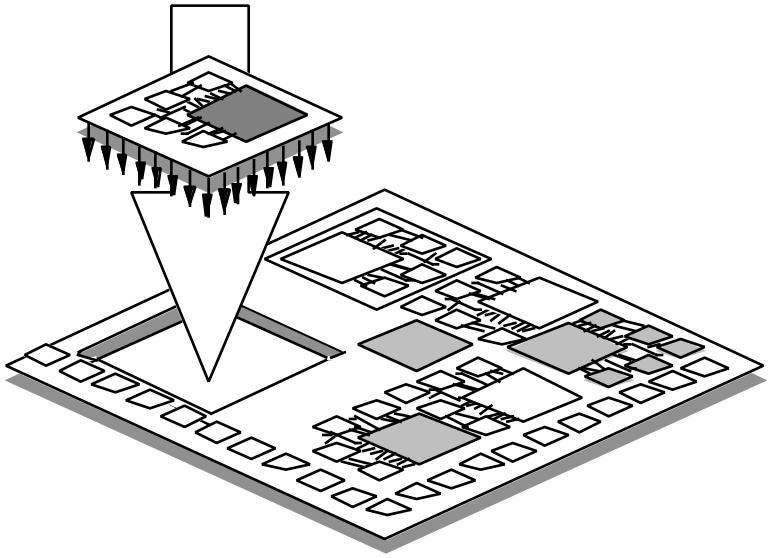
VSI Alliance™ Architecture Document



Version 1.0

Table of Contents

1. Overview

This document represents the baseline architecture for a proposed set of standards called the Virtual Socket Interface$_{TM}$.

1.1. Situation Analysis

As semiconductor technology advances, the business pressure to design large ICs in a short time increases. Design reuse is expected to be a prevalent method for improving design efficiency of large ICs. In many cases, the reused blocks are internally developed. However, even with the rapid advances in fabrication technology and design tools, few companies can dedicate the resources necessary to design and maintain all of the blocks required to offer the customer a total "system-on-a-chip" solution.

Consequently, it is becoming critical for companies to increase their access to a variety of functional blocks, both within companies (other divisions) as well as from other companies, to meet their time-to-market and business objectives. By doing so, each company can focus their limited design resources on areas where they provide maximum value while leveraging design expertise in blocks from the industry to satisfy their needs.

IC designs of the future will contain several reused functional blocks from several internal and external sources, mixed with some functional blocks designed specifically for that particular IC. For all these blocks to work together, it is necessary to define a common set of standards for the exchange of blocks.

Also, the existence of a design standard for the interchange of functional blocks creates the possibility of any design group becoming a provider of functional blocks, giving a company the option to sell or exchange their blocks with other companies in a more efficient manner.

The VSI Alliance was formed to support the needs of the industry for design reuse, and has begun to investigate how to reduce the technology and business barriers in order to accelerate the industry transformation. While there are a number of business barriers which need to be addressed, this document directly addresses some of the more important technical issues.

1.2. Document Overview

1.2.1. Goal

The goal of the VSI standardization effort is:

> To specify or recommend a set of hardware and software interfaces, formats, and design practices for the creation of functional blocks that enable the efficient and accurate integration, verification, and testing of multiple blocks on a single piece of silicon.

This document is the result of the first of a multiple stage evolution of standards for functional blocks, with changes occurring as the industry evolves and the methodologies change.

The goal of the first stage is to establish:

- a common nomenclature for the practice of design reuse,

- a baseline specification for the deliverables between functional block providers and integrators, and
- a context for future VSI standards.

1.2.2. Scope

The VSI Alliance intends to define, develop, ratify, test and promote open specifications relating to (1) data formats, (2) test methodologies and (3) interfaces, which will facilitate the mix and match and the reuse of intellectual property blocks from different sources in the design and development of system chips. The VSI Alliance does not intend to develop Specifications relating to the (a) internal design of intellectual property blocks, (b) functional architectures of subsystem components, (c) fabrication processes or (d) methods, algorithms or techniques for EDA tools.

1.2.3. Audience

This document targets two generic audiences:

1. Providers of functional blocks, and
2. Users — or integrators — of these blocks.

For the providers of functional blocks, it defines the documentation and data formats that users require for effective reuse of the blocks in an integrated product design. For the users of functional blocks compliant to this specification, it represents the information they can expect to receive in terms of content and formats.

1.2.4. Document Organization

This document contains the following sections:

- Section 1 provides an overview, including background, a summary of the contents and the goal for the document.
- Section 2 provides a detailed description of the requirements for documentation and data interchange.
- Section 3 is a set of preliminary VC guidelines intended as a seed to be reviewed by DWGs.

Figure 1.2-a graphically represents a generalized design flow for a complex chip design. It relates the development stages of the design (system specification, detailed block and system design, etc.) to the various disciplines or domains of design activity. Section 2 of this document is organized around those design domains - system design (Sections 2.2 and 2.3), logic design (Section 2.4), test design (Section 2.5) , and physical design (Section 2.6). Each entry in the table reflects a typical design activity for that stage and domain.

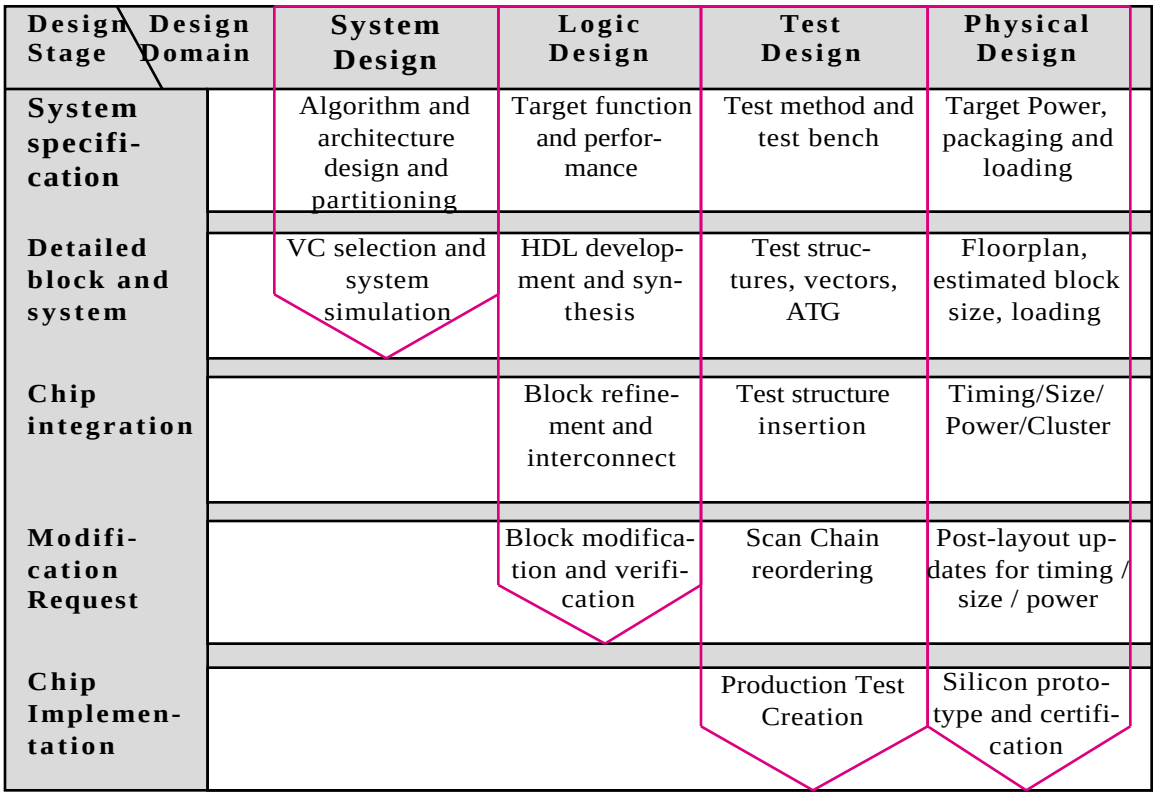


Figure 1.2-a Model of Complex Chip Design Flow

1.2.5. Relationship to Existing and Proposed Standards

This document and its subsequent enhancements are intended to specify recommended usage of existing standards and standards under development whenever possible. This standardization effort will be complementary to the activities of existing standards bodies.

1.2.6. Assumptions

There are some underlying assumptions which have been made while addressing this issue:

- Time is of the essence — a good short term solution, which will evolve, is better than a multiple year perfect solution.
- The proposed standard needs to be supported and endorsed by a number of vendors from the systems, semiconductor, and EDA segments.
- In keeping with the first two goals, the proposed standard needs to be both open and in common use.
- The standardization will prioritize issues to maximize impact on near term IP interchange and integration.

In preparing this proposal, it has been assumed that virtually all new digital designs originate from an HDL description. With increasing complexity, it is encouraged that future designs also include, where practical, a behavior description or model to facilitate inter-block verification..

1.3. Definition of Terms

1.3.1. Basic Definitions

Virtual Socket Interface™	A set of proposed standards and interfaces to enable system-level integration on a chip using pre-designed blocks resulting in rapid development of product. This enables IC design to be done using a component based paradigm.
Virtual Component™ (VC)	A block that meets the Virtual Socket Interface Specification and is used as a component in the Virtual Socket design environment. Virtual Components can be of three forms — Soft, Firm, or Hard -- which are defined below.
Intellectual Property (IP)	The term “Intellectual Property” means products, technology, software, etc. that have been protected through patents, copyrights, or trade secrets.
VC Creation	Process by which a block is designed to a set of specifications. The output should be in a standard format with a pre-defined set of characteristics which will simplify the integration and verification.
VC Integration	The process by which a designer combines and/or reuses multiple Virtual Components to create a much larger IC.

1.3.2. Hardness

One parameter for characterizing VCs is the “hardness” of the block, or the degree to which the VC has been targeted toward a particular fabrication process.

Soft VC	Soft VCs are delivered in the form of synthesizable HDL, and have the advantage of being more flexible and the disadvantage of not being as predictable in terms of performance (i.e., timing, area, power). Soft VCs typically have increased Intellectual Property protection risks because RTL source code is required by the integrator.
Firm VC	Firm VCs have been optimized in structure and in topology for performance and area through floorplanning/placement, possibly using a generic technology library. The level of detail ranges from region placement of RTL sub-blocks, to relatively placed datapaths, to parameterized generators, to a fully placed netlist. Often a combination of these approaches is used to meet the design goals. As indicated in Figure 1.3-a, Firm VCs offer a compromise between Soft and Hard. More flexible and portable than Hard, yet more predictive of performance and area than Soft, Firm VCs include a combination of synthesizable RTL, reference technology library, detailed floorplan, and a full or partial netlist. When a full netlist is present, it is expected that test logic has been inserted and that test lists will accompany the design. Firm VCs do not include routing. Protection risk is equivalent to that of Soft if RTL is included and is less if it is not included.

Hard VC	Hard VCs have been optimized for power, size, or performance and mapped to a specific technology. Examples include netlists fully placed, routed, and optimized for a specific technology library, a custom physical layout, or a combination of the two. Hard Virtual Components are process/vendor specific and generally expressed in GDSII. They have the advantage of being much more predictable, but consequently are less flexible and portable due to process dependencies. Hard VCs require, at a minimum, a high level behavioral model, a test list, full physical and timing models along with the GDSII. The ability to legally protect Hard VCs is much better because of copyright protections and there is no requirement for RTL.

Figure 1.3-a depicts a graphical representation of a design flow view, and summarizes the high level differences between Soft, Firm, and Hard VCs.

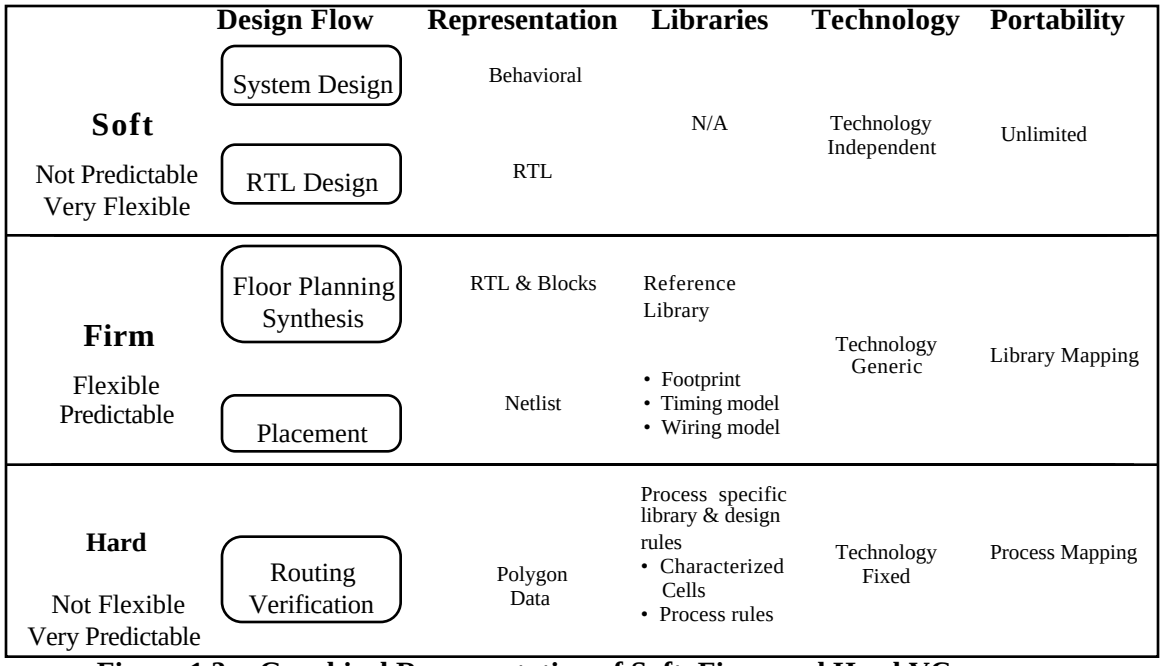


Figure 1.3-a Graphical Representation of Soft, Firm, and Hard VCs

1.4. Methodology Overview

As design complexity increases, so too must design reuse of existing functional blocks. To realize real improvements in productivity and time-to-market, large designs require a set of higher level building blocks. One way to visualize this shift is to adopt a printed circuit board-like model where standard, pre-existing components (or blocks) are assembled, simulated, verified, and ultimately manufactured.

Designs can be divided into two basic areas: block creation and block integration. Extending the board paradigm, block creation involves the development, or authoring, of blocks which are then re-used in either a stand-alone component or as part of a larger chip. Block integration is the activity associated with integration and verification of multiple blocks onto one piece of silicon.

Figure 1.4-a presents a graphical view of the VC creation and integration process and their relationship to VSI.

Creators of VCs go through the design and verification flow on the left side of the figure. The creator flow proceeds as far as necessary, depending on the hardness of the intended deliverable.

Integrators of VCs go through the design and verification flow on the right side of the figure. The integrator is responsible for completion of the chip design through the release to manufacturing (as defined by the manufacturer's requirements).

The data exchanged between the VC creator and the VC Integrator is the domain of the VSI standard.

