IBIS

(I/O Buffer Information Specification)

Version 8.0

Ratified December 5, 2025

© IBIS Open Forum 2025

Contents

1 General Introduction 9
2 Statement of Intent 11
3 General Syntax Rules and Guidelines 17
3.1 File Naming Definitions 17
3.2 Syntax Rules 18
3.3 General Terminology Definitions 21
3.4 Keyword Hierarchy 22
4 File Header and File End Information 32
5 Component Description 35
6 Buffer Modeling 62
6.1 Model Statement 62
6.2 Add Submodel Description 117
6.3 Multi-Lingual Model Extensions 131
6.3.1 Introduction 131
6.3.2 Languages Supported 132
6.3.3 Overview 132
6.3.4 Definitions 133
6.3.5 General Assumptions 133
6.3.6 Keyword Definitions 138
6.4 Test Load and Data Description 175
6.4.1 Introduction 175
6.4.2 Keyword Definitions 175
7 Package Modeling 179
7.1 Introduction 179
7.2 Rules of Precedence 179
7.3 Keyword Definitions 179
8 Electrical Board Description 195
8.1 Introduction 195
8.2 Keyword Definitions 196
9 Notes on Data Derivation Method 205
10 Algorithmic Modeling 211
10.1 Algorithmic Modeling Interface (AMI) 211
10.1.1 Introduction 211
10.1.2 Keyword Definitions 213
10.2 AMI Executable Model File Programming Guide 215
10.2.1 Overview 215
10.2.2 Application Scenarios 216
10.2.3 Function Signatures 222
10.2.4 Code Segment Examples 235
10.3 AMI Parameter Definition File Structure 236
10.3.1 Introduction 236
10.3.2 AMI Parameter Definition File Organization 236
10.3.3 Parameter Rules Summary 237
10.3.4 Reserved Word Rules 239
10.3.5 Combination and Corner Rules 245

10.3.6 Processing and Passing Parameter String Rules 246
10.3.7 Summary Table for Type and Format 247
10.4 General Reserved Parameters 248
10.4.1 Summary Tables for Usage, Type and Format 260
10.5 Reserved Parameters for Data Management 263
10.5.1 Summary Tables for Usage, Type and Format 265
10.6 Jitter and Noise Reserved Parameters 266
10.6.1 Tx-only Reserved Parameters 267
10.6.2 Rx-only Reserved Parameters 271
10.6.3 Summary Tables for Usage, Type and Format 281
10.7 Modulation Reserved Parameters 285
10.7.1 Summary Tables for Usage, Type and Format 293
10.8 Repeaters 295
10.8.1 Summary Tables for Usage, Type and Format 297
10.8.2 Repeater Time-Domain Simulation Flow For A Repeater Link 298
10.9 AMI Reserved Parameter Definitions For Link Training Communications 304
10.9.1 Training Flows 310
10.9.2 Summary Tables for Usage, Type and Format 317
10.10 Alternative AMI Analog Buffer Modeling 319
10.10.1 Transmitter Analog Circuit 319
10.10.2 Receiver Analog Circuit 319
10.10.3 Reserved Parameter Definitions 321
10.10.4 Summary Tables for Usage, Type and Format 325
10.11 AMI Test Configuration 327
10.11.1 Keyword Definition 327
10.12 Model Specific Parameters 336
10.12.1 Tapped Delay Line Example 336
10.13 Reserved Parameter and Data Type Rule Summary Tables 338
11 Interconnect Modeling 352
11.1 Introduction 352
11.2 General Interconnect Syntax Requirements 356
11.2.1 Connecting Pins, Pads and Buffer Terminals 368
12 Electrical Module Description (EMD) 388
12.1 Introduction 388
12.2 EMD Files 389
12.3 Keyword Definitions 390
13 EMD Set And EMD Model Description 400
13.1 EMD Set Keyword Description 400
13.2 General EMD Set and EMD Model File Syntax Requirements 401
13.3 General EMD Model Keyword Description 403
13.4 Terminal_Type Associations for EMD and Designator Pins 408
13.5 RDIMM Example Illustrating Syntax and Net Options 411
13.5.1 RDIMM Figures for Examples in 13.5.2 Through 13.5.4 411
13.5.2 Example 1 (R123 and RN13 Embedded in A07_1 and BA07_1) 412
13.5.3 Example 2 (R123 and RN13 Modeled as Separate IBIS Components in A07_2, BA07_2) 414

13.5.4 Example 3 (R123 IBIS Model Terminals Split into Two [EMD Model]s, POWER Rails in a Separate [EMD Model]) 416
13.6 Connection Rules for EMD Group, EMD Set, and EMD Model 419
13.7 Additional EMD Model Examples 421
14 EMI Parameters 426

Figures

Figure 1 – Example of File Naming Definitions 18
Figure 2 – [PDN Model] Circuit 47
Figure 3 – Reference Load Connections 65
Figure 4 – Single-Ended or True Differential Buffer 65
Figure 5 – Receiver Voltage with Hysteresis Thresholds 68
Figure 6 – Receiver Voltage with Static and Dynamic Overshoot Limits 69
Figure 7 – Receiver Voltage with Dynamic Area Overshoot Limits 70
Figure 8 – Receiver Voltage with Pulse Immunity Thresholds 71
Figure 9 – C Comp Model Structure 89
Figure 10 – Low State (Logic Zero) Isso_pd Data Collection 95
Figure 11 – High State (Logic One) Isso_pu Data Collection 96
Figure 12 – Reference Data Collection 97
Figure 13 – Reference Data Collection with Supply Modulation 97
Figure 14 – [Rgnd], [Rpower], [Rac], [Cac] in Relation to Package and Buffer Data 100
Figure 15 – Series Element Associations 102
Figure 16 – [Series Current] Voltage Polarity and Current Direction 103
Figure 17 – [Series MOSFET] Voltage Polarities and Current Direction 104
Figure 18 – [Rising Waveform] and [Falling Waveform] Fixtures 108
Figure 19 – [External Reference] - Used Only for Non-driver Modes 111
Figure 20 – [Composite Current] Internal Current Paths 112
Figure 21 – [GND Pulse Table] Waveforms at Die 121
Figure 22 – Port Names for I/O Buffer 135
Figure 23 – Port Names for Series Switch 135
Figure 24 – Example Showing [External Circuit] Ports 136
Figure 25 – AMS Model Unit, Using an I/O Buffer as an Example 137
Figure 26 – An Analog-Only Model Unit, Using an I/O Buffer as an Example 138
Figure 27 – Multi-lingual [External Model] I/O Buffer 145
Figure 28 – Multi-lingual Pseudo-differential I/O Buffer 147
Figure 29 – Multi-lingual *-AMS I/O Buffers 148
Figure 30 – Port Names for True Differential I/O Buffer 149
Figure 31 – Multi-lingual True Differential Buffer 150
Figure 32 – Reference Example for [Node Declarations] Keyword 172
Figure 33 – [Test Load] Elements and Placement 177
Figure 34 – Package Matrix Voltage Polarities and Current Directions 188
Figure 35 – SIMM Package Path Example 200
Figure 36 – Fork and Endfork in [Path Description] 201
Figure 37 – Discrete Series Element in [Path Description] 201
Figure 38 – Series Passive Components as Differential Termination 202
Figure 39 – Paths Connected by Series Resistors as Differential Terminators 203
Figure 40 – Example of TTgnd Extraction Setup 209
Figure 41 – Example of Series MOSFET Table Extraction 210
Figure 42 – Examples for Rx_Use_Clock_Input 259
Figure 43 – Repeater Model 295
Figure 44 – Repeater Link 298
Figure 45 – Redriver AMI_Init Execution for Various Tx_Impulse_Input Settings 300

Figure 46 – Transmitter Analog Circuit, Default Port Ordering 319
Figure 47 – Receiver Analog Circuit, Default Port Ordering 320
Figure 48 – Example Interconnect Model Structure 352
Figure 49 – Package and On-die Substrate I/O Paths 354
Figure 50 – Package Substrate Rail Terminals 355
Figure 51 – Aggressor_Only Examples 367
Figure 52 – A Special Case with Aggressor_Only 368
Figure 53 – Electrical Connections for Full Buffer Pin Model with Power Routing 374
Figure 54 – Electrical Terminals for Full Buffer Pin Model with Power Routing 375
Figure 55 – DDR4 Registered DIMM with Labeling 411
Figure 56 – Extended Net 412
Figure 57 – Internal Net 412

Tables

Table 1 – Special Rules for Keyword [Model] 62
Table 2 – Scheduled Model Initial State 79
Table 3 – Terminal_type Definitions 88
Table 4 – Example of Setting Isso_pu and Isso_pd Values 98
Table 5 – Bus Hold without Off_Delay – Initialization 124
Table 6 – Bus Hold without Off_Delay – Transitions 125
Table 7 – Bus Hold with Off_Delay – Initialization 125
Table 8 – Bus Hold with Off_Delay – Transitions 125
Table 9 – Fall Back, Initial State 128
Table 10 – Fall Back, Driver Rising Cycle 128
Table 11 – Fall Back, Driver Falling Cycle 129
Table 12 – Language Extension Keywords 131
Table 13 – Port Names in Multi-Lingual Modeling 134
Table 14 – Required Port Names for Single-ended Model_type Assignments 152
Table 15 – Required Port Names for Differential Model_type Assignments 152
Table 16 – Package Modeling Keywords 180
Table 17 – Voltage Ranges 206
Table 18 – Allowable Data Types for Format Values 247
Table 19 – General Rules and Allowable Usage for General Reserved Parameters 260
Table 20 – Allowable Data Types for General Reserved Parameters 261
Table 21 – Allowable Data Formats for General Reserved Parameters 262
Table 22 – General Rules and Allowable Usage for Supporting Files Reserved Parameters 265
Table 23 – Allowable Data Types for Supporting Files Reserved Parameters 266
Table 24 – Allowable Data Formats for Supporting Files Reserved Parameters 266
Table 25 – General Rules and Allowable Usage for Jitter and Noise Reserved Parameters 281
Table 26 – Allowable Data Types for Jitter and Noise Reserved Parameters 282
Table 27 – Allowable Data Formats for Jitter and Noise Reserved Parameters 283
Table 28 – General Rules and Allowable Usage for Modulation Reserved Parameters 293
Table 29 – Allowable Data Types for Modulation Reserved Parameters 293
Table 30 – Allowable Data Formats for Modulation Reserved Parameters 294
Table 31 – General Rules and Allowable Usage for Repeater Reserved Parameters 297
Table 32 – Allowable Data Types for Repeater Reserved Parameters 297
Table 33 – Allowable Data Formats for Repeater Reserved Parameters 297
Table 34 – General Rules and Allowable Usage for BCI Reserved Parameters 317
Table 35 – Allowable Data Types for BCI Reserved Parameters 317
Table 36 – Allowable Data Formats for BCI Reserved Parameters 318
Table 37 – General Rules and Allowable Usage for Alternative Analog Modeling Reserved Parameters 325
Table 38 – Allowable Data Types for Alternative Analog Modeling Reserved Parameters 325
Table 39 – Allowable Data Formats for Alternative Analog Modeling Reserved Parameters 326
Table 40 – [AMI Test Configuration] Subparameter Rules 333
Table 41 – Reserved Parameters and Supported AMI_Versions 338
Table 42 – General Rules and Allowable Usage for Reserved Parameters 340
Table 43 – Allowable Data Types for Reserved Parameters 342
Table 44 – Allowable Data Formats for Reserved Parameters 344