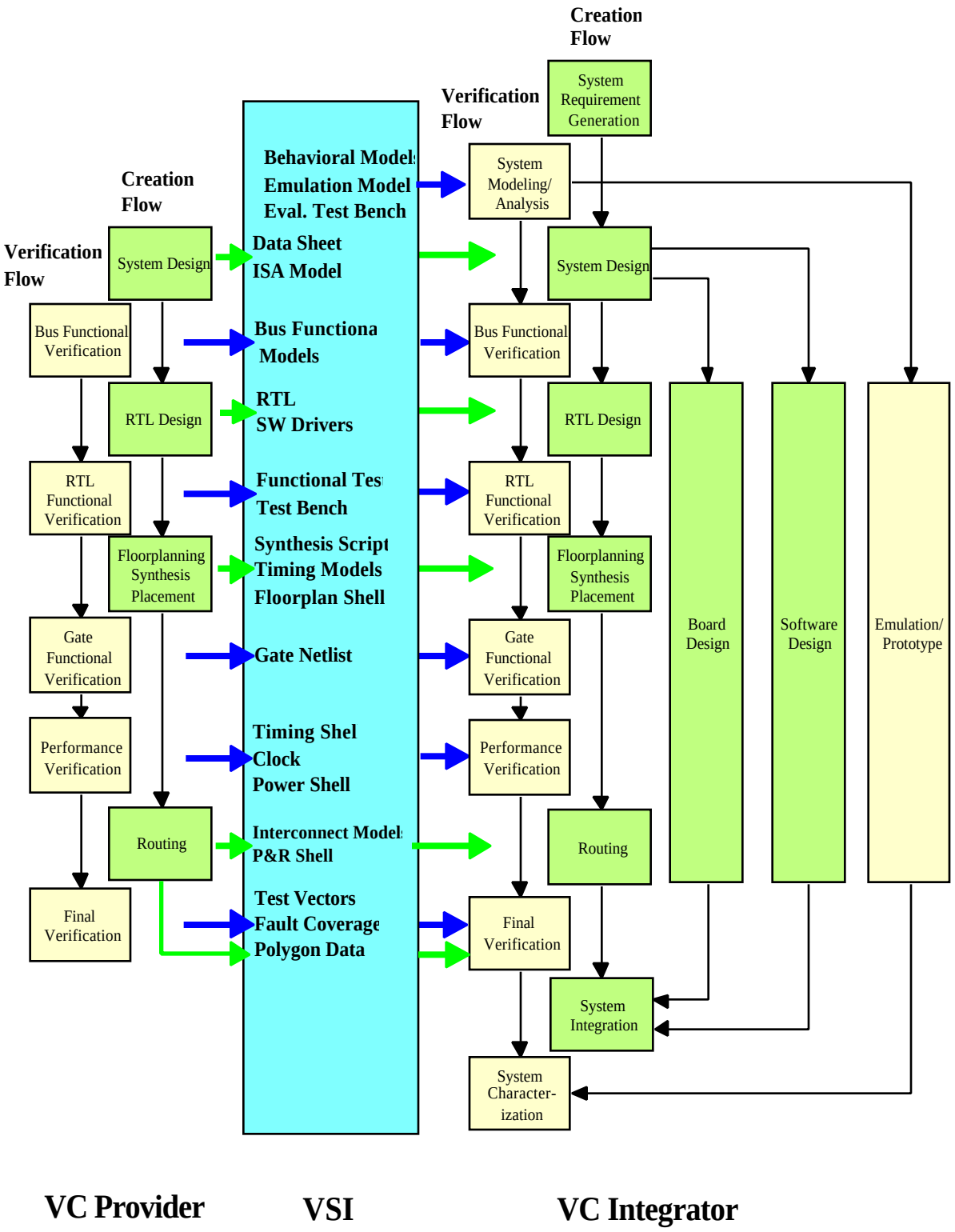


Figure 1.4-a Virtual Component Design Methodology

1.5. Data Interchange

1.5.1. Format Specification Philosophy

The VSI standard identifies information required to enable block-based system integration onto a single piece of silicon. While some of this information comes in the form of documentation, much of it comes in the form of executable models or machine-readable design descriptions.

The goal of the VSI Alliance is to specify a complete interface which:

- provides a practical, reliable link between VC provider and VC user
- provides an evolution consistent with tool, process, and technological advances
- specifies the use of an industry standard open format whenever applicable for executable or machine-readable models

The VSI Alliance advocates convergence among competing formats to reduce the number of recommended alternatives.

Where no standard exists, or where the standard has not yet been widely integrated, the VSI identifies the information that must be exchanged without specifying a particular format.

The VSI Alliance will only recommend proprietary formats whose owners have agreed not to assert rights within the field of use as defined in the VSI standard.

1.5.2. Deliverables Summary

Table 1.5-a summarizes the initial proposal for data formats. The discussion in this section is applicable to tables in Section 2.

The following list describes the columns of the table and the keywords used in that column:

Section references the document section which discusses the deliverable

Deliverable describes the deliverable

Currently Used Formats lists the formats used today as common practice in delivering VCs.

- **Document** Information exchanged on paper or electronically in a widely used format
- **ASCII** Information must be exchanged in a machine- and human-readable format

Candidate VSI Format lists open formats that are under consideration for inclusion in future VSI standards. The decisions of the standard formats will be made by a VSI Alliance DWG.

- **TBD** denotes that no candidate has been selected. The VSI Alliance will take one of the following actions:

1) Actively pursue a candidate within a DWG

2) do not pursue a candidate at this time

Soft, Firm, Hard describe the applicability of the deliverable to a VC of this hardness.

M denotes mandatory

CM denotes conditional mandatory (requirement is based on application)

R denotes recommended

CR denotes conditionally recommended (requirement is based on application)

Comments adds clarifying information

Table 1.5-a VSI Data Deliverables

Section	Deliverable	Currently Used Formats	Candidate VSI Format	Soft	Firm	Hard	Comments
2.1	**User Guide**						
2.1.1	Specification	Document	TBD	M	M	M	
2.1.2	Claims and Assumptions	Document	TBD	M	M	M	
2.1.3	Verification of Claims	Document	TBD	M	M	M	
2.1.4	Version History	Document	TBD	M	M	M	
2.1.5	Known Bugs	Document	TBD	M	M	M	
2.1.6	Application Notes	Document	TBD	R	R	R	
2.2	**System Architecture**						
2.2.1	System Evaluation Model	C, C++, VHDL, Verilog	TBD	CR	CR	CR	
2.3	**System Design**						
2.3.1	Verification Test Bench	VHDL, Verilog +PLI	VHDL, Verilog +PLI	R	R	R	
2.3.2	Behavioral Model	C, C++, VHDL, Verilog	C, C++, VHDL, Verilog	R	CM	M	OK to use RTL if available
2.3.3	Processor Models	C, C++, VHDL, Verilog	C, C++, VHDL, Verilog	R	CM	M	
2.3.4	Bus Functional Model	VHDL, Verilog	VHDL, Verilog	R	R	R	
2.3.5	Bonded Out VC/Prototype			R	R	R	
2.4	**Logic Design**						
2.4.1	Synthesizable RTL Source	(VHDL, Verilog) Synthesizable subset	(VHDL, Verilog) Synthesizable subset	M	CM	CR	Netlist could be used
2.4.2	Synthesis Constraints	Synopsys DC Shell	TBD	M	CM	-	
2.4.3	Floorplanning						
	Floorplanning Shell	ASCII, LEF	TBD	M	M	M	Inter-block
	Floorplanning Constraints	PDEF, LEF, SDF	TBD	R	CM	-	Intra-block
2.4.4	Structural Netlist	Verilog, VHDL, EDIF-netlist,	Verilog, VHDL, EDIF-netlist	-	CM	CR	Synthesizable RTL could be

Section	Deliverable	Currently Used Formats	Candidate VSI Format	Soft	Firm	Hard	Comments
		SPICE					used
2.4.5	Basic Delay Model	TLF, Synopsys NLDM, ITL, MMF	TBD	R	M	M	No current state dependent block format
2.4.6	Peripheral Interconnect Model	SPEF	TBD	-	CR	CM	
2.5	**Test Requirements**						
2.5.2	Test methods	Document	TBD	M	M	M	
	Open-box Test Description	Document	TBD	M	CM	-	If the VC faults fall into open box category
	Black-box Test Description	Document	TBD	-	CM	M	
	Test Patterns	WGL, VCD, or ASCII	TBD	-	CM	M	Dependent upon Test Method
	BIST	WGL, VCD	TBD	CM	CM	CM	Large memory arrays
	Delay Tests	WGL, VCD	TBD	-	R	R	
	Reliability (Burn-in)	WGL, VCD	TBD	-	R	R	
	Failure Analysis & Fault isolation	TBD	TBD	-	R	R	
2.6	**Physical Block Implementation**						
2.6.1	Block description	GDSII, LEF/DEF, SPICE	TBD	-	CM	CM	Estimated for Soft and Firm
2.6.2	Pin list/placement	LEF/DEF	TBD	-	CM	M	Required if Hard is netlist based
2.6.3	Porosity/blockage file	LEF/DEF	TBD	-	CM	M	
2.6.4	Footprint	LEF	TBD	-	CM	M	
2.6.5	Power/ground	LEF/DEF/ ASCII	TBD	-	CR	R	
2.6.6	Power model	TBD	TBD	R	R	R	
2.6.7	Physical Netlist	SPICE	TBD	-	CM	CM	

1.5.3. Proprietary Formats

The intention of the VSI Alliance is that all interface formats referenced in VSI standards must be open and available within the field of use defined by the standards. However, for this document, we have listed known proprietary formats in the "currently used formats" column. To become an open standard, the VSI Alliance must agree that the interface is appropriate and the owner of the standard must agrees to release it.

Table 1.5-b lists the proprietary formats referenced in Table 1.5-a and their owners.

Table 1.5-b Proprietary Formats

Format	Description	Owner
DC Shell	Design Compiler Scripting Language	Synopsys
DEF	Design Exchange Format	Cadence
SPEF	Standard Parasitic Extended Format	Cadence
GDSII	Polygon Level Layout format	Cadence
ITL	Interpolated Table Lookup cell-level timing model	Mentor Graphics
LEF	Layout Exchange Format	Cadence

MMF	Motive Modeling Format	Viewlogic
NLDM	Non-Linear Delay Model cell-level timing model	Synopsys
TLF	Table Lookup Format – cell-level timing model	Cadence
VCD	Verilog Change Dump	Cadence
WGL	Waveform Graphical Language	TSSI

This section addresses the deliverables between VC creators to VC integrators. At this time, the document does not address any issues regarding the generation (manual or automatic) of VCs. However, the deliverables created by a generator must meet the requirements of the standard.

1.6. Design Guidelines

Integration of VCs into a system chip can be made difficult, or fail, for many reasons: inadequate documentation, lack of system level models, electrical mismatches, layout mismatches, mismatch of tool usage assumptions, multiple test protocols, and multiple block interface protocols to name a few. Section 2 discusses the requirements and recommendations for deliverables (such as documentation and system level models). A design guidelines document will discuss some of the most common mismatches and will present guidelines for avoiding them.

2. Specification of Deliverables

2.1. User Guide

2.1.1. Specification

Specifications for VCs are generally in human readable form and provided by the VC creator. They describe functional operation, performance, timing requirements, operating modes, and acceptable operational regions of temperature, voltage, gate size, critical paths, and recommended floorplans. The format includes text, charts, tables, block or flow diagrams, and other graphical information. The actual outline and content of the data sheet will be highly dependent on the VC type and function. Information contained in the datasheet is generally applicable to Soft, Firm, and Hard VCs.

The VC provider needs to describe the socket requirements for such things as: test access interface, test vectors, and test methods that are to be applied. This section enumerates the types of tests, identifies a proposed standard for each, and maps the VC into the test architecture interfaces described in Figures 2.5.1. The VC provider must document the test methods available in the VC and the appropriate information for each test method.

Section	Deliverable	Currently Used Formats	Candidate VSI Format	Soft	Firm	Hard	Comments
2.1.1	**Specification**	**Document**	**TBD**	**M**	**M**	**M**	
2.1.1.1	System Description	Document	TBD	M	M	M	
2.1.1.2	Block Diagram	Document	TBD	M	M	M	
2.1.1.3	Register Description	Document	TBD	CM	CM	CM	
2.1.1.4	Timing Diagram	Document	TBD	M	M	M	

Section	Deliverable	Currently Used Formats	Candidate VSI Format	Soft	Firm	Hard	Comments
2.1.1.5	Clock Distribution	Document	TBD	M	M	M	
2.1.1.6	Bus Interfaces & I/O Configs	Document	TBD	M	M	M	
2.1.1.7	Test Description	Document	TBD	M	M	M	
2.1.1.8	Integration Requirements	Document	TBD	CM	CM	CM	

2.1.1.1. System Description

The system description is in text format, and is written for the user (system designer or chip integrator) to gain an understanding of the functional operation and general applicability of the VC. It contains both implementation and operational information, and can range in complexity from a page or two for simple functions to possibly hundreds of pages for the complex microprocessors.

Programmable functions will need to contain information the user will require to develop needed programs and software. Typical sections are: data formats; instructions and commands; control modes; operating modes; test modes; special modes; and features. Also required is reference to appropriate software support tools such as models, compilers, linkers, and debuggers.

2.1.1.2. Block Diagram

Block, flow, and/or tree diagrams are generally required for descriptive reference. These diagrams are to provide information on the interaction of key sub-elements of the VC. These should describe:

- organization
- structure
- partitioning
- data flow
- critical interactions

2.1.1.3. Register Description

All externally visible states need to be described. All registers important to the understanding, implementation, operation, and programming of the VC need to be described in this section. This should include:

- register definitions
- bit mapping into registers
- register clocks
- access requirements for writing or reading these registers

CONDITIONAL: The requirements for register documentation is conditional on the presence of registers in the VC design.

2.1.1.4. Timing Information

The timing diagram is a graphical representation where the clocks, data interfaces, and status are illustrated, and valid conditions for the inputs and outputs are identified — such as on which edge the input is captured and on which edge the output changes. Also included are false path designations, timing errors which can be ignored, and set-up and hold conditions.

When an asynchronous input is used, sufficient information must be provided to avoid problems in verification and test of the VC. Critical information includes:

1. where the asynchronous circuits are used
2. asynchronous circuit structure
3. metastability probability associated with this logic

2.1.1.5. Clock Distribution

All critical information on clock(s) and clock distribution must be provided by the VC supplier. For example:

- It is recommended that Firm VCs include a detailed description of the clock distribution network as part of the deliverable.
- For Firm VC, if additional clock frequencies or phases are being developed in the VC from the clock I/O on the VC, they must be clearly defined and described in detail.
- The internal clocking distribution should be described for the Firm VC. At a minimum this should include internal clock delay and skew.
- Hard VCs must specify interface requirements for the clocks such as frequency range, duty cycle, rise and fall edges, active edges, relation to other clocks, slew rate, jitter, and clock stop/start conditions.
- Soft VCs should not include the internal clocking structures.

2.1.1.6. Bus Interfaces and I/O Configurations

This section describes the interfaces to the VC with the internal functionality assumed to be at the black-box level. It includes:

- pin list (I/O names, function, and location)
- bus interfaces and protocols
- memory and CPU interfaces
- bus cycle encoding
- interrupts

- data rates
- word formats
- clocking specifications

2.1.1.7. Test Description Summary

The VC provider needs to describe the socket requirements for such things as test access interface, test vectors, and test methods which are to be applied. This section enumerates the types of tests, identifies a proposed standard for each, and maps the VC into the test architecture interfaces. The VC provider must document the test methods available in the VC along with the appropriate information. Section 2.5 describes the test methods.

2.1.1.8. Integration Requirements

The integration requirements section includes:

- those requirements specific to certain VC types as opposed to general requirements. An example might be special noise isolation or power requirements
- non standard files and formats required
- explanation of any design condition that could be considered contrary to VSI standard guidelines

2.1.2. Claims and Assumptions

Information on features and characteristics of a VC must be provided by the VC supplier for user evaluation and selection prior to purchase. This will result in claims related to performance, size, number of gates/bits/transistors, power requirements, etc. The VC supplier has the responsibility of substantiating these claims. This requires the supplier to define the context to which the claims were evaluated and, therefore, validated. This context will be referred to in this document as the “reference environment.” The reference environment will consist of the generic technology library, design flow, design tools, options, parameters, and all other information used to implement the VC in this environment. With this reference environment the user should be able to duplicate the results and verify the VC supplier’s claims.

The VC provider will state the claims for the VC including a minimum of the following:

- functionality
- performance (measured or estimated)
- power usage (measured or estimated)

- implementation size or gates as appropriate to indicate area (measured or estimated)
- VSI Data formats table
- testability
- test suites and/or test streams

These claims represent the VC supplier's products to the system chip integrator. They will represent his value added, market place differentiation, and will serve the chip integrator in the process of choosing the appropriate VC to meet his needs.

2.1.3. Verification of Claims and Assumptions

This section describes the procedures such that the claims made by the VC supplier can be verified. For Hard VCs actual measured values can be used when available. However, for Soft or Firm VC types, estimated values must be developed by running the design through to completion against a reference environment as defined in section 2.1.2. The selection of a reference technology library is at the discretion of the VC provider. However, it must be available to the VC user in a form appropriate for comparison. Generally, the reference library selected should be available in the industry to permit user validation.

This reference environment must be defined by the VC supplier such that the user would be able to repeat the claims by duplicating the conditions specified in the reference environment. This provides an audit capability for the claims.

A document from the VC provider identifying the compliance to Table 1.5-a is required. It is recommended that it be in the form of Table 1.5-a with a column added for status of the deliverables.

2.1.4. Version History

The VC developer must supply a version history of the VC describing changes and version releases associated with the VC. This information should clearly identify the current level of release and be coordinated with the claims.

2.1.5. Known Bugs

A record of bugs, issues, or other problems that are known to exist in a VC must be kept and made available to the VC purchaser. This could also include work-arounds and plans for fixes.

2.1.6. Application Notes

The application notes should discuss the implementation and customization options available to a System Designer using the VC. For example, a Micro-Processor Unit (MPU) VC could offer different memory access mechanisms, such as burst access or pipeline access. The VC provider may provide an application note explaining different implementation options to the VC user in designing the memory subsystem unit.

2.2. System Architecture

2.2.1. System Evaluation Model

The goal of the system, or high-level, models is to allow the VC user to evaluate and select the various VCs which are to be used for the system chip. The evaluation within the system environment, tradeoff analysis, and subsequent decisions on items such bandwidth, function, and performance can be determined within this environment in the context of the overall system chip specification.

The high-level behavioral models required for this evaluation do not need to be cycle accurate as they are intended as a block-level description of the data flow and only need minimal control. They can be created at a very abstract level to allow early delivery of an evaluation model, as well as fast and efficient simulation of the VC and the proposed system chip.

These models generally will contain no information that can be used for synthesis, and could act as an efficient method for the marketing and/or evaluation of the VC. By providing a data sheet and simplified behavioral model, the VC user would be able to quickly gain technical knowledge with respect to the value of the VC in their specific design implementation.

The VC user can generate the system level test bench by starting with the highest level of abstraction and use this for the initial regression test used in the subsequent stages of the design process.

CONDITIONAL: The need for an architectural evaluation model is conditional on the functional complexity of the VC. It is seen as a fundamental differentiator between complex VC designs which the system architect chooses to observe and evaluate in the system environment prior to an implementation decision.

2.3. System Design

High-level behavior models are necessary to describe the functionality at the block level of a VC. The RTL source used for synthesis can also be used for chip verification. Higher level models should be considered for simulator efficiency during functional verification. These higher level models would be used to verify functionality of the VC in the chip or system context, verify the correctness of an RTL description in top-down design, and to provide models for software/hardware co-verification. Protected, or secure, models may be provided in an OMF compliant format.

2.3.1. Verification Test Bench

The presence of a set of functional and timing delay verification patterns is a requirement which serves several basic needs:

1. May serve as a basis for regression when Soft RTL VC is being modified by the system user.

2. Supports the use of verification testbench vectors and can be used for manufacturing test.

3. It provides a functional and timing delay verification check for a Hard VC which has been ported to another vendor.

The form of the verification test bench should be consistent with the Verilog or VHDL RTL model used. Stimulus/response along with timing and clocking controls is required.

2.3.2. Behavioral Model

The behavioral model is used to model the functionality of non-processor blocks of VCs. It is important for all classes of VCs — Soft, Firm, and Hard — and is used for verifying the functionality of the VC, for co-verification with other VCs, and for software-hardware co-verification. This would be a natural output of a top-down design of a VC. When VCs are developed using an alternate methodology, such as VCs designed directly at the RTL level, the behavioral model would need to be written and verified against the RTL description or the netlist description — if that is the only functional representation of the VC. A requirement of this model is that it can be simulated efficiently in the complex system-on-a-chip environment. A cycle accurate model is desirable. It need not model all internal functionality, only that portion which is visible at the boundary of the VC.

CONDITIONAL: The behavioral model is conditional for Firm VCs depending on whether an RTL model is available. If the VC provider supplies only a netlist then a behavioral model is required as a supplement for system level functional design verification.

2.3.3. Processor Models

Programmable devices such as microprocessors, DSPs, and microcontrollers have extra modeling requirements. Additional models are necessary because software is an important part of the verification of a chip containing a processor. Some of the support products for processor models include:

- Instruction Set Architecture (ISA) — a high level model which models the programmable behavior of the processor VC. The ISA emulates the behavior of the processor VC in response to any and all instruction set stimuli. It's written at a sufficiently high level to allow good efficiency in modeling the VC behavior in a system chip environment. As such, it does not need to model specific internal functionality except where it is visible to the user, such as user write-able registers. Two ISA models are recommended - one optimized for speed and targeting hardware/software cosimulation and one optimized for cycle accuracy and having the same interface as the RTL model. The datasheet should specify whether this model is cycle accurate (a cycle accurate model is preferred).

- Code Size and Performance Estimators — for the purposes of VC evaluation, simulation/emulation models and estimators are required for assessing performance criteria, such as RAM and ROM code-size for program and data, execution run-time (cycle-counting), and power.

- Software Development Tools — for the purposes of VC application development, extra software must be added to the VSI: compilers, assemblers, and loaders for application software, dis-assemblers, run-time analysis, and debug environments.

- Run-Time Libraries — part of the value of a processor may be software libraries that have been developed to run on it. Run-time libraries can either be in a source form or pre-compiled in a library that is usable by the software development tools. Run-time libraries may come from the processor VC provider or from another VC provider. Libraries which implement features of the user-visible instruction set architecture in software are particularly important as well as basic hooks for setting breakpoints and examining states for debugger support.

CONDITIONAL: The processor model is conditional for Firm VCs depending on whether an RTL model is available. If the VC provider supplies only a netlist, then a processor model is required as a supplement for system level functional design verification.

2.3.4. Bus Functional Model

The overall rationale for the use of a bus functional model (BFM) of a VC is to permit the user to verify that, as seen from its interface, the VC functions correctly with other VCs to which it is connected. Emphasis on behavior at this level has the following advantages:

- More lightweight, and thus, compile more quickly, occupy less memory, and run much faster than full implementation models. This permits a more thorough coverage of interface behavior to be verified in a reasonable time than would be possible with models that cover the full details of the implementation.

- Enable vendors to give their customers the means to perform thorough verification of the usage of their VCs, without the need to reveal internal implementation details. Most of the essential pins/ports will be present in the model, but some may still be missing — such as test pins. In addition, not all aspects of interface behavior will always be covered — for example, some interrupt modes may be missing in the case of processors. The trade-off is between speed and complexity of the model, and the most important aspects of the VCs likely use.

- Beyond the verification of correct function, BFMs usually have enough detail to permit the verification of interface timing, and will produce detailed error reports of misuse and violations.

- In some cases, a BFM may offer user-callable tasks or procedures that encapsulate and evoke complex bus protocol sequences, and serve as reference stimulus and test for external blocks under design by the user.

- BFM models may provide tasks or procedures that offer a programming interface to the attributes of a block that are not available in the hardware implementation. For example, procedures may be offered that permit the inspection of internal registers and status bits.

2.3.5. Bonded Out VC/Prototype

System chips with embedded software frequently require special simulation. Near the end of design process, to perform integration testing (in which hardware and software run together), a simulation system involving a hardware emulator is evoked. This is done for reasons of functional completeness and execution speed. In order to simulate with a hardware modeler or hardware emulation system, either synthesizeable RTL or a bonded-out VC is needed for all VCs on the chip. The unavailability of a prototype model as either RTL or a bonded-out VC for even one VC can complicate or prevent building a hardware prototype.

2.4. Logic Design

CONDITIONAL: As the definition of a Firm VC can range from RTL with topological planning to a technology specific placed netlist, the presence of synthesis and floorplanning/timing constraints are conditionally mandatory if synthesis from RTL is required. This conditional statement applies to all of section 2.4.

2.4.1. Synthesizable RTL Source

The RTL source file must be written in a standard RTL language and must conform to the synthesizable VHDL or Verilog subset. It should be written in a modular form that clearly identifies:

- ï memory functions to be instantiated through generators, or specified as pre-designed library elements.
- ï structured functions where generators or pre-existing blocks are to be used for performance, area, or power — such as data flow structures (note: structures which are instantiated require the availability of the required generators, libraries and appropriate supporting views).
- ï synthesizable logic.

The value of encryption for VC protection and cross contamination prevention for both the provider and the user respectively makes it a key element in the VC interface. An encrypted RTL file must provide for the modeling of any functions such as registers that need to be user visible for proper integration, verification, and debug of the VC within the system chip. If encryption is required, the OMF standard should be followed.

2.4.2. Synthesis Constraints

Any RTL intended for use in synthesis (Soft or Firm) is required to include a set of synthesis directives and constraints. The synthesis directives include information typically found in synthesis scripts, such as the order for processing the design hierarchy and wire load models. Synthesis constraints describe the performance and area objectives.

2.4.3. Floorplanning

Generically, there are two categories of floorplanning information required for the system chip designer.

Floorplanning Shell

High-level floorplanning is becoming an important step in system design flow. All VC types will need to have models that are usefully predictive at the system design level. At the present time, no model formats or tools are mature enough to be specified. However, the characterization of VCs is intended to establish a starting place for the refinement of system-level models. The floorplanning shells should have the following information:

Deliverable	Currently Used Formats	Candidate VSI Format	Soft	Firm	Hard	Comments
Floorplanning Shell						
Block abstract (size, pin location, and porosity)	ASCII, LEF	TBD		M	M	
Pin attribute (in, out, clock)	ASCII, LEF	TBD	M	M	M	
Timing information or data flow (timing arc)	TLF, Synopsys NLDM, ITL, MMF	TBD	M	M	M	
Power information (frequency)	ASCII	TBD	M	M	M	

Floorplanning Constraints

The presence of floorplanning information at the Soft VC level is typically expected to be region-based and correlated to the wire load constraint models. Relative placement of megacells — such as RAMs — is expected as part of the floorplan. If there is a pre-specified or preferred I/O configuration it should be described in the floorplan. For Firm VCs, the floorplanning is expected to be significantly more detailed, ranging down to a full placement for both internal and I/O cells. Floorplanning constraints have the following information:

Deliverable	Currently Used Formats	Candidate VSI Formats	Soft	Firm	Hard	Comments
Floorplanning Constraints						
Wiring constraints	SDF, PDEF	TBD	R	CM		
Placement constraints	PDEF	TBD	R	CM		
Block abstract (size, pin location & porosity)	LEF	TBD		CM		
Pin attribute (in, out, clock)	LEF	TBD	R	CM		
Timing information or data flow (timing arc)	TLF, Synopsys NLDM, ITL, MMF	TBD	R	CM		
Power information (frequency)	ASCII	TBD	R	CM		

2.4.4. Structural Netlist

In many cases, especially for Firm VCs, the deliverables include a structural netlist of blocks, generic gates, library-specific gates or transistors. The netlist may be used as an input to synthesis, to floorplanning or as a final technology-specific deliverable.

2.4.5. Basic Delay Model

The characterization of the embedded block in the time domain is an important element in delivering VCs which can be embedded. The typical methods for characterizing small SSI/MSI level standard cell functions have matured to provide a fairly accurate approximation of the actual cell behavior. This has typically been achieved through multiple SPICE simulations followed by curve fitting techniques. Unfortunately, this technique is limited by the practical bounds of SPICE simulation. For larger VCs, an alternative method is needed which combines static timing analysis with SPICE to provide a useful characterization. Furthermore, a method is needed to properly handle the I/O boundary conditions affecting the timing model and the true behavior over all possible clock domains.

The timing specification format remains relatively constant over the various types of VC. However, the accuracy of the characterization model improves as the design implementation becomes more technology specific. The format proposed consists of two fundamental representations, which are combined to provide a complete characterization. The first representation covers the VC independent of the peripheral interconnect, and the second (section 2.4.6) addresses the peripheral portion of the block recognizing the possibility that nets extending beyond the block boundaries must be modeled in the context of use at the system-level. A discussion of the utility of each model in light of the VC type is included.

The basic delay model includes :

- ï propagation delay model (delay between I/O ports expressed as a function of output load and input slew)
- ï slew model (output slew expressed as a function of output load and input slew)
- ï input and output port capacitance
- ï external setup and hold constraints (expressed at the input ports to the block)
- ï all values should be expressed for min, max, rise, and fall transitions

The recommended methodology for generating the model is through static timing analysis for all synchronous blocks. The model should include as a reference either:

- · the technology characterization parameters used in the model generation process (temperature, process, voltage); or,
- · the reference environment used in the VC layout.

The model format for the VC is executable at the next level of timing analysis and suitable for use by timing simulation or system-level static timing verification.

Soft VC

This model is not required because the accuracy without the presence of a cell library or interconnect structure is so poor it becomes useless. If the provider has verified his VC against a reference environment (Section 2.1.2), a model may be provided that is usable to the VC user. However, this provides limited information on how the block will behave in the customer application — primarily due to variances in layout and interconnect delays. We would recommend the creation of a generic library for a technology level (e.g. 0.5u, 3ML) against which vendors could characterize their cell libraries. The comparison estimates would be better, but this model would still suffer at this level due to a lack of predictable interconnect structure.

Firm VC

The interconnect delay approximations based on a topological layout description are considered sufficiently predictive for preliminary system-level timing analysis. Therefore, the base timing model is required for Firm VCs.

Hard VC

This model is required. A transistor-level static timing analysis with characterization for temperature, process, and voltage variations and a curve fit should be performed. Accuracy will be limited by the tools used and should be backed up with a SPICE analysis of the critical paths. The process used for characterization of the Hard VC should be listed as part of the VC documentation.