Table 45 – Allowable Data Types for Format Values 347
Table 46 – Defined Directions for Reserved Parameters 347
Table 47 – [Algorithmic Model] Subparameter and [Model] Model_type Interaction 350
Table 48 – Interconnect Modeling Keywords and Subparameters 356
Table 49 – Allowed Terminal_type Associations for Interconnect Models[1] 370
Table 50 – EMD Set and EMD Model Keywords and Subparameters 401
Table 51 – Allowed Terminal_type Associations for EMD Models[1] 409

1 GENERAL INTRODUCTION

This section gives a general overview of the remainder of this document.

Sections 2 and 3 contain general information about the IBIS versions and the general rules and guidelines. Several progressions of IBIS documents are referenced in Section 2 and in the discussion below. They are:

- IBIS Version 1.1 (ratified August 20, 1993)
- IBIS Version 2.1 (ratified as ANSI/EIA-656 on December 13, 1995)
- IBIS Version 3.2 (ratified as ANSI/EIA-656-A on August 20, 1999 and renewed on August 17, 2005)
- IBIS Version 4.2 (ratified as ANSI/EIA-656-B on March 1, 2007)
- IBIS Version 5.0 (ratified on August 29, 2008)
- IBIS Version 5.1 (ratified on August 24, 2012)
- IBIS Version 6.0 (ratified on September 20, 2013)
- IBIS Version 6.1 (ratified on September 11, 2015)
- IBIS Version 7.0 (ratified on March 15, 2019)
- IBIS Version 7.1 (ratified on December 10, 2021)
- IBIS Version 7.2 (ratified on January 27, 2023)
- IBIS Version 8.0 (ratified on December 5, 2025)

The functionality of IBIS follows in Section 4 through Section 14. Sections 3.2 through 6 describe the IBIS Version 1.1 core functionality and its extensions in later versions. The data in these sections are contained in .ibs files. Section 7 describes the package model format of IBIS Version 2.1 and a subsequent extension. Package models can be formatted within .ibs files or can be formatted (along with the Section file header keywords) as .pkg files. Section 8 contains the Electrical Board Description (EBD) format introduced in IBIS Version 3.2. Along with Section 4 header information, electrical board descriptions must be contained in separate .ebd files.

The content in Sections 10 and 14 was introduced in IBIS Version 5.0 and contains reference and modeling information related to algorithmic modeling interface (AMI) support, and electromagnetic interference (EMI) parameters. The content in Sections 6.4 and 10.3 was introduced in IBIS Version 5.1, to place test loads and data appropriately in the keyword hierarchy and to more fully describe algorithmic models, respectively. Repeater support was added in Section 10.8 as part of IBIS Version 6.0, including repeater keywords, AMI parameters, and data flow. IBIS Version 6.0 also modifies the organization of the document. Data modulation was added as Section 10.7 in IBIS Version 6.0. Support for dependent AMI parameters was added in an expanded Section 10.2.2. Section 10 was expanded to support PAMn (pulse amplitude modulation with an arbitrary number of levels) in IBIS 7.2.

The content in Section 11 was added in IBIS 7.0 to describe interconnect modeling, expanding package descriptions as well as introducing support for on-die interconnect descriptions. Link training (i.e., backchannel) communications and alternative AMI analog buffer modeling support were added in Sections 10.9 and 10.10, respectively. Additionally, more rigorous file naming rules were defined in a new Section 3.1 as part of IBIS 7.0.

Sections 12 and 13 were introduced as part of IBIS 7.1 to support the Electrical Module Description (EMD) format, as an alternative to the EBD format and an expansion of its capabilities.

IBIS 7.1 also expanded Sections 10.8 and 10.9 to support statistical back-channel adaptive equalization. Other areas of Section 10 were updated to help AMI support single-ended buffer modeling. The core functions of IBIS were expanded under IBIS 7.1 to include, among other features, the ability to model C_comp using data in IBIS-ISS or Touchstone format. Additional keywords supporting on-die power distribution network modeling were added to Section 5.

Section 9 contains some notes regarding the extraction conditions and data requirements for IBIS. This section focuses on implementation conditions based on measurement or simulation for gathering the IBIS compliant data.

A section is added in IBIS 8.0 to support the AMI Test Configuration keyword.

2 STATEMENT OF INTENT

In order to enable an industry standard method to electronically transport modeling data between semiconductor vendors, electronic design automation (EDA) tool vendors, and end customers, the IBIS syntax was developed. The intention is to specify a consistent format that can be parsed by software, allowing EDA tool vendors to derive models compatible with their own products.

One goal of the format is to represent the current state of model data, while allowing a growth path to more complex models/methods (when deemed appropriate). This would be accomplished by a revision of the format, with the addition of new keywords or categories.

Another goal of the format is to ensure that it is simple enough for semiconductor vendors and customers to use and modify, while ensuring that it is rigid enough for EDA tool vendors to write reliable parsers.

Finally, this format is meant to contain a complete description of the I/O elements on an entire component. Consequently, several models will need to be defined in each file, as well as a table that equates the appropriate buffer to the correct pin and signal name.

Version 8.0 of this electronic format was finalized by an industry-wide group of experts representing various companies and interests. Regular "IBIS Open Forum" meetings were held to accomplish this task.

Changes to the specification are proposed and approved through Buffer Issue Resolution Documents (BIRDs). All submitted BIRDs may be viewed through the IBIS Open Forum website, http://www.ibis.org/.

Commitment to Backward Compatibility. Version 1.0 was the first valid IBIS ASCII file format. It represents the minimum amount of I/O buffer information required to create an accurate IBIS model of common CMOS and bipolar I/O structures. Future revisions of the ASCII file added items considered to be "enhancements" to Version 1.0 to allow accurate modeling of new, or other I/O buffer structures. Consequently, all future revisions are considered supersets of Version 1.0, allowing backward compatibility. In addition, as EDA tools develop support for revisions of the IBIS ASCII format, all previous revisions of the format must also be supported.

Version 1.1. Version 1.1, (published as "ver1_1.ibs") is conceptually the same as the 1.0 version of the IBIS ASCII format (published as "ver1_0.ibs"). However, various comments have been added for further clarification.

Version 2.0. Version 2.0 maintains backward compatibility with Versions 1.0 and 1.1. All new keywords and elements added in Version 2.0 are optional. A complete list of changes to the specification is in the IBIS Version 2.0 Release Notes document ("ver2_0.rn.txt"). Some changes are also documented in 14 BIRDs:

BIRD2.2	Requiring VIH VIL thresholds for input devices
BIRD3	Multiple power supplies and references
BIRD4	ECL Extensions
BIRD5.4	Pin Mapping for Ground Bounce Simulation
BIRD6.2	Differential Pin Specification
BIRD7.2	Open Specification Completion
BIRD8.2	Specification of V/I data monotonicity

BIRD9.3 Terminator Specification
BIRD10.2 Describing coupling effects in package models
BIRD11.2 Improving common error detection in IBIS_CHK program.
BIRD12.2 Non-Linear Driver Waveforms
BIRD13.2 Clarify Some Conditions of Measurements
BIRD14.3 Adding four new sub-parameters to [Model]
BIRD15 Clarification on the usage of the V/I tables.