2.4.6. Peripheral Interconnect Model

The peripheral interconnect model represents the interface interconnect network shell around the block internal model. This provides a separation of the peripheral interconnect RCs from the block intrinsic delays expressed in section 2.4.5. By preserving the interconnect model for these nets from I/O ports to gates, delay calculation at the next level of hierarchy can be performed using the actual interconnect rather than an inaccurate approximation of loading and interconnect. This model is required for Hard VCs where the I/O interconnect is clearly understood (see Figure 2.4-a).

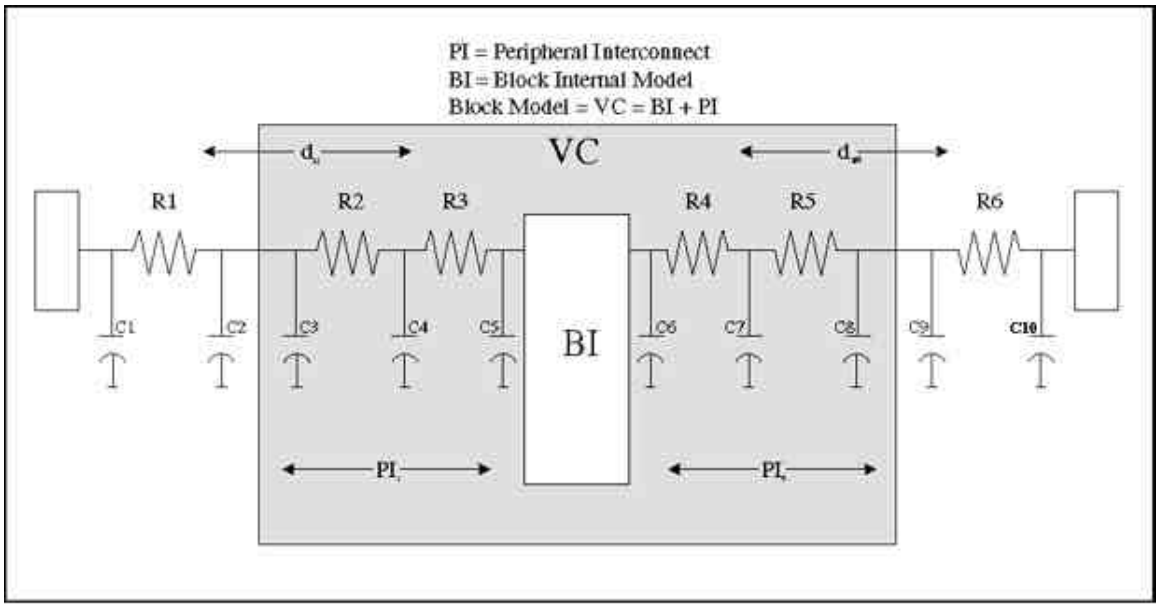


Figure 2.4-a Peripheral Interconnect Model

2.5. Test Requirements

Testing designs with embedded blocks is often challenging, as the I/O of the blocks may not be accessible from the chip I/O. The test methodology must be optimized based on test execution time, pattern size, performance degradation, and area overhead (including routing resources) due to test logic, high fault coverage, and fault isolation. Special attention should be given to fault isolation and test execution time. Multiple test methods (BIST, IDDQ, Stuck-at, Scan) may be required to adequately test VCs embedded within a system chip. The test architecture at VC level must support parallel testing of different VCs at chip-level. Adhering to test standards during the design of a VC will make the task of assembling a chip level test possible. Assuming they meet DFT constraints, Soft VCs can be tested by treating them just like any other system logic. In the case of Hard VCs, it is critical to have the associated test vectors, details on test methods, fault modeling, and fault isolation information. Thus, test requirements can be simplified into two categories:

- **Black-box —** these are VCs for which the provider must supply a test pattern set and does not allow modification of a VC. Typically, this category would apply to Hard and netlist-based Firm VC types.

- **Open-box —** these are VCs for which the user has access to the underlying logic. This allows the user to insert test infrastructure and create test vectors. Typically, Soft VCs are in the Open-box category for test. For Open-box VCs, it is required that the VC provider demonstrate design-for-test by supplying data that indicates test coverage and test vector counts for the VC using a reference technology library. This information is created by the VC provider taking the design through a paper implementation, inserting test, executing ATPG, and calculating fault coverage numbers.

2.5.1. Virtual Component Test Interface Architecture

The test architecture for block-based design can be broken into two areas of discussion: Socket interface and internal logic. The Socket interface describes the way the VC can be individually accessed from the chip IO for testing that VC, while the internal logic describes the test logic which is included within the VC or can be included within the VC by the user to facilitate test generation of vectors for the VC.

VC Test Socket Interface Types

The test architecture for block-based design requires that multiple types of sockets be provided at the chip-level to support the different VC requirements. Each VC must specify compatibility with one or more of these socket interfaces. Below is a brief description of each type, the detailed diagrams and descriptions are contained in the appendix.

1. Open-box: VC which can have test logic inserted by the VC user. Practically it is any VC which is Soft or possibly Firm that conforms to some DFT rules.

The following are all Black-box VC:

2. Internal boundary scan-based: VC which has boundary scan with its test controller and access port included as part of the VC. The following are all External socket interfaces, and as such are added by the VC user:

3. External Boundary Scan-based: The user adds a boundary scan around the VC which provides access to all non-global IO.

4. External Full Mux Interface: The user adds muxes to all non-global VC IO which directly connect these IO to chip pins in test mode.

5. External partial Mux/scan Interface: The user adds some combination of either boundary scan or muxes to all non-global VC IO. In this case the Muxed IO usually have some timing critical testing requirements.

VC Internal Test Logic Options

The VC may be designed with or for various types of internal test logic contained within the VC. This logic may already be included within the VC if it is Firm or Hard (Black-box) or the VC may be designed to meet certain DFT rules which would support the addition of the appropriate test logic by the user (Open-box). There are three types of internal test logic referred to in this section:

a) Scan: Full or Partial Scan may be included within the VC or the VC may be designed to meet the DFT rules for inclusion of scan by the user.

b) BIST: Built-In Self Test structures may be included within the VC to aid in testing the VC. This is specially true for large memories.

c) None: No special test logic for test modes may be built into the VC. Vectors may still be provided and applied by way of the chosen Socket interface method.

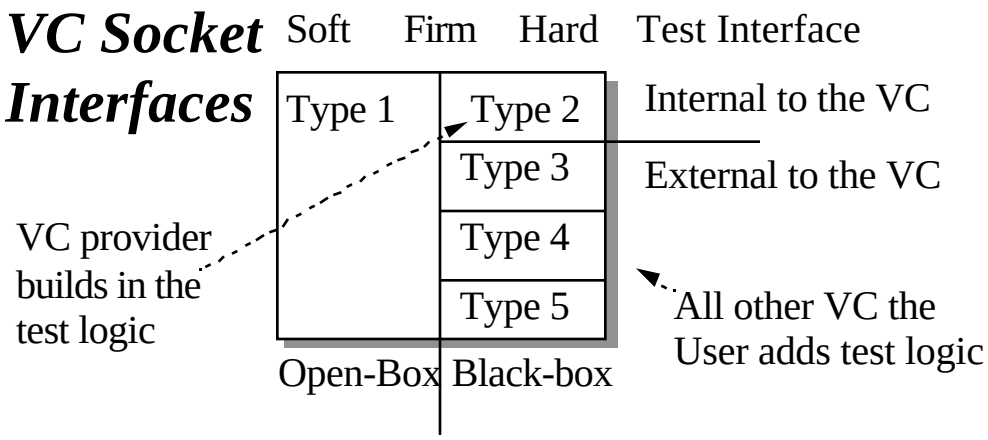


Figure 2.5.1-a VC Test Categories

2.5.2. Test Methods

The successful testing of a system chip, which incorporates multiple VCs, requires that the system test designer and the VC designer provide complementary information regarding test. The system test designer is responsible for the infrastructure to interconnect the VC to the sockets and to satisfy the test requirements of the VCs which are used.

The VC provider needs to describe the socket requirements for such things as test access interface, test vectors, and test methods which are to be applied. This section references the types of tests, identifies a proposed standard for each, and maps the VC into the test architecture interfaces described in figures 2.5.1. The VC provider must document the test methods available in the VC and the appropriate information enumerated below for each test method.

- Global Signals: the VC provider needs to identify all global test signals, clocks, and test ports along with a detailed description, rules to follow, and other requirements during normal operation and test mode. If the core has a bus interface, it must also have a global bus disable signal. That signal can be specifically designated pin/port, or it can be a 1149.1 defined ‘EXTEST’ instruction op code.
- TAP Controller: If there is a TAP controller in a VC, it must be designed as per the IEEE 1149.1 standard, and the associated BSDL file must be provided. If the VC provides a test vector set based on using the 1149.1 test data registers, the VC provider must provide user instruction(s) that facilitates testing of the core through TAP, or through the ‘INTEST’ instruction.
- VC provider is required to design the VC so that it can be put into quiescent state for IDDQ testing.
- The socket type required by the VC.

Open-box Test

The open-box test includes Soft VCs and certain classes of Firm VCs.

CONDITIONAL: This information is mandatory for Firm VCs which do not include test logic.

Open-box Test Descriptions

The VC provider is responsible for providing the following information: recommended test methods to test the VC along with test coverage numbers, as well as the tools and libraries used in obtaining the stated coverage. There are two cases which apply to the open-box test:

1. In most situations, the VC provider relies on the system designer to insert the test logic and create the test patterns. In this case, the VC provider meets the test requirements by performing a “paper” test generation exercise, in which the test logic is inserted and ATPG is run against a reference environment.

2. In some cases the VC provider may choose to build a BIST feature into the VC. In this case, the coverage number provided would be based on a fault grade of the BIST vectors. The rules and requirements specified for BIST apply.

Black-box Test

The black-box test applies when the VC provider provides test vectors but does not supply a detailed description of the VC.

Black-box Test Description

The black-box VC provider must specify the test method used, the type of test structures, the test coverage, etc. The different methods and requirements are outlined below.

CONDITIONAL: This information is mandatory for Firm VCs that include test logic.

Deliverable	Format (current)	Format future)	Soft	Firm	Hard	Comments
Open-box Test Description	Document	Document	M	CM	-	If the VC faults fall into open box category
Black-box Test Description	Document	Document	-	CM	M	

Fault-based Test Patterns

The VC provider must provide applicable test patterns (such as functional, toggle, stuck-at, shorts, open, delay test, etc.), and ensure that the vectors conform to the following guidelines:

1. Pattern set must have a stated fault coverage from an industry available tool, along with detailed information on fault

modeling and DFT methodology used in obtaining the stated fault coverage.

2. Test execution time is important, and the VC provider must make every attempt to provide a compact vector set that provides maximum fault coverage and minimizes test execution time.

3. The patterns must be in a usable form and a detailed explanation of how to apply them at chip-level (including any chip-level requirements) must be provided.

CONDITIONAL: This information is mandatory for Firm VCs that include test logic.

Stuck-at Patterns

CONDITIONAL: The VC provider must provide stuck-at fault patterns if the stuck-at faults are not covered by other test types , such as BIST. The test pattern set should have high stuck-at fault coverage.

Scan Chain Confidence Patterns

Scan chain confidence patterns will ensure coverage of faults within the maintenance logic. They also provide users a simple way to verify their Virtual Socket is interacting properly with the VC, and provide some measure of isolation and failure debug.

CONDITIONAL: If the VC provider supplies a pattern set that relies on a scan infrastructure, then a set of scan chain confidence patterns must also be provided.

Deliverable	Format (current)	Format (future)	Soft	Firm	Hard	Comments
Fault-based Test Patterns	WGL, VCD	TBD	-	CM	M	If the VC faults fall into black box category
Stuck at Patterns	WGL, VCD	TBD	-	CM	CM	If not covered by BIST
Scan chain confidence patterns	WGL, VCD	TBD	-	CM	CM	If scan chain is used

IDDQ Test Patterns

IDDQ test patterns, when provided, must conform to the following requirements:

1. VC providers who claim IDDQ capability must supply sufficient information to the VC user to support system chip IDDQ test requirements — such as a set of port/state pairs and sequencing directives that place the block into an IDDQ compatible mode.

2. IDDQ patterns must conform to the format of the other test vectors (WGL or VCD).

CONDITIONAL: This may be mandatory based on silicon manufacturing requirements.

BIST

BIST is the preferred method for meeting the requirements for RAM and general control logic structures. Embedded processor diagnostic testing via a test bus would be included in this category. VC providers who have BIST logic within their design must provide:

1. Stuck-at fault coverage numbers for logic BIST (LBIST). For RAM and register array structures, a text description of faults covered is sufficient.
2. The set of port/state pairs that set the VC into its BIST test mode.
3. The pattern sequence that initializes the BIST sequence.
4. How many cycles/clocks to burst.
5. Expected output signature.

It is expected that the BIST infrastructure will be designed into the VC.

CONDITIONAL: Mandatory for large memory structures.

Deliverable	Format (current)	Format (future)	Soft	Firm	Hard	Comments
IDDQ Test Patterns	ASCII and WGL, or VCD	TBD	-	CM	CM	Per manufacturer's requirement
BIST	WGL, VCD	TBD	CM	CM	CM	Large memory arrays

Delay Tests

Delay tests are optional and are aimed at detecting manufacturing defects that result in the circuit failing to operate at the required operating speed. Since there are many variations of delay tests with different names, VC providers who do supply delay test pattern sets must provide:

1. A detailed description of the test and reasons for performing the test — including fault model information, if applicable.
2. The name and active state of any test mode signal that enables the test logic necessary to perform delay testing.
3. A list of timing-critical ports on the VC that need access from the chip I/O. Additionally, providers of Hard and Firm VCs must supply the timing relationship between the critical ports.

Reliability (Burn-in)

It is recommended to providers of Hard and Firm VC to supply a pattern set for burn-in (for reliability consideration). This pattern set could be an identified subset of the complete pattern set that has been provided. If scan is present, scan chain shift patterns should also be provided.

Failure Analysis and Fault Isolation

This requires that providers of black-box VCs supply specific diagnostic vectors or failure analysis information, along with their test patterns so as to isolate the defect to the lowest block-level in the block-level diagram provided by the VC provider. Analysis of failing parts may become very critical for identifying the problem source and improving the yield.

For scan-based VCs, it is recommended that scan chain confidence patterns and the scan based patterns be supplied. For BIST and embedded block test sets it is recommended that intermediate fail signatures be supplied to assist in fault isolation. The format for the fault isolation information will be defined in the future.

Deliverable	Format (current)	Format (future)	Soft	Firm	Hard	Comments
Delay Tests	WGL, VCD	TBD	-	R	R	
Reliability (Burn-in)	WGL, VCD	TBD	-	R	R	
Failure analysis & Fault isolation	TBD	TBD	-	R	R	

Below is a Flow chart which simply diagrams out the requirements for test. It also notes what are the applicable Socket interface types and internal test logic options associated with each requirement.

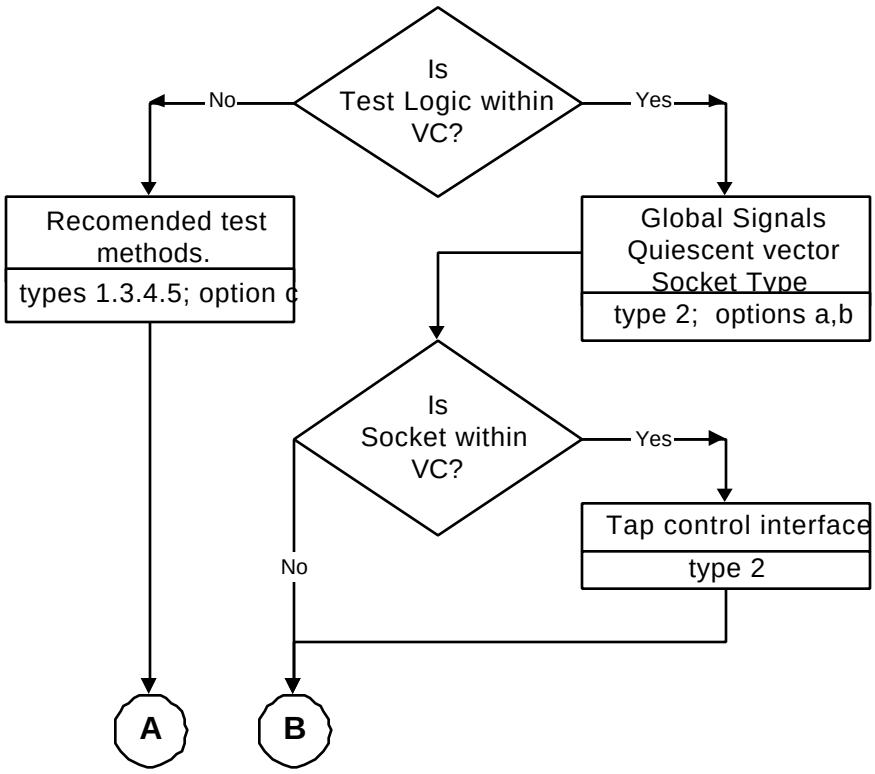


Figure 2.5-a Test Requirements Decision Tree (Part 1)

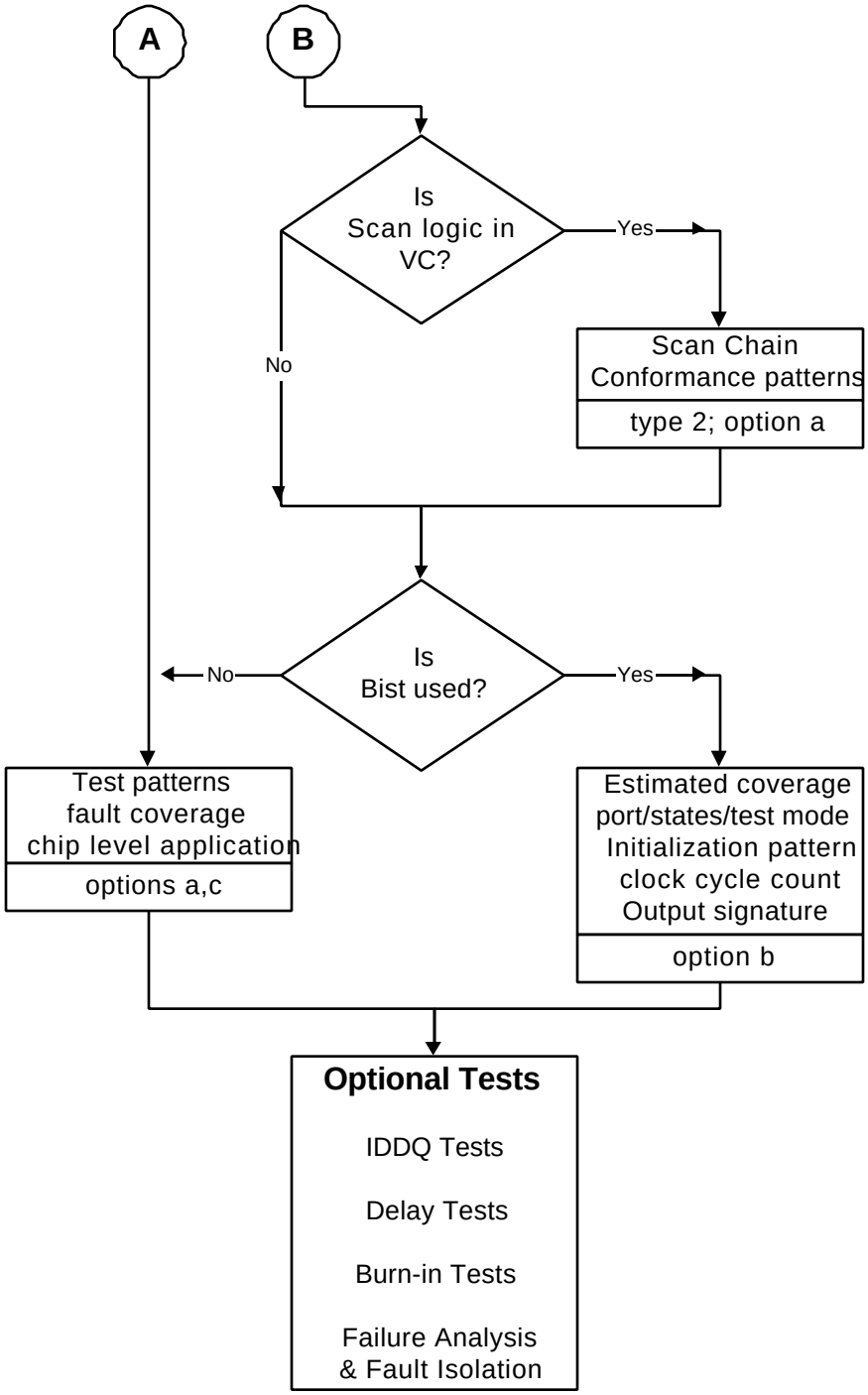


Figure 2.5-b Test Requirements Decision Tree (Part 2)

Each box in the above decision tree contains a section which lists the applicable Socket interface types and internal test logic options. Any one of the listed types or options constitutes an

applicable structure to have the requirements within the box apply to the VC.

2.6. Physical Block Implementation

For Hard VCs, the physical block implementation — sometimes called the physical abstract or footprint — is a set of files which describe the physical attributes of the block necessary for the integration of that VC within the higher level chip physical architecture and along with other VCs. This set of models include the following:

CONDITIONAL: The presence of detailed physical information for Firm VCs is conditional if a placed netlist is included. If the Firm VC includes a placed netlist with fixed I/O, then a footprint, pin placement, blockage, and power grid is mandatory. This conditional statement applies to all of section 2.6.

2.6.1. Block Description

The block description provides the most detailed physical description of the VC in GDSII format. Primarily, this image is used at the system-level following place-and-route for final chip integration and design rule verification.

CONDITIONAL: GDSII must be available either to the VC integrator or to the manufacturer.

2.6.2. Pin List/Pin Placement

This contains the VC information on size, dimensions, ports, and all other information required to allow the placement of the block, within a chip, and on the appropriate grid structure for place, route, and delay parasitic extraction. This information is generally referred to as the physical abstract.

2.6.3. Porosity/Blockage File

A blockage file needs to be available, such that available channels and over-the-block routing can be utilized at the chip integration level. This file must also identify (block) area/channels that are physically open, and where routing should be denied due to crosstalk or other noise related issues.

2.6.4. Footprint

The VC dimensions are included in the format file, and define the block boundary in standard format that would allow the placement of the block within a chip on the appropriate grid structure for place, route, and delay parasitic extraction .

2.6.5. Power and Ground

Power and ground connections must be defined in the file. This file should describe the top level power grid, the power and ground ring description (if used) and the allowable connect matrix for both power and ground. A separate current density file, designating current requirements as a function of contact, should also be included for

reliability and chip-level voltage drop budgeting. This file needs to also include the number of power and ground drops (pins) required.

Sufficient power and ground connections must be provided to meet voltage drop, noise, and simultaneous switching requirements under worst case conditions.

2.6.6. Power Model

The power model characterizes the power usage of a VC as a function technology of implementation, clock speed, and typical or worse case data model. Power should be characterized for typical and worst case expectations.

The purpose of this model is for reliability, design of a chip, power budgeting, and power management of the chip design.

2.6.7. Physical Netlist

A device level netlist is recommended which is sufficient to define the electrical characteristics of the VC interface.

CONDITIONAL: Dependent upon the availability of sufficiently accurate I/O models.

3. Virtual Component Guidelines

This section is a preliminary document. Section 3 is intended as a seed for design guidelines to be reviewed by working group(s). Sections 1 & 2 are intended for review - Section 3 is only intended for refinement.

Guidelines fall into three major categories:

Interoperability	In order to ensure the ability to use VCs from different providers, which may have been developed in different environments, several issues need to be addressed to guarantee this "interoperability". For example, the matching of assumptions with regard to the reference environment and the matching of naming conventions will be necessary.
Hierarchical Integration	As the VC and the system chip designs represent two different levels in the overall design hierarchy, rules are required to bridge the hierarchical boundaries between the VCs and the system chip. This is important when VCs, which were designed separately, are used in parallel with other VCs on the system chip.
Portability	Future versions of this document will address VC portability with respect to issues like general mask pattern design guide lines (e.g., in the case of those Hard VCs to be transferred between technologies).

These guidelines are neither an attempt to define design methods for VCs nor to define design methods for system design. Rather they facilitate the integration of VCs into a system chip.

Both rules and recommendations are included as guidelines to the VC provider to facilitate the integration of the VC into the larger IC design. Guidelines that must be followed by VC authors is denoted by either the words "must" or "rule." Guidelines which encourage VC authors are denoted by the words "recommendation" or "should."

Some specific areas for design guidelines have been identified:

Naming Conventions	Guidelines to prevent name collisions among VCs on a chip and to prevent common translation problems between formats and tools.
Test	Guidelines to promote a common approach to chip-level testing in the presence of multiple VCs.
Logic Design	Guidelines to promote electrical and functional compatibility among VCs on a chip, and to avoid common tool limitations. This includes clocking, and restrictions on the use of bidirectional ports.
On-chip Busses and Protocols	Definition of specific methods for passing information between VCs, analogous to the domain-specific busses and protocols in the board world.
Physical Design	Guidelines to promote layout compatibility among VCs on a chip and to avoid common tool limitations.

3.1. Naming Guidelines

This shall describe the conventions to be followed for VC component naming, pin naming, and modeling language inter-operability.

There are appendices attached to this document in support of naming definitions: Appendix A identifies keywords that should be avoided in naming by VC providers; and, Appendix B proposes a set of conventions that EDA vendors should follow to facilitate inter-operability. Names and naming consistencies/ transformations between Verilog, VHDL, C, C++, and the file system are defined in detail.

3.1.1. VC Naming

1. The top-level VC name must be meaningful and reflect the function of the VC. (e.g., MPEG2 core, vc_mpeg2).
2. Any known standard, implied or otherwise, in the component name must be documented in the claims and verification section of the VSI document.
3. Different cases should not be mixed (i.e., all lower case or all upper case, no mixed case naming).
4. Lower case is recommended.
5. The previous conventions should also be applied hierarchically to all lower level module names that are seen by the VC user.

3.1.2. VC Pin Naming

1. Pin names need to comply with any existing naming conventions for formats being used (e.g., VHDL, Verilog naming requirements, reserved key words).
2. Different cases should not be mixed (i.e., all lower case or all upper case, no mix case naming).
3. Lower case is recommended.
4. A consistent and coherent naming convention must be used for all common or global signals — such as clock, reset, etc.
5. Active high and active low signals should be labeled to differentiate between the two types (e.g., sig1x, sig2x, where x denotes active low).
6. All pin names must be documented in the User Guide for the VC. This should detail the various signal attributes which may be required by the VC user (e.g., clock, reset, in/out/bi-directional).
7. All busses must start at zero.
8. It is recommended that all busses be designated with the most significant bit on the left (i.e., D[7:0], so that D[7] refers to 2^7).

3.1.3. VC File Naming

1. Known keywords should be avoided — see Appendix 1

2. A consistent naming convention should be used throughout the file naming used for the VC. Recommended example naming formats are:

   - ï C files vcname.c
   - ï C++ files vcname.cpp
   - ï Verilog vcname.v
   - ï VHDL vcname.vhd
   - ï SDF data vcname.sdf
   - ï test, etc.

3. Directories should be used to allow the separation of different data types, such as gate-level and RTL VHDL:

   - ï rtl/vcname.vhd
   - ï rtl/vcsubmodule.vhd
   - ï gate/vcname.vhd
   - ï gate/vcsubmodule.vhd

4. Whichever naming convention is used must be reflected in the User Guide. It should be included in Section 2.1.4 Version History. This will enable the revision control to be closely linked in with the filenames.

5. Where multi-platform support is required, the VC provider is responsible for ensuring a consistent naming format — not the VC user. The VC provider should document which platforms are supported for a particular VC.

6. Filenames should not differ by case alone. This will reduce any problems with multi-platform support.

3.2. Test Guidelines

3.2.1. Functional Test

There are two types of functional tests that are applicable: compliance tests and functional verification tests. Compliance tests are a set of tests that verifies the interface function meets (complies) with some known industry standard. Functional verification tests are a set of tests to check out the system functionality of the VC. In both cases, the developer should verify that these functional tests at least meet some control state completeness and provide the associated documentation. For example, one way to do this is to make sure the tests traverse all the arcs of the VCs control logic state diagram. Some tools may be available in the near future to help evaluate these tests. Coverage could be defined as the percentage of arcs traversed versus the total arcs in the control state diagram. Further work should be done in this area.

3.2.2. Virtual Socket Test Interfaces

The test architecture for block-based design requires that multiple types of sockets be provided at the chip-level to support the different VC requirements. Each VC must specify compatibility with one or more of these socket interfaces. This section defines the types of socket options available in the test architecture. Figures 3.2.a - 3.2.d provide examples to support the definitions below.

1. Classic Open-box VC: in this scenario, the provider has supplied a VC along with reference data which demonstrates that the VC conforms to DFT rules and what the user can assume about coverage, vector counts, and area. The user of the VC then inserts the scan infrastructure and generates tests. In this scenario the VC is integrated into the system like the user designed logic.