Version 2.1. Version 2.1 contains clarification text changes, corrections, and two additional waveform parameters beyond Version 2.0 documented in 9 BIRDs:

BIRD18.2 [Diff Pin] Parameter Order
BIRD19.1 V_fixture Subparameter Min/Max Additions
BIRD20.1 Error correction regarding monotonicity statement in V2.1 IBIS Specification
BIRD21 Waveform Table Minimum Number of Entries
BIRD23 Waveform Table Minimum Number of Numerical Entries
BIRD24.1 C_comp, ramp rates and waveform tables
BIRD25.3 Data Derivation Expansion
BIRD26 General syntax rules and guidelines on TAB character usage
BIRD29.2 Banded_matrix Extension

Version 3.0. Version 3.0 adds a number of new keywords and functionality. Some changes are documented in 10 BIRDS:

BIRD28.3 Enhancement To The Package Model (.pak file) Specification
BIRD30.2 Programmable buffers in IBIS models
BIRD34.2 Stored Charge Effects
BIRD35.3 Multi-staged Outputs
BIRD36.3 Electric Descriptions of Boards
BIRD37.3 Enhancement To The Package Model (.pkg file) Specification
BIRD39 Specification Enhancement
BIRD40 Overshoot Nomenclature
BIRD41.8 Modelling Series Switchable Devices
BIRD43 Component Test Point Subparameters

Version 3.1. Version 3.1 contains a major reformatting of the document and a simplification of the wording. It also contains some new technical enhancements that were unresolved when Version 3.0 was approved. Some changes are documented in 2 BIRDS:

BIRD47 Remove pin name as a sub-param of the [Pin List] keyword
BIRD52 [Driver Schedule] Clarifications

Version 3.2. Version 3.2 adds more technical advances and also a number of editorial changes in responses to public letter ballot comments and documented in 13 BIRDs:

BIRD46.1 Relaxation of some IBIS model file name restrictions
BIRD48.4 Add Submodel
BIRD49.4 Add Submodel Dynamic Clamps

BIRD50.3 Add Submodel Bus Hold
BIRD51 3-state_ECL
BIRD53.1 IBIS File Character Set
BIRD54 Package Model Corrections
BIRD55 [Model Spec] Vmeas Addition
BIRD56.1 Relaxation of [Series Pin Mapping] Restriction
BIRD57.1 Timed Bus Hold Extension
BIRD58.3 Driver Schedule Keyword Clarification
BIRD59.2 Model Spec Diagrams
BIRD60 Electrical Board Description Diagrams

Version 4.0. Version 4.0 adds more technical advances and a few editorial changes documented in 11 BIRDs:

BIRD62.6 Enhanced Specification of Receiver Thresholds
BIRD64.4 Alternate Package Models
BIRD65.2 C_comp Refinements
BIRD66 [Model Spec] Vref Addition
BIRD67.1 Increase V-T Table 100 Point Limit
BIRD68.1 Clarify that Rising and Falling Waveforms Should be Correlated
BIRD70.5 Golden Waveforms
BIRD71 Timing Test Loads in [Model Spec] to Support PCI & PCI-X
BIRD72.3 Accommodating PMOS and NMOS//PMOS Series FET Models
BIRD73.4 Fall Back Submodel
BIRD76.1 Additional Information Related to C_comp Refinements

Version 4.1. Version 4.1 adds more technical advances and a few editorial changes documented in 10 BIRDs:

BIRD75.8 Multi-Lingual Model Support
BIRD77.2 Differential Subparameter Additions
BIRD78.1 Comment Line Length Limit
BIRD80.1 Add External Reference Column to Pin Mapping Keyword
BIRD81.1 Clarify Usage Rule for [Pin] I/O Model Assignment
BIRD82.2 Clarification of Clamp Table Use
BIRD83.2 Series Element Clarifications
BIRD84.1 Driver Schedule Clarifications
BIRD85.3 Slew Time Estimate Clarifications
BIRD86.1 Clarification of Submodel Mode

Version 4.2. Version 4.2 adds more technical advances and some editorial changes documented in 13 BIRDs:

BIRD87 Series Pin Mapping Clarifications
BIRD88.3 Driver Schedule Initialization
BIRD89.1 Keyword Hierarchy Tree
BIRD90.2 Multiple A_to_D Subparameters Clarification
BIRD91.3 Multi-lingual Logic States Clarification

BIRD92.1 Multiple Terminator and Series Elements under [Model]
BIRD93.1 Model and Signal Name Limit Extension
BIRD94.2 Clarifications on [Diff Pin] Parameters
BIRD96 [Model Spec] and [Receiver Thresholds] Ordering
BIRD99.1 AMS Language Versions
BIRD100.2 Allow Pure Analog *-AMS Models
BIRD101 Section 6b, Figure 12 Example Note
BIRD102 File Name Limit Extension

Version 5.0. Version 5.0 adds more technical advances and some editorial changes documented in 10 BIRDs:

BIRD74.6 EMI Parameters
BIRD95.6 Power Integrity Using IBIS
BIRD98.3 Gate Modulation Effect (Table Format)
BIRD103.1 [Model Spec] DDR2 Overshoot/Undershoot Parameters
BIRD104.1 Algorithmic Modeling API (AMI) Support in IBIS
BIRD106 Clarification on Signal_pin Parameters
BIRD107.2 Update to Algorithmic Modeling API (AMI) Support in IBIS
BIRD108.1 Fixing Algorithmic Modeling API Impulse_matrix Nomenclature
BIRD109.1 S_overshoot_high/S_overshoot_low Clarification
BIRD110 Algorithmic Modeling Interface Section Title

Version 5.1. Version 5.1 uses a new document format and adds more technical advances and some editorial changes documented in 25 BIRDs:

BIRD111.3 Extended Usage of External Series Components in EBDs
BIRD112 IBIS-AMI clock_times Clarification
BIRD113.3 Weak Pull-up and Weak Pull-down Resistance and Voltage
BIRD114.3 IBIS-AMI Definition Clarifications
BIRD115 Clarifying Min/Typ/Max in IBIS-AMI
BIRD120.1 IBIS-AMI Flow Correction
BIRD126 IBIS-AMI New Reserved Parameter AMI_Version
BIRD127.4 IBIS-AMI Typographical Corrections
BIRD130 Crosstalk Clarification With Respect to AMI
BIRD132 Clarification of the Table Format for IBIS_AMI
BIRD133.1 Model Corner C_comp
BIRD134 AMI Function Return Value Clarification
BIRD135.1 Add Boolean to BNF for IBIS-AMI
BIRD136 Defining Relationships between Type and Format
BIRD137.2 AMI_parameters_in, AMI_parameters_out, msg Clarifications
BIRD138 IBIS-AMI Section 6c Tables Update
BIRD139.2 Reserved_Parameters Order
BIRD140.2 Format Corner and Range Clarification for IBIS-AMI
BIRD141 [Composite Current] Clarifications
BIRD142 Clarification of [Test Data] and [Test Load] scoping
BIRD143.1 Correcting the rules for AMI_Close

BIRD146 Clarify sample_interval for IBIS-AMI
BIRD148 Allowable Model_types with IBIS-AMI
BIRD149.1 Usage Out Syntax Correction
BIRD151 IBIS-AMI Modified Reserved Parameters for Jitter/Noise

Version 6.0. Version 6.0 adds more technical advances and some editorial changes documented in 7 BIRDs:

BIRD121.2 IBIS-AMI New Reserved Parameters for Data Management
BIRD123.5 IBIS-AMI New Reserved Parameters for Jitter/Noise
BIRD152 Analog Model Boundary Definition
BIRD154.1 Using IBIS-AMI Leaf List_Tip in List Parameters
BIRD156.3 IBIS-AMI Extension for Mid-channel Redrivers and Retimers
BIRD160.1 Analog Buffer Modeling Improvements
BIRD162.1 Change to Usage “Info, Out” for AMI Jitter and Noise Parameters

Version 6.1. Version 6.1 adds more technical advances and some editorial changes documented in 13 BIRDs:

BIRD155.2 New AMI API to Resolve Dependent Model Parameter
BIRD167.1 Table Corrections for Tx Jitter Parameters and Ignore_Bits
BIRD168.1 Handling of Overclocking Caused by Delay in Waveform Tables
BIRD169.1 DLL Dependency Checking
BIRD170 Delete Extra Paragraph for Ports under [External Circuit]
BIRD171.3 Clarify that Empty Root Name is Not Permitted in AMI Files
BIRD172.2 Extend Multilingual Parameter and Converter Parameter Rules
BIRD173.3 Package RLC Matrix Diagonals
BIRD174.1 Quote Character Clarifications
BIRD175.3 Extending IBIS-AMI for PAM4 Analysis
BIRD176 Power Pin Package Modeling
BIRD177 [Initial Delay] keyword for Submodels and Driver Schedules
BIRD178.3 Specifying Buffer Directionality for AMI

Version 7.0. Version 7.0 adds more technical advances and some editorial changes documented in 17 BIRDs:

BIRD147.6 Back-channel Support
BIRD165.1 Parameter Passing Improvements for [External Circuit]s
BIRD179 New IBIS-AMI Reserved Parameter Special_Param_Names
BIRD180 Require Unique Pin Names in [Pin]
BIRD182 POWER and GND [Pin] signal_name as [Pin Mapping] bus_label
BIRD183 [Model Data] Matrix Subparameter Terminology Correction
BIRD184.2 Model_name and Signal_name Restriction for POWER and GND Pins
BIRD185.2 Section 3 Reserved Word Guideline Update
BIRD186.4 File Naming Rules
BIRD187.3 Format and Usage Out Clarifications
BIRD188.1 Expanded Rx Noise Support for AMI
BIRD189.7 Interconnect Modeling Using IBIS-ISS and Touchstone

BIRD191.2 Clarifying Locations for Si_location and Timing_location
BIRD192.1 Clarification of List Default Rules
BIRD193 Figure 29 corrections
BIRD194 Revised AMI Ts4file Analog Buffer Models
BIRD196.1 Prohibit Periods at the Ends of File Names

Version 7.1. Version 7.1 adds more technical advances and some editorial changes documented in 15 BIRDs:

BIRD195.1 Enabling [Rgnd] and [Rpower] Keywords for Input Models
BIRD197.7 New AMI Reserved Parameter DC_Offset
BIRD198.3 Keyword Additions for On-Die PDN (Power Distribution Network) Modeling
BIRD199 Fix Rx_Receiver_Sensitivity Inconsistencies
BIRD200 C_comp Model Using IBIS-ISS or Touchstone
BIRD202.3 Electrical Descriptions of Modules
BIRD203 Submodel Clarification
BIRD205 New AMI Reserved Parameter for Sampling Position in AMI_Init Flow
BIRD206 Clarification of text “transition time”
BIRD207 New AMI Reserved Parameters Component_Name and Signal_Name
BIRD208 Clock-Data Pin Relationship Keyword
BIRD209 Make Clock Times Output Required for Clock Executable Models
BIRD212 Clarification of PAM4_UpperThreshold, PAM4_CenterThreshold, PAM4_LowerThreshold
BIRD214 Change “bit_time” to “symbol_time”
BIRD215 Back-channel Statistical Optimization Editorial Update

Version 7.2. Version 7.2 adds more technical advances, clarifications and some editorial changes documented in 8 BIRDs:

BIRD211.4 IBIS AMI Reference Flow Improvements
BIRD213.1 Extending IBIS-AMI for PAMn Analysis
BIRD216 Alphanumeric Pin Names
BIRD217 Require Clocked Rx Models to Return Clock Times
BIRD218 Designator Pin List Relaxation
BIRD219.1 AMI Parameter Root Name Clarifications
BIRD221 AMI_parameters_in Clarification
BIRD222 Clock Times Clarifications

Version 8.0. Version 8.0 adds more technical advances, clarifications and some editorial changes documented in 7 BIRDs:

BIRD224 New AMI Reserved Parameters for Ts4file port order
BIRD225 Clarification for bus_label rules
BIRD227 AMI Ignore Block Feature
BIRD228 Pin Name Field Extension
BIRD229.1 AMI Test Data Support
BIRD230.1 Adding a Definitions Section to IBIS
BIRD231 Clarifications on AMI Block Concepts

3 GENERAL SYNTAX RULES AND GUIDELINES

Unless noted otherwise, this section contains general syntax rules and guidelines for IBIS file formats .ibs (Sections 4, 5, 6 and 14), .pkg (Section 7), .ebd (Section 8), .ims (Section 11), .emd (Section 12), .ems (Section 13) and where applicable, .ami (Sections 10.3 through 10.12) and parameter passing files (Section 6.3).

3.1 FILE NAMING DEFINITIONS

The following terms and definitions relate to file naming and referencing for all file formats described in this specification.

- **file name**: The name of a file without its location.
- **stem**: The portion of a file name before the last period, or the full file name if no period.
- **extension**: The portion of a file name after the last period, if any.
- **directory**: A directory contains a list of files. Directories may include other directories, forming the basis for a hierarchical filesystem.
- **path**: A sequence of root directory (optional), directory elements and file name that identify the location of a file. A path may be absolute or relative.
- **absolute path**: A path that unambiguously identifies the location of a file without reference to an additional starting location.
- **relative path**: A path that is not absolute, and so only unambiguously identifies the location of a file when resolved relative to an implied starting location.
- **root name**: For operating systems supporting multiple filesystem roots, a name to identify the filesystem.
- **root directory**: A standard designation for the root of a filesystem.
- **file reference**: A reference to a file, expressed as either a simple file name or a relative path, which includes a simple file name.

The separator character between directory names in a directory path is the forward slash (“/”). The backward slash or backslash (“\”) character is not permitted. The EDA tool is responsible for making any operating system-specific adjustments (for example, replacing forward slashes “/” with backslashes “\”) if necessary.

Figure 1 shows an example of a file path with its parts delineated.

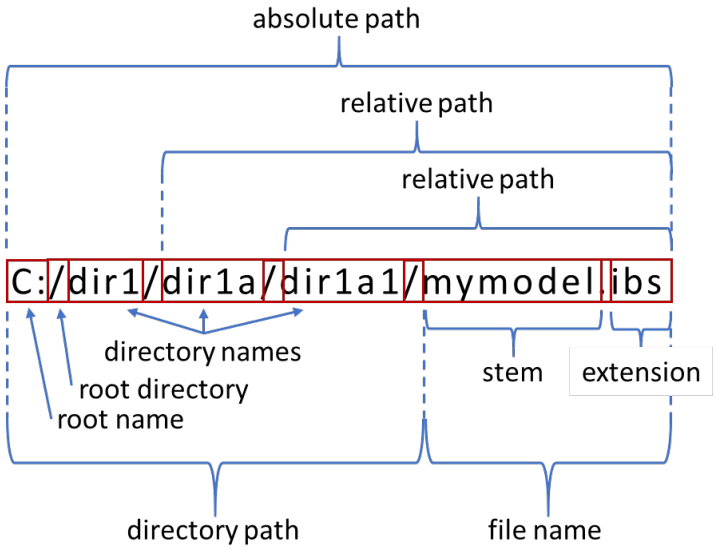


Figure 1 – Example of File Naming Definitions

3.2 SYNTAX RULES

1. The content of the files is case sensitive, except for reserved words and keywords.
2. The following words are reserved words for the purposes described below or where their usage is explicitly documented:

POWER	- reserved model name, used with power supply pins
GND	- reserved model name, used with ground pins
NC	- reserved model name, used with no-connect pins
NA	- used where data not available
CIRCUITCALL	- used for circuit call references in Section 6.3

   These words can be used elsewhere in a case-sensitive manner when they comply with other rules. For example, these words can be used as pin_names (except for CIRCUITCALL, which exceeds the maximum number of characters allowed under the first column of the [Pin] keyword) and signal_names under the [Pin] keyword (described later in Section 5).
3. Unless otherwise noted, file names shall have a stem of no more than sixty (60) characters and no less than one character followed by a period (“.”), followed by a file name extension. The file name (stem and extension) shall use characters from the set, except the extension shall not use the “.” character (space, “ ”, ASCII hexadecimal 0x20 is not included in the set):

   ```
   a b c d e f g h i j k l m n o p q r s t u v w x y z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   0 1 2 3 4 5 6 7 8 9 _ ^ $ ~ ! # % & - { } ) ( @ ' ` .
   ```

   Note that the apostrophe and grave accent characters shown above are ASCII hexadecimal 0x27 and 0x60, respectively.

   The character sequence “./” and sequences including it, such as “../” are not permitted in any reference to an IBIS file or to any other file format, effectively restricting the naming of files to

those in the same directory as the referring file or a subdirectory of that directory. Absolute paths - those beginning with a root name or root directory - are not permitted in a reference to any file.

4. Except for .ami files, a line of the file shall have at most 1024 characters, followed by a line termination sequence. The line termination sequence shall be one of the following two sequences: a linefeed character or a carriage return followed by a linefeed character.
5. Anything following the comment character is ignored and considered a comment on that line. The default "|" (pipe) character can be changed by the keyword [Comment Char] to any other character. The [Comment Char] keyword can be used anywhere in the file after the [IBIS Ver] keyword, as desired.
6. Keywords must be enclosed in square brackets, "[]", and must start in the first character of the line. No space or tab is allowed immediately after the opening bracket "[" or immediately before the closing bracket "]". If used, only one space (" ") or underscore ("_") character separates the parts of a multi-word keyword.
7. Underscores and spaces are equivalent in keywords. Spaces are not allowed in subparameter names. Some keywords use a table format with column headers appearing on the same line as the keyword name; these column headers might be referred to as 'subparameters' in this document.
8. Valid scaling factors are:

T = tera	k = kilo	n = nano
G = giga	m = milli	p = pico
M = mega	u = micro	f = femto

   When no scaling factor is specified, the appropriate base units are assumed (these are volts, amperes, ohms, farads, henries, and seconds). The parser looks at only one alphabetic character after a numerical entry. Therefore, it is enough to use only the prefixes to scale the parameters. However, for clarity, using full abbreviations for the units (e.g., pF, nH, mA) is allowed. In addition, scientific notation is allowed (e.g., 1.2345e-12).
9. The I-V data tables should use enough data points around sharply curved areas of the I-V curves to describe the curvature accurately. In linear regions there is no need to define unnecessary data points.
10. The use of tab characters is legal, but should be avoided as much as possible. This is to eliminate possible complications that might arise in situations when tab characters are automatically converted to multiple spaces by text editing, file transferring and similar software. In cases like that, lines might become longer than 1024 characters, which is illegal in IBIS file formats (except for .ami files).
11. Currents are considered positive when their direction is into the component.
12. All temperatures are represented in degrees Celsius.
13. Important supplemental information is contained in Section 9, "NOTES ON DATA DERIVATION METHOD", concerning how data values are derived.
14. Only ASCII characters, as defined in ANSI Standard X3.4-1986, shall be used in IBIS file formats. The use of characters with codes greater than hexadecimal 0x7E is not allowed. Also, ASCII control characters (those numerically less than hexadecimal 0x20) are not allowed,

except for tabs or in a line termination sequence. As mentioned above, the use of tab characters is discouraged.