2. Internal boundary scan-based Black-box VC: the VC provider supplies a set of test vectors that are accessible through formally published standards, such as 1149.1 interface. This mechanism can support scan-, functional-, or BIST-based test generation techniques at the providers discretion (Figure 3.2-a)

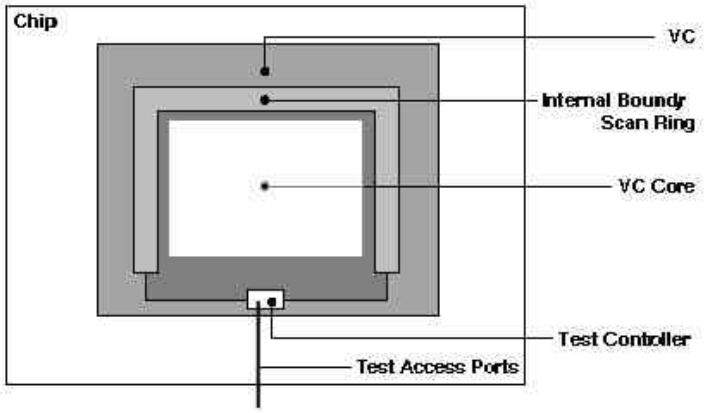


Figure 3.2-a Internal Scan Black Box VC

3. External Boundary Scan-based Black-box VC: the VC provider supplies a set of test vectors that are applied through an external boundary scan ring created by the system designer (Figure 3.2-b)

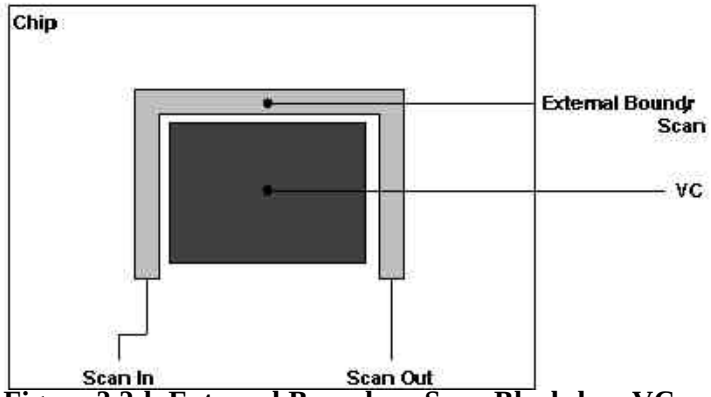


Figure 3.2-b External Boundary Scan Black-box VC

4. External Full MUX Interface Requirement: In this scenario, the provider has supplied a pattern set that requires direct chip port access for all ports on the VC. At-speed tests for RAMs and other types of structures are prime examples where the full MUX interface can be used. The provider is cautioned that this type of testing can rapidly consume a significant number of pins and routing resources at the chip level (Figure 3.2-c)

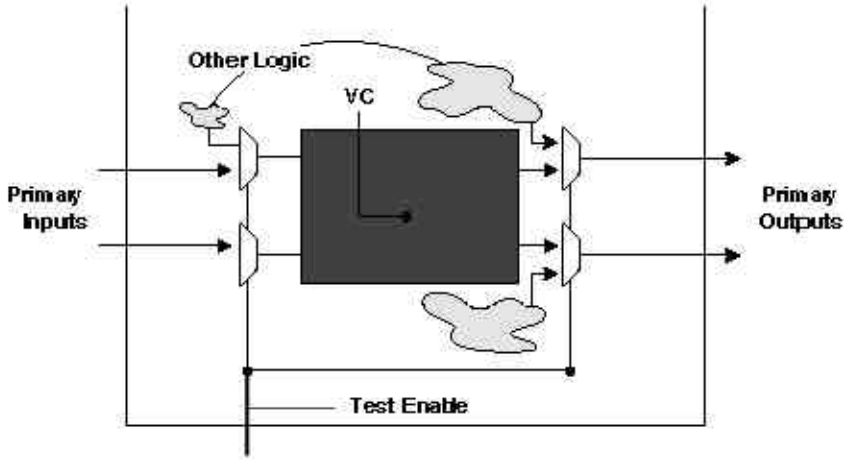


Figure 3.2-c External Full MUX Interface

5. External partial MUX/scan interface: in this scenario, the VC provider has supplied a pattern set that requires direct chip port access for some of the ports on the VC. The user is require to provide a combination of MUX access to critical ports and scan access to the remaining ports. This can be thought of as a combination of the two previously described techniques.

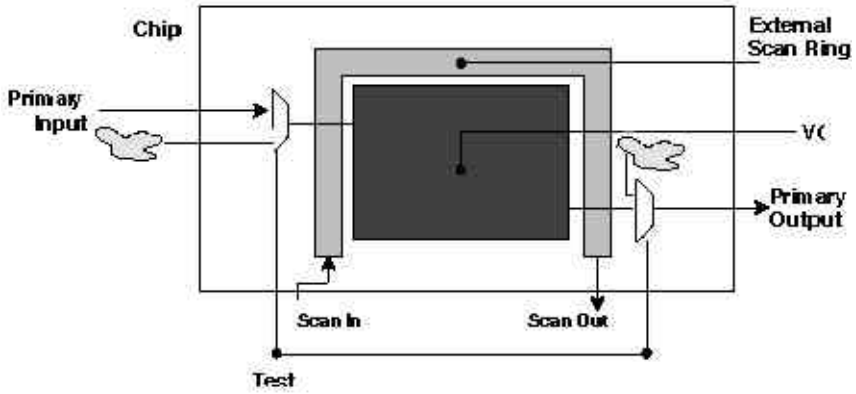


Figure 3.2-d External Partial MUX/Scan Interface

3.2.3. Structural Test Methods and Guidelines

Much of the testability discussion references fault models. For a detailed explanation refer to your favorite book on test engineering.

General Testability Improvements

Generally, test patterns can be more easily generated when there is a high degree of observability and control. Design-for-testability guidelines are listed below.

1. A circuit should not contain asynchronous feedback loops. A circuit with asynchronous loops is much harder to test than a comparable synchronous circuit due to time dependency.

2. Make the VC easy to initialize — a VC needs to be in a stable initial state to help generate test patterns.

3. Avoid gated clock — clocking in is essential to be able to control and observe during test. If a gated clock is necessary, design the circuit such that:

   - ï there are states or control signals that allow the clock to pass through the gating

   - ï minimize clock skew between gated and non-gated registers/flip-flops

   - ï it can handle the short-path problem for scan paths, given some clock delay due to gating

   - ï it ensures the gated clock circuit can be initialized.

4. Eliminate race conditions between data and clock of sequential elements

5. Eliminate potential tri-state contention. Design the circuit in such a way that internal bus contentions cannot occur, especially when the circuit is not in its normal mode.

6. Decoded multiplexer — similar to a tri-state contention problem in a tri-state bus, decoded multiplexers need to have only one input selected during test.

7. Circuits which require large sequential patterns should have test logic inserted to partition the design for increased parallel testing (example: 32-bit counter should be partitioned into four 8-bit parallel groups).

8. Circuits with separate power supplies should be isolated from each other (analog-analog, analog-digital, digital-digital).

9. The VC provider should supply a test method for controlling and observing all input and output signals to the VC.

Scan Design

Definition

A scan latch/flip-flop is a latch/flip-flop that has a set of connections from one latch/flip-flop directly to another in test mode. These latches/flip-flops are serially connected together via these test connections to form a scan chain. The scan chain's contents can be shifted in and out of the serial chain in test mode. In this way, the scan registers and control states can be defined by shifting in values, and the results from clocking these storage elements under normal operational mode can be shifted back out.

There are two prevalent types of scan implementations: full scan and partial scan. In full scans all internal registers and control states are scan latches/flip-flops, while in partial scans only some of the registers and control states are defined as scan latches/flip-flops.

There are different rules for different categories of VCs. In the case of Open-box VCs, the scan logic may be added to the design by the user, while in the case of Black-box VCs the scan logic must have been added by the VC provider. In the rule section below, the rules are annotated with either Open-box or Black-box, depending upon which is applicable.

Rules for Scan-Based Design

1. The design should be synchronous (Open-box and Black-box).
2. The design must meet standard scan design rules or some variant of them (Open-box and Black-box).
3. Provide scan control, clock, and data as external pins (Black-box VC).
4. Provide a separate file containing the scan chain order (Black-box VC).

Methodology Guidelines

In general, the logic should be organized to minimize the number of patterns that either were generated to get at least 99.9% stuck fault coverage or would be generated. To minimize the required patterns necessary to test the VC in the partial scan case, additional test only registers should be added to those areas with low ability to observe and control. Other guidelines include:

1. Full-scan design is highly recommended when possible.

2. Avoid destructive flip-flops in the design as much as possible. Destructive flip-flops are defined as flip-flops not in the scan chain that can change state during scan mode.

3. Free running oscillators are difficult to synchronize to the tester. It is imperative to bypass the internally generated clock signal with an external test clock during test.

Multiplexer Technique

Definition

In this technique, multiplexers are added around functional blocks within the VC, such that these blocks can be independently tested from the VCs boundary without having to test the whole VC.

Rules

1. Each block of logic must be completely and independently testable.

2. The test logic must be constructed such that it is completely tested in the process of testing all the multiplexed functions.

3. All functional block outputs within a VC must either pass through another functional block that has a bypass mode, or the pins must be multiplexed directly to some of the VCs outputs.

Methodology Guidelines

It is important to connect to the VC's multiplexed output pins the output of multiplexers that are used as inputs to adjacent functional blocks rather than the output of the original functional block. In this way, the data side of the input test multiplexers for any functional block will be tested by the adjacent functional blocks tests. Care should be given to make the structure as simple as possible.

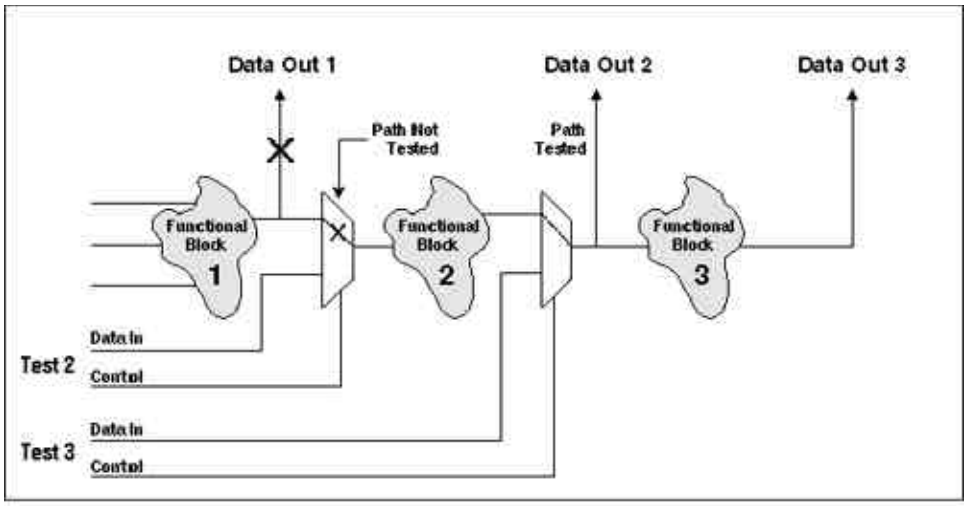


Figure 3.2-e Multiplexer Test Access Points

BIST

Definition

Built-In-Self-Test (BIST) brings the test application into the chip. A BIST structure typically contains a test pattern generator and an optional a result compressor. The test pattern generator can either be a ROM containing all desired input vectors or a counter generating a sequence of vectors. The results of the self-test need to be predictable to differentiate a pass from fail, and therefore, the test pattern cannot be truly random. Resulting compression can be done with parity checker, counter, or signature analyzer.

Rules

1. The logic being tested must have determinate output values over the set of possible inputs (no floating or undefined states).
2. The BIST must be able to single step for diagnostic isolation of failures.
3. The algorithm used in generation and checksum must be guaranteed to not repeat over the number of tests applied.
4. To effectively run a BIST test in parallel with other BIST tests, the infrastructure must automatically shutoff after the specified number of burst cycles/clocks has been issued.
5. To facilitate manufacturing debug, the BIST structure should supply the ability to read out the intermediate signatures and status of registers and storage elements.
6. The VC provider must provide a means of testing the self-test logic.

Methodology Guidelines

Essentially, all of the above rules require that the logic and the test sequence are robust enough to sufficiently test the logic in a deterministic manner. The valid and known states requirement is there because the design may be tested in ways that it was not intended to be used when the BIST tests are applied. If this happens, erroneous timing problems or illegal undefined states could cause a failure in BIST when the design is actually good. Conversely, rule 3 is required to insure that the test sequence and resulting checksum is good enough to actually detect almost all the possible real failures in the design. Some fault grading or statistical fault grading should be done on the resulting BIST test sequence to verify the coverage.

Boundary Scan Design

Definition

Boundary scan-based VCs place registers at all terminals of all VCs, and allow the choice and control of the VC through a test protocol, such as defined in IEEE 1149.1. The advantage of boundary scan is that it provides access for running functional tests on the VC. It is also possible to check the inter-net connections between VCs. While requiring a small number of chip-level pins, this technique does increase test cycle time. The testing speed of the chip becomes slower due to boundary scan shifting. The area taken by the boundary scan register is also large. There are two basic approaches to boundary scan as described in Section 2 of this document. Type 2 is internal to the VC and type 3 is external to the VC. The rules below note, in parenthesis, to which approach they apply.

Rules for Boundary Scan Design

1. All I/Os should be included within the boundary scan, except clocks, asynchronous reset, power, and ground. (internal)

2. When connecting adjacent VCs together, it is acceptable to only include the outputs of the source VC or the inputs of the target VC, but not both. (external)

3. Separate latches should be used for input, output, and enable lines going to a bi-directional pin. (internal and external)

4. The logic must be constructed so it is able to test the connections between VCs, the logic within each VC, and the test logic itself. (internal and external)

5. A technique for bypassing each VC should be provided. (internal and external)

Methodology Guidelines

When including the boundary scan within the VC, it is necessary to include either an instruction similar to the INTEST or bring internal scan chains out of the VC to use scan test in the VC. When including the boundary scan internally to the VC, all I/Os must be connected to the boundary scan since the VC provider does not know which test technique will be used external to the VC. In the

case where the boundary scan may be added externally to the VC, the user may choose to merge boundary scan chains together to minimize the test logic. In general, the logic implementation I/O guidelines should be followed, but if bi-directional or tri-state I/Os are used, the boundary scan should include separate control lines for the enables and data-in in a fashion similar to IEEE 1149.1. Clocks may be gated, but the gating should be done within the VC. This will allow the boundary scan logic to control the gating of the clock directly within that VC's specific tests, otherwise the pattern may become complicated since one VC must be set in the proper state to test another VC. It may not be possible in this case to setup the required initial conditions to properly test the VCs in a system. Lastly, a bypass option will allow for smaller tests sets since not all data needs to be scanned in on each pattern.

Bus Technique

Definition

There are two versions of bus oriented testing: parallel scan, and system bus test access.

In the first case, the bus merely does a parallel load of test data in the same fashion that scan chains are loaded serially. Test bus is a quick and dirty way of accessing primary ports of each core; one or more buses are connected to all primary ports of all VCs. A decoder is used to select which core or cores receive vectors from primary inputs of the chip, and a multiplexer is used to select outputs of which core(s) is to be observed at primary outputs of the chip. The exact number of pin depends on the desired number of accessible primary ports of each core. The major drawback of this method is the requirement of global bus routing. The lack of automation and tools limits the wide implementation of this test structure.

The second case uses a special test bus, or the system bus, for testing the chip. The chip must include a controller or processor that will be the test engine. In this case, there is a special test mode that allows the test logic to load instructions and data into the controller for executing diagnostics, and to access the peripheral VCs outputs through a special bus access of "boundary scan-like" test ports. This section will only deal with the second type of test structure.

Rules for Test Bus

1. The test port must have both master and slave capabilities on the system bus in test mode.
2. The test port must be able to load cache and any internal memories independent from the processor.
3. The test port must be able to access all internal busses directly or indirectly.
4. The test port must be able to test itself.
5. The test port must be able to access other special test registers that are added for testability.

Methodology Guidelines

Initially, the test port must act like a master to load the memories in a state necessary to begin the diagnostic tests. In many cases, the test port will be built into a bus bridge so as to be able to directly control both busses on either side of the bridge. In some cases, where there are multiple processors on the chip, each with separate system busses, it will be necessary to put the test logic into some common port, such as the memory controller. Special test registers should be inserted on lines between VCs, and between VCs and outside ports. This allows the test port to isolate the individual VCs, set up the external conditions, and feed back the resulting output from the peripheral VCs onto the system bus for the processor to check. In some cases it may be sufficient to connect multiple copies of a peripheral VC together with a multiplexer controlled by a single bit test register. Data could then be written to and read back into the processor to test the peripherals. This test technique is a form of system-level BIST, in that the system is set up and then a large number of clocks are issued.

IDDQ

Definition

CMOS is inherently low power. As such, any high current condition normally would be a sign of a fault in the design. IDDQ is a set of vectors that are designed to test the independent gates by checking the standby current after each vector. These can be used in conjunction with other functional test techniques.

IDDQ Rules

To enable IDDQ measurements, the following rules should be observed.

1. No floating nets.
2. No dynamic latches.
3. No totem-pole or high current conditions.

Methodology Guidelines

As is the case with partial scan, if a gate cannot be adequately exercised the potential faults on that gate cannot be tested. Unlike scan-based testing, the effective output is the power supply, which means there are no sensitized paths to create. Still coverage for IDDQ can be low if there are not sufficient test points for controllability of the gates. When IDDQ is the only test technique used, care must be taken to add test points for adequate controllability. Lastly, any normal high current conditions or floating nets that cause unintended high current conditions will mask out any failures on single gates which result in high current. As designs become very large it is possible that the normal leakage current of the overall chip might exceed any single IDDQ failure. In this case, the power supply may have to be designed to be shut off such that each section of the chip can be IDDQ tested individually and still detect single transistor faults.

Delay Testing

To be defined.

3.2.4. Example of VLSI Tester Capabilities

It is important for both VC provider and the VC user to know actual specifications of a VLSI test instrument. These specifications are shown as a reference, although these specifications will help the VC provider consider the test pattern limitations. As an example, the following table shows two sets of reference: (a) a typical tester, and (b) an advanced tester.

No.	Items	Typical Tester	Advanced Tester
1-1	Maximum Operating Frequency	40 MHz	150 MHz
1-2	Resolution of Test Rate	150 pS	15 pS
1-3	Resolution of Timing Edge	60 pS	50 pS
1-4	Total Timing accuracy	± 1.5 nS	± 500 pS
1-5	Maximum Pattern Length	128 Mbyte total	4 Mbit per channel
1-6	DC Level Accuracy	Drv = ± 50 mV	Drv = ± 25 mV
		Cmp = ± 50 mV	Cmp = ± 25 mV
		(@ 3V)	(@3 V)
1-7	SCAN Pattern Size	256 Mbit	512 Mbit
	SCAN Operating Frequency	40 MHz	40 MHz
	SCAN Chain	1/2/4/8	1/2/4/8

Table 3.2-a VLSI Tester Examples

WGL Description Guide

WGL can handle various test pattern waveforms and generic data structure, although a typical VLSI tester cannot accept all the specifications of WGL. These limitations will be shown as a reference in the near future. Several fundamental restrictions are:

1. Waveform format: WGL accepts triple-clock format but a typical tester does not recognize the format.
2. Test pattern length: WGL supports test pattern using REPEAT or subroutine syntax to reduce redundancy in test program descriptions. However, some testers do not support the functions.

3.3. Chip-to-VC Hierarchical Integration

Designs at the VC-level and chip-level often use similar methods but at different levels of design hierarchy. However, where differences in methodology occur, it becomes necessary to provide an approach to “bridge” the two levels of hierarchy together. This can be done through providing guidelines at both the logical and physical integration levels.

3.3.1. Logic Design Integration Guidelines

Logic design integration guidelines are those guidelines affecting the logic at the boundary of the VC interfacing to the system chip.

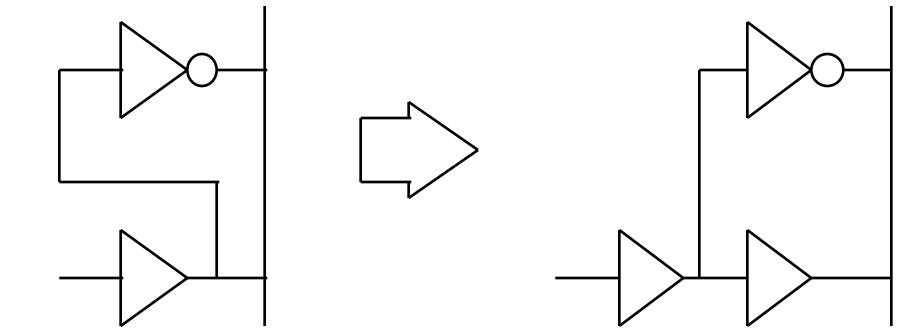
Logical I/O Ports

The I/O ports provide the chip-level interface to the VC. These ports must provide the interface between two distinct levels of hierarchy: the VC and the system chip. As such, these ports must be carefully designed such that:

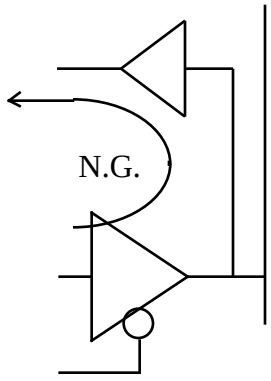
- ï the system chip integration can be accomplished with as much flexibility as possible without affecting the functionality or performance of the VC; and,
- ï the VC operations do not affect the overall operation of the chip in any unexpected or unplanned ways.

The goal of good I/O port design should be to isolate the two levels of hierarchy from undesirable affects on each other. The following are recommendations

1. Re-entrant outputs should be avoided



N.G.(re-entrant) OK(non-re-entrant)



OK(except N.G. case)

Figure 3.3-a Re-entrant Outputs

2. Through nets should be avoided

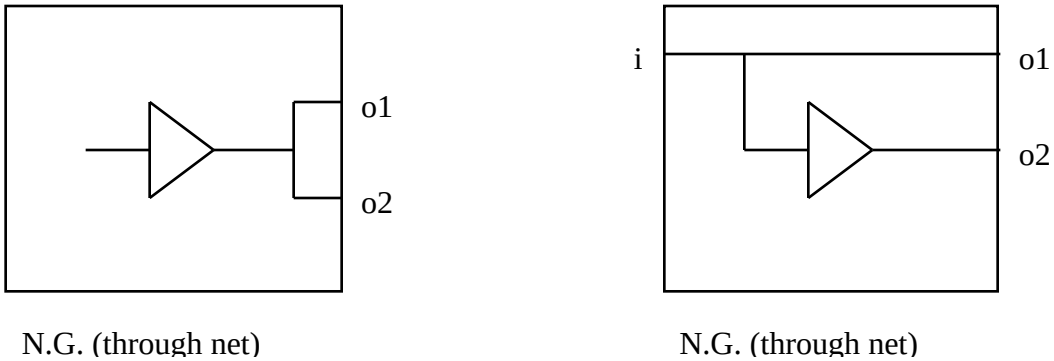


Figure 3.3-b Through Net Rule

3. When tri-state capable output or bi-directional ports are used, tri-state enable/bi-directional control should be available at the VC boundary.

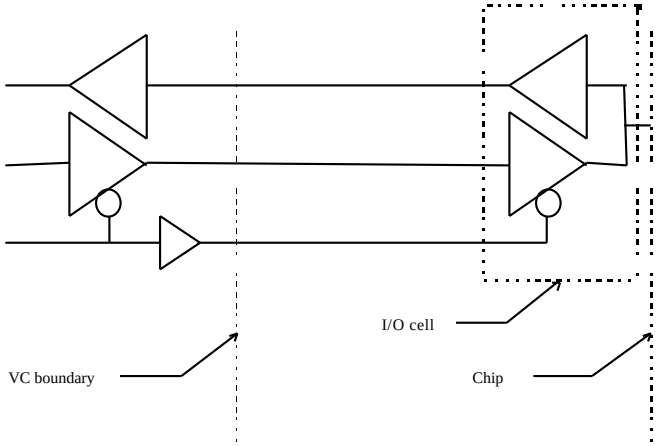


Figure 3.3-c Tri-State Control Available at The VC Boundary

4. The VC user is responsible for providing hold cells for the chip level busses

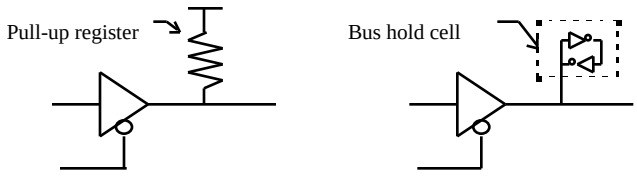


Figure 3.3-d Bus Hold Cell

5. Bi-directional I/O cells should be divided as separate input and output cells if possible, as timing calculation becomes more difficult.

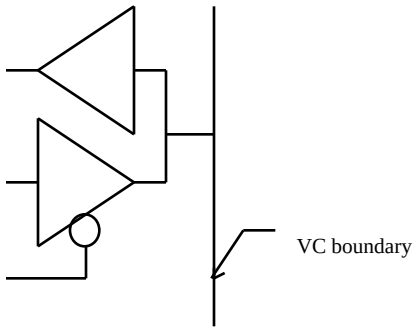


figure (1)

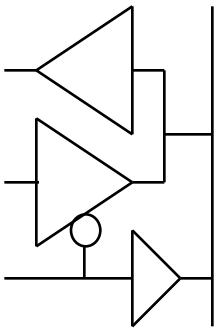


figure (2)

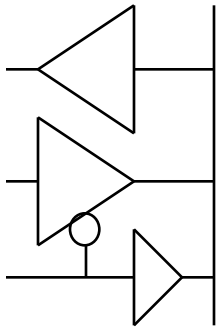


figure (3)

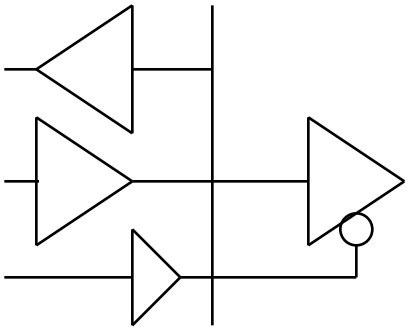


figure (4)

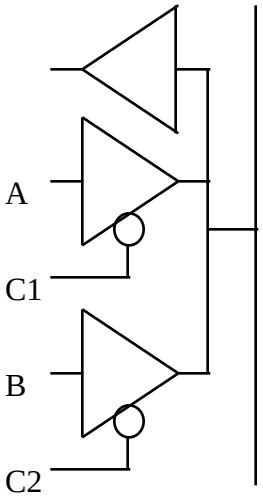


figure (5)

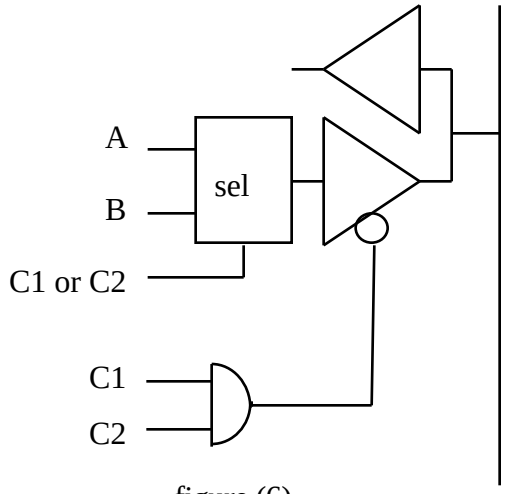


figure (6)

Figure 3.3-e Dividing Bi-Directional

6. All reset and preset signals (synchronous and asynchronous) shall be active low at the periphery of the macro

Clocking Guidelines

The system chip should provide synchronized clocking at the system-level. The system chip clock designer will need to provide the clocking signal(s) to the VC for this purpose. The VC designer must, therefore, provide clock access to the VC and pass along to the system designer the appropriate VC clock characteristics to enable overall chip clock synchronization. Recommendations to accomplish the above are:

- Synchronous designs are preferred.
- Single clock, single phase clock architectures are preferred.
- Clock port access should be properly identified by the VC provide.
- Clock characterization of the VC should be provided to the system chip designer. This should describe the valid conditions for VC clock operation — such as valid frequencies, skew requirements, duty cycle, and pulse width.
- The VC designer should provide information on any special clocking requirements.
- When asynchronous signals are mixed with synchronous signals, the internal block design structure should separate synchronous and asynchronous domains. The interface relationships between the blocks should be documented.

Clocking Guidelines — Soft VC

- It is recommended that no clock distribution circuits be included as part of a Soft VC.
- It is a general recommendation that all Soft VCs be “characterized” against a reference implementation library. When this is done, a description of the clock distribution tree, the clock skew achieved, and the clock delays observed in the reference implementation should be included.

Clocking Guidelines — Hard and Firm VCs

- It is recommended that Hard and Firm VCs include the clock distribution network as part of the deliverable. Furthermore, if additional clock frequencies or phases are being developed in the VC from the clock I/O on the VC, they must be clearly defined and described in detail. An alternative is to provide the additional clock circuitry as a companion VC to be used to provide the unique frequency or conditions.
- The internal clocking distribution should be described. At a minimum this should include internal clock delay and skew.
- The tradeoff between clock delay and clock skew is the responsibility of the Hard VC designer. Caution must be exercised to assure that a proper balance is achieved and also that on-chip variations are considered.
- Many different clocking schemes can be utilized to meet aggressive performance or power objectives in a VC. When

methods such as gated clocks, multiple frequency, multiple pulse width, multi-phase or asynchronous circuits are employed the VC provider must provide detailed documentation on the operation of the clock circuitry and the legitimate frequencies at which the design can be run.

Timing

Internal timing of the VC is the responsibility of the VC designer, as is the chip-level timing of the system chip the responsibility of the system chip designer. Recommendations for the timing of the interface between the system chip and the VC are:

- Static timing analysis is the preferred characterization method for VCs. When there are "don't care" paths or conditions in the circuit these must be described (in terms of clocking conditions and path) to permit successful static timing analysis.
- It is recommended that the VC designer either follow strict rules in the design and placement of the I/O port or a peripheral timing model be provided with the VC.

3.3.2. Physical Design Integration

Predictability and controllability are required for success of system chip physical designs. Performance factors — such as size, timing, and power — must be properly considered for system chip success. These attributes need to be properly accounted for at system chip integration with guidelines written to bridge the level of hierarchy between the system chip and the VC. The three different types of VC require different types of guidelines. Hard VCs should have been designed to be consistent with system chip integration, and be accompanied by proper documentation and specifications. Soft and Firm VCs should provide pre-analysis results and physical design control factors.

Physical I/O Ports

The VC port is the pad or point of interconnection between the VC and the system chip. The location and dimensions are defined in the physical abstract.

- It is recommended that the VC port and I/O buffer be located near the edge of the VC, with access layers defined in order to simplify timing between the VC and the chip, and for accuracy and repeatability.
- There should be direct access to the pin, at least on the layer defined, but preferably in the direction the routing normally occurs on that layer as well.
- To avoid additional routing channels, all ports must be positioned so that they can be wired to the intended grid. This means that no more than three adjacent pins can be spaced less than the external grid they are intended to connect to.
- The supplier of a Hard VC should specify ports which connect to the external chip I/O pad and the appropriate electro-static discharge protection devices, taps, and other reliability and manufacturability structures used.

- Any inputs which have diode or substrate connections on first metal should be separately listed. This will inform the chip integrator that he need not be concerned about antenna problems with these inputs.
- GDSII labels on all I/O ports with their pin names are required to adjust them to the grid.

Porosity and Blockage

A blockage file provided with the VC designates those regions where system chip level routing is allowed. This file should include all areas there are wires and the possibility of error due to crosstalk.