15. A “whitespace character” in this document refers to the space (“ ”) or tab characters. “Whitespace” may be one of these characters or a combination of any number of both.
16. The term “alphanumeric” in this document refers to words composed of the following characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
```

3.3 GENERAL TERMINOLOGY DEFINITIONS

This section contains definitions of terms used throughout this document.

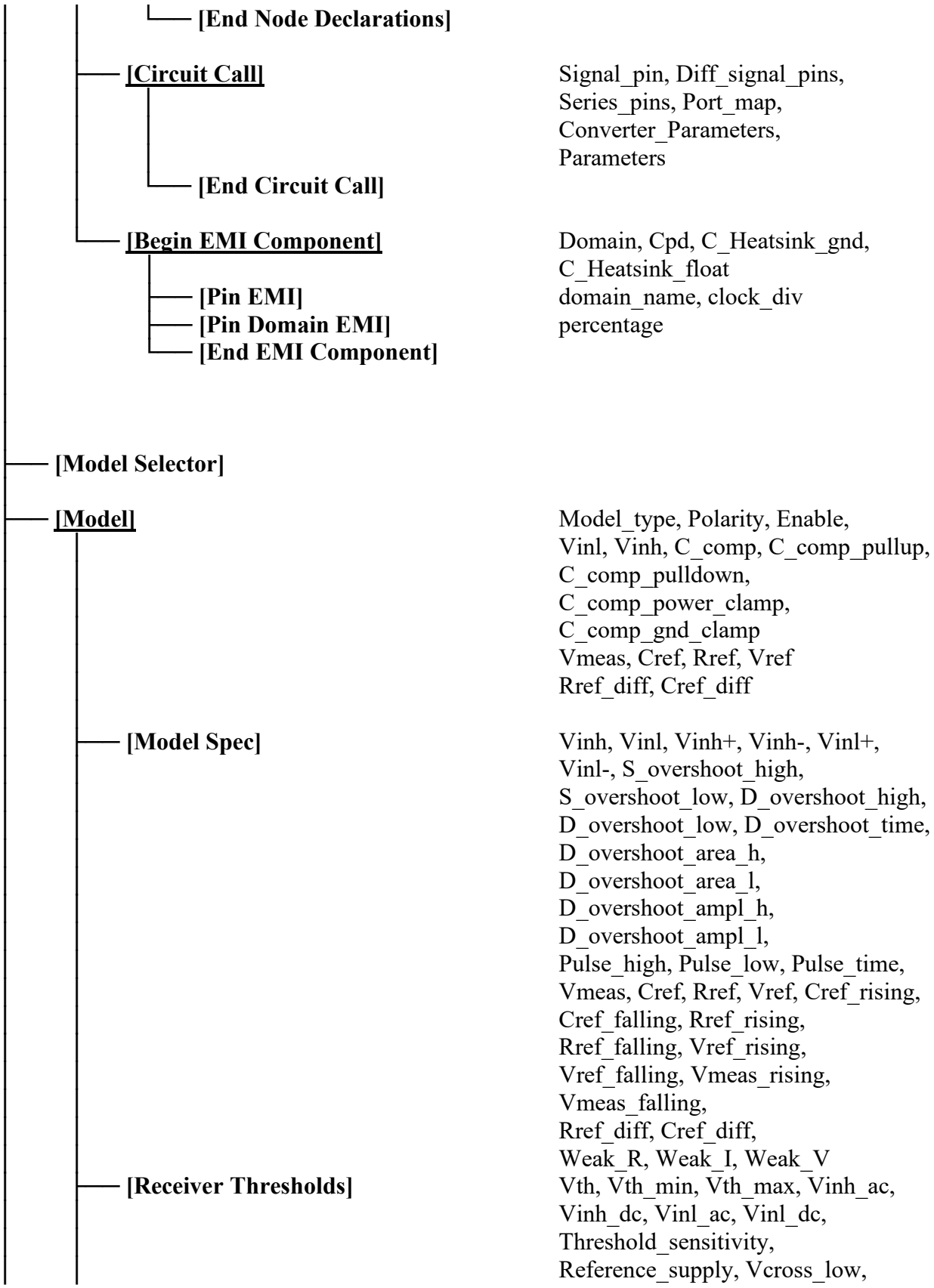
- **Block**: In the AMI context, a group of waveform samples, denominated as a number of samples, a number of symbols, or a number of bits (in the case of NRZ signaling). See also "segment" below. For example, at a sampling rate of 32 samples per UI, a block of 512 samples would represent 16 symbols. As a corollary to this, a block of 16 symbols would consist of 512 samples.
- **NRZ**: Non-Return-to-Zero signaling, where two distinct voltage levels define the two logic states possible in a symbol.
- **PAM**: Pulse-Amplitude Modulation, a signaling designation where the number of voltage levels and therefore the number of logic states possible in a symbol is explicitly defined. The term is followed by a non-zero integer to indicate the number of levels (e.g., in PAM4, four levels are possible within a symbol; PAM2 is equivalent to NRZ). For the general case, the abbreviation "PAMn" is used.
- **Segment**: A group of sequential waveform samples. In the AMI context, this is used in reference to the wave_size parameter passed as part of AMI function calls. Also called "time segment".
- **Symbol**: A unit of data transmitted in a given time interval, consisting of a single bit or multiple bits depending on the signaling used (in NRZ signaling, two values are possible: 0 or 1; in PAM4 signaling, four values are possible, each containing two bits: 00, 01, 10, and 11).
- **Time segment**: see "segment".
- **UI**: Unit Interval, the time duration in which a symbol is transmitted.

3.4 KEYWORD HIERARCHY

ibs FILE

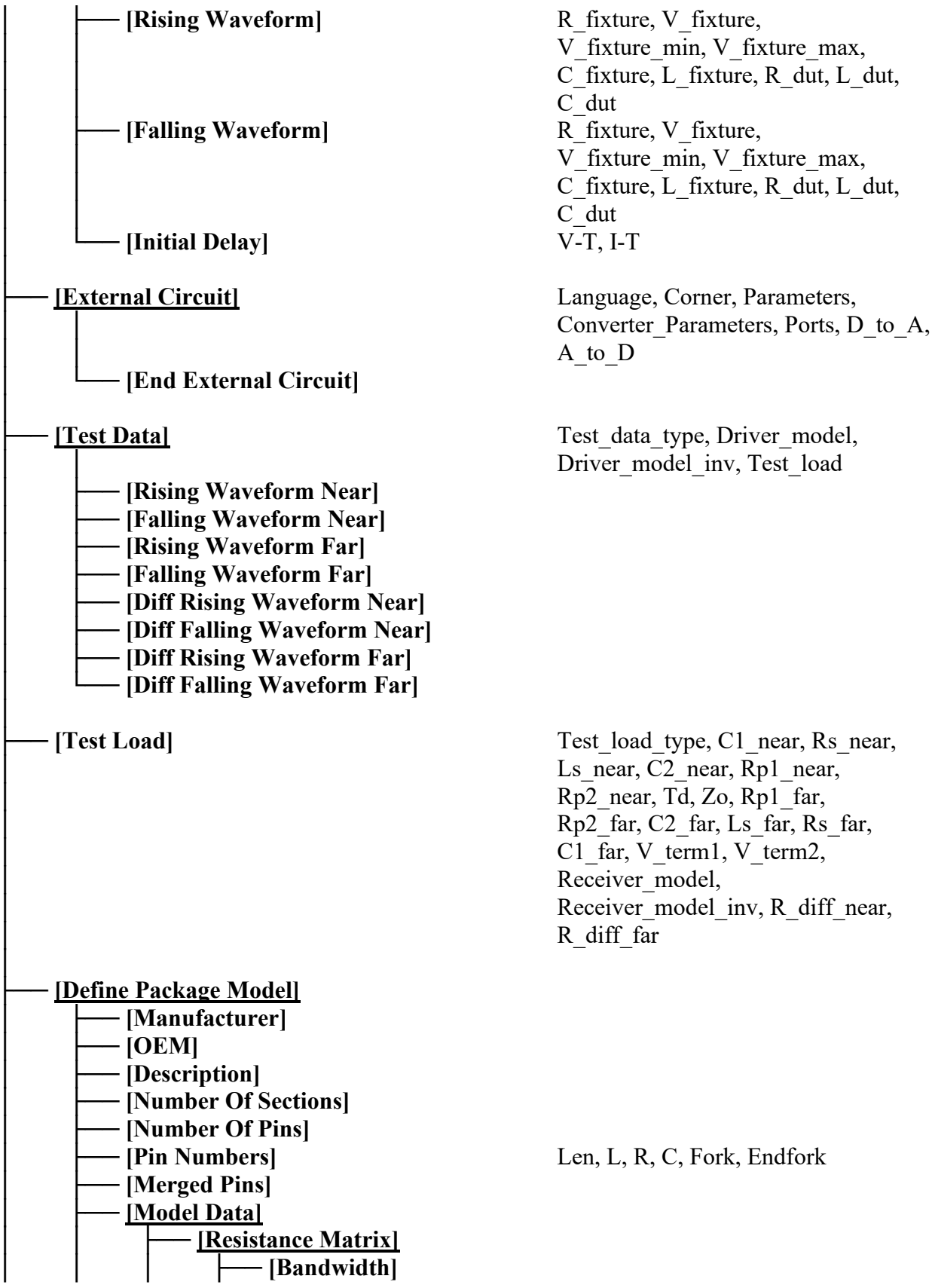
- File Header Section
  - **[IBIS Ver]**
  - **[Comment Char]**
  - **[File Name]**
  - **[File Rev]**
  - **[Date]**
  - **[Source]**
  - **[Notes]**
  - **[Disclaimer]**
  - **[Copyright]**

[Component]	Si_location, Timing_location
[Manufacturer]	
[Package]	R_pkg, L_pkg, C_pkg
[Pin]	signal_name, model_name, R_pin, L_pin, C_pin
[Clock Pins]	clocked_pins, relationship
[Package Model]	
[Alternate Package Models]	
[End Alternate Package Models]	
[Interconnect Model Group]	
[End Interconnect Model Group]	
[PDN Domain]	Bus_label, Signal_name
[PDN Model]	R_pdn, C_pdn, R_leak
[End PDN Model]	
[End PDN Domain]	
[Pin Mapping]	pulldown_ref, pullup_ref, gnd_clamp_ref, power_clamp_ref, ext_ref
[Bus Label]	
[Die Supply Pads]	
[Diff Pin]	inv_pin, vdiff, tdelay_typ, tdelay_min, tdelay_max
[Repeater Pin]	tx_non_inv_pin
[Series Pin Mapping]	pin_2, model_name, function_table_group
[Series Switch Groups]	On, Off
[Node Declarations]	


[End Node Declarations]
[Circuit Call]
Signal_pin, Diff_signal_pins, Series_pins, Port_map, Converter_Parameters, Parameters
[End Circuit Call]
[Begin EMI Component]
Domain, Cpd, C_Heatsink_gnd, C_Heatsink_float
[Pin EMI]
domain_name, clock_div
[Pin Domain EMI]
percentage
[End EMI Component]
[Model Selector]
[Model]
Model_type, Polarity, Enable, Vinl, Vinh, C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, C_comp_gnd_clamp Vmeas, Cref, Rref, Vref Rref_diff, Cref_diff
[Model Spec]
Vinh, Vinl, Vinh+, Vinh-, Vinl+, Vinl-, S_overshoot_high, S_overshoot_low, D_overshoot_high, D_overshoot_low, D_overshoot_time, D_overshoot_area_h, D_overshoot_area_l, D_overshoot_ampl_h, D_overshoot_ampl_l, Pulse_high, Pulse_low, Pulse_time, Vmeas, Cref, Rref, Vref, Cref_rising, Cref_falling, Rref_rising, Rref_falling, Vref_rising, Vref_falling, Vmeas_rising, Vmeas_falling, Rref_diff, Cref_diff, Weak_R, Weak_I, Weak_V
[Receiver Thresholds]
Vth, Vth_min, Vth_max, Vinh_ac, Vinh_dc, Vinl_ac, Vinl_dc, Threshold_sensitivity, Reference_supply, Vcross_low,

	Vcross_high, Vdiff_ac, Vdiff_dc, Tslew_ac, Tdiffslew_ac
[Add Submodel]	
[Driver Schedule]	
[Temperature Range]	
[Voltage Range]	
[Pullup Reference]	
[Pulldown Reference]	
[POWER Clamp Reference]	
[GND Clamp Reference]	
[External Reference]	
[C Comp Corner]	C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, C_comp_gnd_clamp
[C Comp Model]	C_comp_model_mode, Param, File_IBIS-ISS, File_TS, Number_of_terminals
[End C Comp Model]	
[TTgnd]	
[TTpower]	
[Pulldown]	
[Pullup]	
[GND Clamp]	
[POWER Clamp]	
[ISSO PU]	
[ISSO PD]	
[Rgnd]	
[Rpower]	
[Rac]	
[Cac]	
[On]	
[Off]	
[R Series]	
[L Series]	
[Rl Series]	
[C Series]	
[Lc Series]	
[Rc Series]	
[Series Current]	
[Series MOSFET]	Vds
[Ramp]	dV/dt_r, dV/dt_f, R_load
[Rising Waveform]	R_fixture, V_fixture, V_fixture_min, V_fixture_max,

	C_fixture, L_fixture, R_dut, L_dut, C_dut
[Composite Current]	
[Falling Waveform]	R_fixture, V_fixture, V_fixture_min, V_fixture_max, C_fixture, L_fixture, R_dut, L_dut, C_dut
[Composite Current]	
[Initial Delay]	V-T, I-T
[External Model]	Language, Corner, Parameters, Converter_Parameters, Ports, D_to_A, A_to_D
[End External Model]	
[Algorithmic Model]	Executable, Executable_Rx, Executable_Tx
[AMI Test Configuration]	Type, Direction, AMI_input_parameters_file, Input_IR_file, Input_waveform_file, Golden_IR_file, Golden_waveform_file, Clock_input_file, Clock_output_file, AMI_output_parameters_file, Executable_index