- Timing models should include the assumed capacitive effects of feed throughs through non-blockage points. In general, non-blocked areas should be in the same form and direction as the preferred routing direction for that layer.
- Blockages should be included for all possible layers.
- Sufficient porosity should be created for the design. One logical minimum is the minimum number of feed-throughs that are likely required if the VC is placed on an edge or corner of the die. More feed-throughs will probably be needed for VC placement anywhere else in the system chip, but the amount is a function of the size of the resulting system chip and the location of the core within the system chip.
- It is recommended that this file ensure the blockage of chip-level interconnect routing from critical areas, such as noise or timing, since they could affect the implementation of the VC.

Physical Structure (Hard VC)

- Hard VCs are recommended to be rectangular.
- All process layers should be specified in the GDSII, even blank layers since blockages are as important as wires in the usage of the core.
- Hard VCs should be self contained as much as possible. This means all substrate taps required for the power usage of the VC should be included within the VC, or any field isolation for different power supplies should be wholly contained within the VCs boundary.

Power Access

VCs will have various current requirements and will be designed with specific voltage drop expectations. Further, they may have multiple voltage requirements or special power requirements.

- It is required that power requirements of VCs be carefully described by the VC provider.
- On flip chips where solder ball grids are to be used for the chip, the VC must provide the appropriate power pads on grid for access to power.

- The VC must provide power access contacts that are sufficient to meet both current and voltage drop requirements of the VC. It is recommended that special requirements be appropriately described for the system chip integrator.
- Special power access must be properly described and defined, as appropriate, in LEF/DEF format.
- The maximum power requirement should be stated (at least be defined for the specific process the GDSII was originally targeted for).
- The resulting operating voltage de-rating from nominal resulting from IR drop should be noted to the user of the VC for proper timing design.
- The power and ground connections should be on the edge of the VC with layers specified.
- The connection locations should be specified along with the width of available connection on that edge. There should be a suggested minimum total width of power and ground connections required for each VC, and the VC should have sufficient margin for integration.
- For VCs which have multiple power supplies the different supplies access points should be clearly labeled and differentiated from one another.

Shapes

Today's chip technologies have finely tuned photo-lithographic processes. This is especially true for the more advanced (higher performance, high density) technologies. This results in rules for critical spacing and geometry shapes. Most shape rules require rectilinear geometries with possibly some conditional exceptions and most critical spacing/sizing rules are consistent across a chip with possibly some conditional exceptions.

It is recommended that VC providers use only rectilinear geometries and generally consistent critical spacing rules. Only when a VC design is intended to be used for one process should technology specific rules be considered.

Grid

Today's design systems and routers can use the layout grid in various ways. It is expected that physical VCs will come in a variety of grid styles.

It is recommended that the system chip level integration design systems be able to accommodate grided, gridless, or a mixture. The physical VC supplier should provide a boundary description in physical dimensions (e.g., microns rather than grids).

Appendices

A. VC Reserved Words

For Virtual Component design we recommend that authors refrain from use of the following set of keywords.

A1. Verilog

<all reserved keywords as described in OVI Verilog Language Reference>

A2. VHDL

<all reserved keywords as described in IEEE1076.X Language Reference>

1. Windows NT
   - CON
   - AUX
   - COM1
   - COM2
   - COM3
   - COM4
   - LPT1
   - LPT2
   - LPT3
   - PRN
   - NUL
   - .
   - ..
   - \
   - Non-printable and control characters (0x1F-0x7F, del)

2. Unix

   - .
   - ..
   - /
   - Non-printable and control characters (0x1F-0x7F, del)

B. Name Space Descriptions and Transformation Rules

The following proposes a set of definitions for name spaces for EDA vendors to support, as well as transformation rules for mapping between name spaces. Implementing these name spaces transformation capabilities will support ease of integration of Virtual Component information produced by tools from different vendors.

B.1. Name Space Descriptions

- A raw string of characters (represented with uchar or ushort (UNICODE))
- String is terminated with the NULL character (value 0)
- Case sensitive
- No character restrictions
- Allowed characters are currently ASCII values 1 to 255
- In the future, to support UNICODE, character values will be 1 to 2^16-1

B.2. File System

- UNIX is case sensitive
- *** NT is case INSENSITIVE but case preserving ***
- Forward slash and back slash not allowed (back slash because of NT)
- White space not allowed
- Characters other than alphanumerics and underscore are discouraged (we are allowing dash and dollar sign to be used in names)
- NT does not allow filenames that match DOS devices, and thus the following names are reserved in NT: CON, AUX, COM1, COM2, COM3, COM4, LPT1, LPT2, LPT3, PRN, NUL
- Let N_Filesystem be the set of characters that will be escaped in file system names. N_file system includes all characters which are not a letter, digit, underscore, dash and dollar sign. All characters greater than 127 are in N_file system.
- N_file system includes the pound sign and percent character.

- (Note that we want UNIX and NT to share the same name space since we want to be able to tar a library on UNIX, untar it on NT (or vice-versa), and have the library work in the new environment. This requirement leads us to treat the combined file system name space as case INSENSITIVE, but it also requires that we NOT use case preservation due to UNIX's case sensitivity.)

B.3. VHDL

- Normal and Escaped identifiers
- Normal identifiers are case insensitive
- Escaped identifiers are case sensitive
- Escaped identifiers can contain any graphic character, and spaces
- Escaped identifiers begin and end with \ (backslash)
- To embed \ (backslash) in an escaped identifier, use \\
- Equivalent normal and escaped identifiers denote different objects.
- VHDL keywords must be escaped
- Normal identifiers start with alpha, contain alphanums and underscores
- All objects share the same name space (signals, instances, etc.)
- Let N_VHDL be the set of characters which will be escaped VHDL characters. N_VHDL includes all characters which are not in the graphic character set defined in chapter 13 of the VHDL 93 LRM. N_VHDL includes all characters greater than 255.

B.4. Verilog/Sensitive

- Normal and Escaped identifiers
- Normal identifiers are case sensitive
- Escaped identifiers are case sensitive
- Escaped identifiers can contain any graphic character, but no spaces
- Escaped identifiers begin with \ and are terminated by white space
- Verilog keywords must be escaped
- Normal identifiers contain letters, digits, dollar sign, and underscore
- Normal identifiers cannot start with digit or dollar sign.
- Equivalent normal and escaped identifiers denote the same objects.
- All objects share the same name space (signals, instances, etc.)

- Let N_Verilog be the set of characters which will be escaped Verilog characters. N_Verilog includes non-graphic characters, whitespace characters, and characters greater than 127.

B.5. Verilog/Insensitive (Verilog -u option)

Same as Verilog/Sensitive except that normal and escaped identifiers are case insensitive.

B.6. Name space Transformation Rules

The next sections outline the name space transformation rules for mapping between the name spaces defined above.

Note: In the rules below:

- #XX is used to map an “objectionable” character in the ASCII range 0 to 255.
- ##XXXX is used to map an “objectionable” character in the UNICODE range 256 to (2^16-1).
- X will be one of the characters 0-9, a-f. Note that a-f will always be in lowercase. The #XX and ##XXXX character sequence are referred to as “pound-hex groups” in the rules below.

B.7. Raw -> Verilog/Sensitive

If the raw identifier is a legal Verilog normal identifier, it is represented as such. Else, if the raw identifier contains any characters in N_Verilog, these characters are replaced with pound-hex groups. The resulting string is represented as a Verilog escaped identifier (that is, it is prefixed with backslash).

B.8. Verilog/Sensitive ->Raw

If the Verilog identifier is not escaped, the identifier is converted as-is into the raw identifier. If the identifier is escaped, the leading \ is removed, any pound-hex groups which in fact represent characters in N_Verilog are replaced by their single character equivalents, any pound-hex groups which do not represent characters in N_Verilog are left as-is, and the resulting string is the raw identifier.

B.9. Raw ->Verilog/Insensitive

Let ESC_VLOGUP be the character ^ (caret).

Any uppercase characters are preceded with the ESC_VLOGUP prefix. If the resulting string is a legal Verilog normal identifier, it is represented as such. Else, if the resulting string contains any characters in N_Verilog, these characters are **replaced** with pound-hex groups. The resulting string is represented as a Verilog escaped identifier (that is, it is prefixed with \).

B.10. Verilog/Insensitive ->Raw

If the Verilog identifier is not escaped, the identifier is copied to a temp string. Else, the leading \ is removed, any pound-hex groups which in fact represent characters in N_Verilog are replaced by their single character equivalents, any pound-hex groups which do not represent characters in N_Verilog are left as-is, and the resulting string is copied to a temp string.

Next, all letters A-Z in the temp string are lowercased unless they are preceded with the ESC_VLOGUP prefix, in which case the ESC_VLOGUP is removed and the letter is uppercased. The identifier then is represented as-is as a raw identifier.

B.11. Raw ->VHDL

Let ESC_VHDL be the six character sequence ESC, V, H, D, L, ESC, where ESC represents the escape character (ASCII hex value 1B).

If a raw identifier is prefixed with ESC_VHDL,

{

Then the ESC_VHDL sequence is removed, backslashes are doubled, and the remaining characters are represented as-is as a VHDL escaped identifier,

}

else If (the raw identifier

contains no uppercase letters

begins with alpha, contains only alphanums and underscores, and does not contain a trailing or duplicate underscore and

is not a VHDL keyword)

{

Then it is represented as a normal VHDL identifier

}

else

{

Any characters in N_VHDL are replaced with pound-hex groups, backslashes are doubled, and the resulting identifier is represented as an escaped VHDL identifier in its original case. (That is, it begins and ends with backslash).

}

B.12. VHDL ->Raw

If a VHDL identifier is represented escaped {

Unescape the identifier, turn double backslashes into single backslashes. Replace any pound-hex groups which in fact represent characters in N_VHDL with their single character equivalents. Leave and pound-hex groups which represent characters not in N_VHDL as-is.

If the resulting string needn't be escaped according to the rules of VHDL

{

Prefix the identifier with ESC_VHDL

}

return the resulting string

}

else

{

lowercase the identifier and return it

}

B.13. Raw ->File System

Any characters in N_file system are replaced with a pound-hex group. Every uppercase character is preceded with % and is left in uppercase. If the resulting name is a NT/DOS reserved device name (i.e.: CON, AUX, COM1, COM2, COM3, COM4, LPT1, LPT2, LPT3, PRN, NUL) then % is appended to the name.

- Example Raw -> file system mappings:
- foo foo
- Foo %Foo
- Hi There %Hi#20%There
- con con%
- nec$f04 nec$f04

Note that these rules imply that “Foo” is not a legal file system name.

B.14. File System ->Raw

Replace every pound hex group that in fact represents a character in N_file system with the corresponding character. Leave any pound hex group that does not represent a character in N_file system as-is. Replace every occurrence of % followed by a letter by the same uppercase letter. Any trailing % is removed. All other characters are translated in lowercase. The resulting string is the raw identifier.

B.15. Case Preservation

When mapping between two case insensitive name spaces, it is often desirable (but never strictly the case of identifiers. Note that case preservation is not an issue when either name spaces is case sensitive.

To support case preservation, the name mapping routines invisibly associate an additional piece of data with a RAW identifier -- this additional piece of data is termed the “case preservation string”. The case preservation string is a normal null terminated string which has the same length as the RAW identifier. The characters of the case preservation string are one of U, L, and ?, indicating that the original case of the corresponding character in the RAW identifier is either upper, lower, or of unknown case.

The case preservation capability provides only aesthetic benefits. The fact that case preservation occurs has no semantic effect.

An example of case preservation follows. Assume an application needs to map the identifier FooBar from the VHDL name space to the Verilog/Insensitive name space. The following transformations would occur:

- VHDL: FooBar
- Raw: foobar -- identifier
- ULLULL -- case preservation string
- Verilog/Insensitive: FooBar

Note that an application which calls the name mapping routines will have no knowledge of the use of the case preservation string since its use is completely internal to the name mapping routines.

Note that the case preservation capability is intentionally turned off when mapping to or from the file system namespace. Because we want to treat the file system name space as a single unified name space, and because UNIX is in fact case sensitive while NT is case insensitive, it is required that case preservation not occur in this mapping.

Examples

- Convert an Verilog/Sensitive identifier “buffer” into the VHDL name space.
  - Verilog/Sensitive -> Raw
  - buffer -> buffer
  - Raw -> VHDL
  - buffer -> \buffer\ -- since “buffer” is a VHDL keyword
- Convert VHDL identifier FOO into Verilog/Sensitive name space
  - VHDL -> Raw
  - FOO -> foo
  - Raw -> Verilog/Sensitive
  - foo -> foo
- Find the file system cell directory for the Verilog module named “Foo”
  - Verilog -> Raw
  - Foo -> Foo
  - Raw -> file system
  - Foo -> %Foo
- Cross-name space references work sensibly. For example, if a VHDL entity has ports named “FOO” and “WIRE”, and if this entity is instantiated in a Verilog module, then the ports can be referenced in

Verilog by the names “foo” and “\wire”. No additional information is required.

- As another example, if a Verilog/Sensitive name space path to an instance is a.b.c.A where “a” is the root instance and “a” and “A” are different modules due to Verilog’s case sensitivity, then the corresponding path expressed in the VHDL name space would be \A\

Again, no additional information is required to indicate that a particular identifier exists in one name space or another. Note that this path may represent a mixed design hierarchy containing both Verilog and VHDL instances.

C. Open Standards References

- IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual.
- ANSI/IEEE Std 1076-1993 (Revision of IEEE Std 1076-1987), IEEE Standard VHDL Language Reference Manual.
- IEEE Std 1164-1993, IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164).1
- IEEE Std 1364, Verilog Hardware Description Language Reference Manual.
- OVI Standard Delay Format Specification, Version 2.1-3.0.
- OVI-CFI Standard Delay Calculation System Specification, Version 1.0.
- ISO/IEC 9899-1990, The Programming Language C.

C.1. Reference Source Locations

- IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.
- IEEE Std 1076-1987 has been superseded by IEEE Std 1076-1993. IEEE Std 1076-1987 is no longer in print; however, it is available from the IEEE.
- ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.
- OVI publications are available from Open Verilog International (OVI), 15466 Los Gatos Blvd., Suite 109-071, Los Gatos, CA 95032.
- CFI publications are available from CFI, 4030 West Braker Lane, Suite 550, Austin, TX 78759, USA.

D. Glossary of Acronyms

ATPG	Automatic Test Pattern Generation
BFM	Bus Functional Model
BIST	Built-In Self Test
BIST	Built-In Self Test
DEF	Design Exchange Format
DFT	Design-For-Test
DSM	Deep Submicron
DSP	Digital Signal Processor
ESPF	Extended Standard Parasitic Format
GCF	General Constraint Format
GIF	Geometric Image Format
HDL	Hardware Design Language
IDDQ	Quiesent drain current
ISA	Instruction Set Architecture
ITL	Interpolated Table Lookup-format
LBIST	Logic Built-In Self Test
LEF	Layout Exchange Format
NLDM	Non-Linear Delay Model
OMF	Open Model Forum
PDEF	Physical Design Exchange Format
PLI	Procedural Language Interface
RTL	Register Transfer Language
SDF	Standard Delay Format
TAP	Test Access Port
TLF	Timing Library Format
VC	Virtual Component
VCD	Verilog Change Dump
VERILOG	Verilog (VHDL)
VHDL	VHSIC Hardware Design Language
WGL	Waveform Generation Language