[End Algorithmic Model]	
[Begin EMI Model]	Model_emi_type, Model_Domain
[End EMI Model]	
[Submodel]	Submodel_type
[Submodel Spec]	V_trigger_r, V_trigger_f, Off_delay
[POWER Pulse Table]	
[GND Pulse Table]	
[Pulldown]	
[Pullup]	
[GND Clamp]	
[POWER Clamp]	
[Ramp]	dV/dt_r, dV/dt_f, R_load


[Rising Waveform]
R_fixture, V_fixture, V_fixture_min, V_fixture_max, C_fixture, L_fixture, R_dut, L_dut, C_dut
[Falling Waveform]
R_fixture, V_fixture, V_fixture_min, V_fixture_max, C_fixture, L_fixture, R_dut, L_dut, C_dut
[Initial Delay]
V-T, I-T
[External Circuit]
Language, Corner, Parameters, Converter_Parameters, Ports, D_to_A, A_to_D
[End External Circuit]
[Test Data]
Test_data_type, Driver_model, Driver_model_inv, Test_load
[Rising Waveform Near]
[Falling Waveform Near]
[Rising Waveform Far]
[Falling Waveform Far]
[Diff Rising Waveform Near]
[Diff Falling Waveform Near]
[Diff Rising Waveform Far]
[Diff Falling Waveform Far]
[Test Load]
Test_load_type, C1_near, Rs_near, Ls_near, C2_near, Rp1_near, Rp2_near, Td, Zo, Rp1_far, Rp2_far, C2_far, Ls_far, Rs_far, C1_far, V_term1, V_term2, Receiver_model, Receiver_model_inv, R_diff_near, R_diff_far
[Define Package Model]
[Manufacturer]
[OEM]
[Description]
[Number Of Sections]
[Number Of Pins]
[Pin Numbers]
Len, L, R, C, Fork, Endfork
[Merged Pins]
[Model Data]
[Resistance Matrix]
[Bandwidth]

- **[Row]**
- **[Inductance Matrix]**
  - **[Bandwidth]**
  - **[Row]**
- **[Capacitance Matrix]**
  - **[Bandwidth]**
  - **[Row]**
- **[End Model Data]**
- **[End Package Model]**
- **[Interconnect Model Set]**
  - **[Manufacturer]**
  - **[Description]**
  - **[Interconnect Model]** — Param, File_TS, File_IBIS-ISS, Unused_port_termination, Number_of_terminals
    - **[End Interconnect Model]**
  - **[End Interconnect Model Set]**
- **[End]**

.pkg FILE

- File Header Section
  - **[IBIS Ver]**
  - **[Comment Char]**
  - **[File Name]**
  - **[File Rev]**
  - **[Date]**
  - **[Source]**
  - **[Notes]**
  - **[Disclaimer]**
  - **[Copyright]**
- **[Define Package Model]**
  - **[Manufacturer]**
  - **[OEM]**
  - **[Description]**
  - **[Number Of Sections]**
  - **[Number Of Pins]**
  - **[Pin Numbers]** — Len, L, R, C, Fork, Endfork

- **[Merged Pins]**
- **[Model Data]**
  - **[Resistance Matrix]**
    - **[Bandwidth]**
    - **[Row]**
  - **[Inductance Matrix]**
    - **[Bandwidth]**
    - **[Row]**
  - **[Capacitance Matrix]**
    - **[Bandwidth]**
    - **[Row]**
  - **[End Model Data]**
- **[End Package Model]**

[End]

.ebd FILE

- File Header Section
  - **[IBIS Ver]**
  - **[Comment Char]**
  - **[File Name]**
  - **[File Rev]**
  - **[Date]**
  - **[Source]**
  - **[Notes]**
  - **[Disclaimer]**
  - **[Copyright]**
- **[Begin Board Description]**
  - **[Manufacturer]**
  - **[Number of Pins]**
  - **[Pin List]** signal_name
  - **[Path Description]** Len, L, R, C, Fork, Endfork, Pin, Node
  - **[Reference Designator Map]**
  - **[End Board Description]**

[End]

.ims FILE

- File Header Section
  - **[IBIS Ver]**
  - **[Comment Char]**
  - **[File Name]**
  - **[File Rev]**
  - **[Date]**
  - **[Source]**
  - **[Notes]**
  - **[Disclaimer]**
  - **[Copyright]**
- **[Interconnect Model Set]**
  - **[Manufacturer]**
  - **[Description]**
  - **[Interconnect Model]** — Param, File_TS, File_IBIS-ISS, Unused_port_termination, Number_of_terminals
    - **[End Interconnect Model]**
  - **[End Interconnect Model Set]**
- **[End]**

.emd FILE

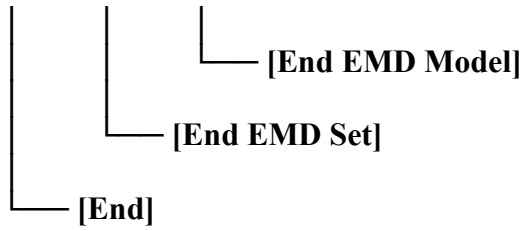
- File Header Section
  - **[IBIS Ver]**
  - **[Comment Char]**
  - **[File Name]**
  - **[File Rev]**
  - **[Date]**
  - **[Source]**
  - **[Notes]**
  - **[Disclaimer]**
  - **[Copyright]**
- **[Begin EMD]**
  - **[Manufacturer]**
  - **[Description]**
  - **[Number of EMD Pins]**
  - **[EMD Pin List]** — signal_name, signal_type, bus_label
    - **[End EMD Pin List]**
  - **[EMD Parts]**
    - **[End EMD Parts]**

- **[EMD Designator List]**
  - **[End EMD Designator List]**
- **[Designator Pin List]** — signal_name, signal_type bus_label
  - **[End Designator Pin List]**
- **[Voltage List]**
  - **[End Voltage List]**
- **[EMD Group]**
  - **[End EMD Group]**
- **[End EMD]**

- **[EMD Set]**
  - **[Manufacturer]**
  - **[Description]**
  - **[EMD Model]** — Param, File_TS, File_IBIS-ISS, Unused_port_termination, Number_of_terminals
    - **[End EMD Model]**
  - **[End EMD Set]**
- **[End]**

.ems FILE

- File Header Section
  - **[IBIS Ver]**
  - **[Comment Char]**
  - **[File Name]**
  - **[File Rev]**
  - **[Date]**
  - **[Source]**
  - **[Notes]**
  - **[Disclaimer]**
  - **[Copyright]**
- **[EMD Set]**
  - **[Manufacturer]**
  - **[Description]**
  - **[EMD Model]** — Param, File_TS, File_IBIS-ISS, Unused_port_termination,



Number_of_terminals

4 FILE HEADER AND FILE END INFORMATION

Keyword: **[IBIS Ver]**

Required: Yes

Description: Specifies the version of the .ibs, .pkg, .ims, or .ebd file. This keyword informs electronic parsers of the kinds of data types that are present in the file.

Usage Rules: [IBIS Ver] must be the first keyword in any .ibs file. It is normally on the first line of the file, but can be preceded by comment lines that must begin with a "|". The value shall be the number of an approved IBIS version: 1.1… 8.0.

Example:

```
[IBIS Ver]  8.0   | The version of the syntax used
```

Keyword: **[Comment Char]**

Required: No

Description: Defines a new comment character to replace the default "|" (pipe) character, if desired, for the .ibs, .pkg, .ims, or .ebd file.

Usage Rules: The new comment character to be defined must be followed by the underscore character and the letters "char". For example: "|_char" redundantly redefines the comment character to be the pipe character. The new comment character is in effect only following the [Comment Char] keyword. The following characters MAY be used:

```
! " # $ % & ' ( ) * , : ; < > ? @ \ ^ ` { | } ~
```

Other Notes: The [Comment Char] keyword can be used anywhere in the file after the [IBIS Ver] keyword, as desired. The keyword may appear multiple times in the file.

Example:

```
[Comment Char]  |_char
```

Keyword: **[File Name]**

Required: Yes

Description: Specifies the file name of the file containing this keyword.

Usage Rules: The file name shall conform to the rules in item 3 of Section 3.2, "SYNTAX RULES". In addition, the file name shall use the extension "ibs", "pkg", "ebd", "ims", "emd", or "ems". The file name shall be the actual name of the file.

Example:

```
[File Name]     ver7_2.ibs
```

Keyword: **[File Rev]**

Required: Yes

Description: Tracks the revision level of a particular .ibs, .pkg, .ebd, .ims, .emd, or .ems file.

Usage Rules: Revision level is set at the discretion of the engineer defining the file. The following guidelines are recommended:

0.x silicon and file in development
1.x pre-silicon file data from silicon model only
2.x file correlated to actual silicon measurements
3.x mature product, no more changes likely

Example:

```
[File Rev]      1.0                     | Used for IBIS file variations
```

Keywords: **[Date]**, **[Source]**, **[Notes]**, **[Disclaimer]**, **[Copyright]**

Required: No

Description: Optionally provides additional information about the .ibs, .pkg, .ebd, or .ims file.

Usage Rules: The keyword arguments can contain blanks, and be of any format. The [Date] keyword argument is limited to a maximum of 40 characters, and the month should be spelled out for clarity.

Because IBIS model writers may consider the information in these keywords essential to users, and sometimes legally required, design automation tools should make this information available. Derivative models should include this text verbatim. Any text following the [Copyright] keyword must be included, verbatim, in any derivative models.

Examples:

```
[Date]          December 5, 2025          | The latest file revision date
|
[Source]        Put originator and the source of information here.  For
                example:
                From silicon level SPICE model at NoName.
                From lab measurement.
                Compiled from manufacturer's data book, etc.
|
[Notes]         Use this section for any special notes related to the file.
|
[Disclaimer]    This information is for modeling purposes only, and is not
                guaranteed.                     | May vary by component
|
[Copyright]     Copyright 2025, XYZ Corp., All Rights Reserved
```

Keyword: **[End]**
Required: Yes
Description: Defines the end of the .ibs, .pkg, .ims, or .ebd file.
Example:

```
[End]
```

5 COMPONENT DESCRIPTION

Keyword: **[Component]**

Required: Yes

Description: Marks the beginning of the IBIS description of the integrated circuit named after the keyword.

Sub-Params: Si_location, Timing_location

Usage Rules: If the .ibs file contains data for more than one component, each section must begin with a new [Component] keyword. The length of the component name must not exceed 40 characters, and blank characters are allowed.

NOTE: Blank characters are not recommended due to usability issues.

Si_location and Timing_location are optional and specify where the Signal Integrity and Timing measurements are made for the component. Allowed values for either subparameter are "Die" or "Pin". For pins that connect to a buffer through an [Interconnect Model Set] keyword, described below, the "Die" selection shall be at the buffer terminal location. The default location is at the "Pin". The "Die" location refers to the Buffer_I terminal of a [C Comp Model], if [C Comp Model] is present and includes the Buffer_I terminal.

Example:

```
[Component]     7403398 MC452
|
Si_location     Pin    | Optional subparameters to give measurement
Timing_location Die    | location positions
```

Keyword: **[Manufacturer]**

Required: Yes

Description: Specifies the name of the component's manufacturer.

Usage Rules: The length of the manufacturer's name must not exceed 40 characters (blank characters are allowed, e.g., Texas Instruments). In addition, each manufacturer must use a consistent name in all .ibs files.

Example:

```
[Manufacturer]  NoName Corp.
```

Keyword: **[Package]**

Required: Yes

Description: Defines a range of values for the default packaging resistance, inductance, and capacitance of the component pins, organized by corner.

Sub-Params: R_pkg, L_pkg, C_pkg

Usage Rules: The typical (typ) column must be specified. If data for the other columns are not available, they must be noted with "NA".

Other Notes: If RLC parameters are available for individual pins, they can be listed in columns 4-6 under keyword [Pin]. The values listed in the [Pin] description section override the default values defined here. Use the [Package Model] or [Interconnect Model Group] keyword for more complex package descriptions.

If defined, the [Package Model] or [Interconnect Model Group] data overrides the values in the [Package] keyword. Regardless, the data listed under the [Package] keyword must still contain valid data.

Example:

```
[Package]
| variable      typ             min             max
R_pkg           250.0m          225.0m          275.0m
L_pkg           15.0nH          12.0nH          18.0nH
C_pkg           18.0pF          15.0pF          20.0pF
```

Keyword: **[Pin]**

Required: Yes

Description: Associates the component's I/O models to its various external pin names and signal names.

Sub-Params: signal_name, model_name, R_pin, L_pin, C_pin

Usage Rules: For a full component description, all pins on a component should be specified. The first column shall contain the alphanumeric pin name, which shall not be repeated within the same [Pin] keyword for a [Component] (the entries in the first column are also referred to as pin_names elsewhere in this document). The second column, signal_name, gives the data book name for the signal on that pin. The third column, model_name, maps a pin to a specific I/O buffer model or model selector name. Each model_name shall have a corresponding model or model selector name listed in a [Model] or [Model Selector] keyword below, unless it is a reserved model name (POWER, GND, CIRCUITCALL, or NC).

If a pin has a model_name POWER, then all other pins with the same signal_name as this pin shall have model_name POWER. If a pin has model_name GND, then all other pins with the same signal_name as this pin shall have model_name GND.

The model_name column cannot be used for model or model selector names that reference Series and Series_switch models.

Each line must contain either three or six columns. A pin line with three columns only associates the pin's signal and model. Six columns can be used to override the default package values (specified under [Package]) *for that pin only*. When using six columns, the headers R_pin, L_pin, and C_pin must be listed. If "NA" is in columns 4 through 6, the default packaging values must be used. The headers R_pin, L_pin, and C_pin may be listed in any order.

Column length limits are:

[Pin]	20 characters max
model_name	40 characters max
signal_name	40 characters max
R_pin	9 characters max

L_pin 9 characters max
C_pin 9 characters max

Example:

```
[Pin]   signal_name     model_name      R_pin   L_pin   C_pin
|
  1     RAS0#           Buffer1         200.0m  5.0nH   2.0pF
  2     RAS1#           Buffer2         209.0m  NA      2.5pF
  3     EN1#            Input1          NA      6.3nH   NA
  4     A0              3-state
  5     D0              I/O1
  6     RD#             Input2          310.0m  3.0nH   2.0pF
  7     WR#             Input2
  8     A1              I/O2
  9     D1              I/O2
 10     GND             GND             297.0m  6.7nH   3.4pF
 11     RDY#            Input2
 12     GND             GND             270.0m  5.3nH   4.0pF
|  .
|  .
|  .
 18     Vcc3            POWER
 19     NC              NC
 20     Vcc5            POWER           226.0m  NA      1.0pF
 21     BAD1            Series_switch1     |   Illegal assignment
 22     BAD2            Series_selector1   |   Illegal assignment
```

Keyword: **[Package Model]**

Required: No

Description: Indicates the name of the package model to be used for the component.

Usage Rules: The package model name is limited to 40 characters. Spaces are allowed in the name. The name should include the company name or initials to help ensure uniqueness. The EDA tool will search for a matching package model name as an argument to a [Define Package Model] keyword in the current .ibs file first. If a match is not found, the EDA tool will next look for a match in an external .pkg file. If the matching package model is in an external .pkg file, it must be located in the same directory as the .ibs file. The file names of .pkg files must follow the rules for file names given in Section 3.2, “SYNTAX RULES”.

Other Notes: Use the [Package Model] keyword within a [Component] to indicate which package model should be used for that component. The specification permits .ibs files to contain [Define Package Model] keywords as well. These are described under “Package Modeling” in Section 7. When package model definitions occur within a .ibs file, their scope is “local”, i.e., they are known only within that .ibs file and no other. In addition, within that .ibs file, they override any externally defined package models that have the same name.

Example:

```
[Package Model]     QS-SMT-cer-8-pin-pkgs
```

Keywords: **[Alternate Package Models]**, **[End Alternate Package Models]**

Required: No

Description: Used to select a package model from a list of package models.

Usage Rules: The [Alternate Package Models] keyword can be used in addition to the [Package Model] keyword. [Alternate Package Models] shall be used only for components that use the [Package Model] keyword.

Each [Alternate Package Models] keyword specifies a set of alternate package model names for only one component, which is given by the previous [Component] keyword. The [Alternate Package Models] keyword shall not appear before the first [Component] keyword in a .ibs file. The [Alternate Package Models] keyword applies only to the [Component] section in which it appears, and must be followed by an [End Alternate Package Models] keyword.

All alternate package model names must appear below the [Alternate Package Models] keyword, and above the following [End Alternate Package Models] keyword. The package model names listed under the [Alternate Package Models] must follow the rules of the package model names associated with the [Package Model] keyword. The package model names correspond to the names of package models defined by [Define Package Model] keywords. EDA tools may offer users a facility for choosing between the default package model and any of the alternate package models, when analyzing occurrences of the [Component].

The package model named by [Package Model] can be optionally repeated in the [Alternate Package Models] list of names.

Example:

```
[Alternate Package Models]
|
208-pin_plastic_PQFP_package-even_mode | Descriptive names are shown
208-pin_plastic_PQFP_package-odd_mode
208-pin_ceramic_PQFP_package-even_mode
208-pin_ceramic_PQFP_package-odd_mode
|
[End Alternate Package Models]
```

Keyword: **[Interconnect Model Group]**

Required: No

Description: [Interconnect Model Group] has a single argument, which is the name of the associated Interconnect Model Group. The length of the Interconnect Model Group name shall not exceed 40 characters. Blank characters are not allowed. The [Interconnect Model Group]/[End Interconnect Model Group] keyword pair is hierarchically scoped by the [Component] keyword. The [Interconnect Model Group] keyword is used to define a list of [Interconnect Model Set]s by name that shall be used together to define Interconnect Models to be used in a simulation. A simulation may contain Interconnect Models from the Interconnect Model Sets listed in only one Group.

Usage Rules: [Component] may contain zero or more [Interconnect Model Group] keywords (identified by a name). Each [Interconnect Model Group] must contain at least one [Interconnect Model Set] name. Interconnect Model Sets contain Interconnect Models used to describe pin, die

pad or buffer terminal connections to IBIS-ISS subcircuits or n-port networks described by Touchstone files.

If a component's interconnect is modeled using one or more Interconnect Model Sets, those Interconnect Model Sets should be listed in one or more Interconnect Model Groups. An Interconnect Model Group is required even if it references only one Interconnect Model Set. If there are no Interconnect Model Sets, the [Interconnect Model Group] keyword is illegal.

The section under the [Interconnect Model Group] keyword shall have two entries per line, with each line identifying one Interconnect Model Set associated with the component. The entries shall be separated by at least one whitespace character. The first entry lists the Interconnect Model Set name (up to 40 characters long). The second entry is the file reference of the file containing the Interconnect Model Set and shall have the extension "ims". This file reference shall conform to the rules given in Section 3.2, "SYNTAX RULES". If the Interconnect Model Set is in the same .ibs file as [Component], then the second entry shall be "NA".

The files containing the Interconnect Model Sets with the "ims" extension shall be located in the same directory as the .ibs file or in a specified directory under the .ibs file as determined by the directory path according to the file name rules given in Section 3.2, "SYNTAX RULES" (i.e., a file reference containing a relative path to a directory below that of the referencing .ibs file is permitted). An [Interconnect Model Set] with matching name shall be found in the stated location for each Interconnect Model Set named in the [Interconnect Model Group].

Each Interconnect Model Set name and its file_reference may only appear once under each [Interconnect Model Group] keyword for a given component.

As discussed in Section 11, three interface locations exist: pin, die pad, and buffer. These interfaces are identified in the terminal lines under the [Interconnect Model] keyword and by their Terminal_type column entries (shown in Table 49) as follows:

pin:	Pin_I/O, Pin_Rail, A_gnd
die pad:	Pad_I/O, Pad_Rail, A_gnd
buffer:	Buffer_I/O, Buffer_Rail, Pullup_ref, Pulldown_ref, Power_clamp_ref, Gnd_clamp_ref, Ext_ref, A_gnd

A_gnd is the simulator global reference node of the Interconnect Model.

Identifiers associated with these Terminal_type *_I/Os are pin_name entries. In addition, some *_I/O terminals may have the optional Aggressor_Only column. If any *_I/O pin is marked as Aggressor_Only, then any *_I/O pin with the same pin_name entry shall be considered as an aggressor, and not a victim. Any *_I/O Terminal_type without the Aggressor_Only column may be considered as an aggressor or a victim.

The remaining terminals are used for POWER or GND and are referred to as "rails". The rail identifiers are pin_name, signal_name, bus_label (described below) and pad_name entries (described below) according to the allowable association rules summarized in Section 11.2.1 (Connecting Pins, Pads and Buffer Terminals) and Table 49.

An Interconnect Model Group contains of a list of Interconnect Model Sets, which in turn contains a list of Interconnect Models. There are several rules that apply to this combined list of Interconnect Models in an Interconnect Model Group:

- I/O pin_name rules

- I/O terminals use pin_name identifiers
- All *_I/O pin_names may omit the Aggressor_Only column (may be aggressors or victims)
- No I/O pin_name in a component may appear as a Pin_I/O terminal without the Aggressor_Only column in more than one Interconnect Model in the Interconnect Model Group.
- No I/O pin_name in a component may appear as a Buffer_I/O terminal without the Aggressor_Only column in more than one Interconnect Model in the Interconnect Model Group.
- An I/O pin_name may appear in Interconnect Models with the following interface combinations:
  - pin to buffer
  - pin to die pad (in one Interconnect Model) and die pad to buffer (in another Interconnect Model)
  - pin to die pad
  - die pad to buffer
- A *_I/O pin_name may not appear in Interconnect Models of Interconnect Model Sets that are listed in one Interconnect Model Group with the following interface combinations:
  - pin to buffer (in one Interconnect Model) and pin to die pad (in another Interconnect Model)
  - pin to buffer (in one Interconnect Model) and die pad to buffer (in another Interconnect Model)
  - pin to buffer and pin to die pad and die pad to buffer in three separate Interconnect Models

- General description of rail terminals
  - At the pin interface, a terminal whose Terminal_type is Pin_Rail can be identified by a pin_name, signal_name or bus_label entry. A pin_name maps directly into a Pin_Rail pin_name entry or the pin_name can be mapped into a bus_label or a signal_name with the information given in the [Pin] keyword or by the [Pin Mapping], [Bus Label], or [Die Supply Pads] keywords described later in this section.
    - Note that a terminal whose Terminal_type is Pin_Rail may be associated with one pin_name or a list of pin_names on a rail that is associated with a signal_name or bus_label. If the terminal is associated with more than one pin_name then these pins are shorted together.
  - At a die pad interface, a terminal whose Terminal_type is Pad_Rail can be identified by a pad_name, signal_name or bus_label entry. Connections between die pad interfaces in different Interconnect Models can be made by using identical pad_names or by identifying a common bus_label or signal_name that is available in the [Pin], [Pin Mapping], [Die Supply Pads], or [Bus Label] keywords.

        - Note that a terminal whose Terminal_type is Pad_Rail may be associated with one pad_name or a list of pad_names on a rail that is associated with a single signal_name or bus_label. If the terminal is associated with more than one pad_name then these pads are shorted together.
    - At the buffer interface, a terminal whose Terminal_type is Pullup_ref, Pulldown_ref, Power_clamp_ref, Gnd_clamp_ref, or Ext_ref may be identified by a Buffer_I/O pin_name. Terminals of Terminal_type Buffer_Rail may be identified by a signal_name or bus_label entry.
        - Note that a terminal whose Terminal_type is Pullup_ref, Pulldown_ref, Power_clamp_ref, Gnd_clamp_ref, Ext_ref or Buffer_Rail may be associated with one buffer terminal or a list of buffer terminals on a rail that is associated with a single signal_name or bus_label. If it is associated with more than one buffer terminal, then these buffer terminals are shorted together.
    - A Power Delivery Network (PDN) has one or more connections of rail terminals between Pin and Buffer, Pin and Pad or Pad and Buffer.
    - An Interconnect Model with only rail terminals and two interfaces (no I/O terminals) can be used for a PDN.
    - An Interconnect Model with only rail terminals (no I/O terminals) and only one interface is permitted for applications such as for modeling rail decoupling circuits.
    - A PDN structure can also exist in an Interconnect Model with I/O terminals.
    - Rail terminals or A_gnd can be used in Interconnect Models to provide a reference node for the electrical interconnections associated with *_I/O terminals.
- Rail terminal rules
    - At the pin interface, a rail pin_name may appear on a terminal line whose Terminal_type is Pin_Rail in multiple Interconnect Models in the Interconnect Model Group.
    - At the buffer interface, a rail pin_name may appear on a terminal line whose Terminal_type is Pullup_ref, Pulldown_ref, Power_clamp_ref, Gnd_clamp_ref, Ext_ref or as a Buffer_Rail in more than one power delivery Interconnect Model in the Interconnect Model Group.
    - A rail terminal may be in Interconnect Models with the following interface combinations:
        - pin to buffer
        - pin to die pad (in one Interconnect Model) and die pad to buffer (in another Interconnect Model)
        - pin to die pad
        - die pad to buffer
        - pin only
        - die pad only
        - buffer only

- A rail terminal may not be in Interconnect Models with the following interface combinations:
  - pin to buffer (in one Interconnect Model) and pin to die pad (in another Interconnect Model)
  - pin to buffer (in one Interconnect Model) and die pad to buffer (in another Interconnect Model)
  - pin to buffer, pin to die pad, and die pad to buffer in three separate Interconnect Models

Note that these rules apply to the complete list of Interconnect Models that are included in each Interconnect Model Group, regardless of which Interconnect Model Sets contain the Interconnect Models.

All Interconnect Models without I/O terminals, but with only rail terminals are available for simulations.

If an *_I/O pin_name appears on terminal lines of Interconnect Model(s) in the Interconnect Model Group with the interface combinations: pin to buffer, or pin to die pad and die pad to buffer, then the Interconnect Model(s) in the Interconnect Model Group define the full interconnect electrical path between the pin and buffer interfaces. If this is not the case, then:

- If an *_I/O pin_name appears only in a pin to die pad Interconnect Model in the Interconnect Model Group, then the *_I/O pin_name electrical path from the die pad to buffer shall be shorted.
- If an *_I/O pin_name appears only in a buffer to die pad Interconnect Model in the Interconnect Model Group, then the *_I/O pin_name electrical path from die pad to buffer shall be connected using any other existing package model in this document including those under [Package] R_pkg, L_pkg, and C_pkg entries; [Pin] R_pin, L_pin, and C_pin entries in this section; or entries under the [Define Package Model] keyword described in Section 7. Note, if several [Define Package Model] keywords exist, the EDA tool may have to select which one to use. EDA tools may provide the option to ignore any of the other package model formats and to use shorted paths instead.
- If an *_I/O pin_name does not appear on a terminal line in any Interconnect Model in an Interconnect Model Group, then the EDA tool should use any other existing package model in this document.

If a PDN structure has terminals in an Interconnect Model(s) in the Interconnect Model Group with the interface combinations: pin to buffer, or pin to die pad and die pad to buffer, then the Interconnect Model(s) in the Interconnect Model Group define the full PDN electrical path between the pin and buffer interfaces. If this is not the case, then:

- If rail terminals describe a PDN structure with only a pin to die pad Interconnect Model in the Interconnect Model Group, then the rail electrical path from the die pad to buffer shall be shorted. Note, the shorted connections from the die pad terminal names to the buffer interface rail terminal names might require using the information under the [Pin], [Pin Mapping], [Die Supply Pads] or [Bus Label] keywords in this section.

- If rail terminals describe a PDN structure with only a buffer to die pad Interconnect Model in the Interconnect Model Group, then the rail electrical path from die pad to Pin_Rail pin_name entry shall be connected using any other existing package model in this document including those with [Package] R_pkg, L_pkg, and C_pkg entries or [Pin] R_pin, L_pin, and C_pin entries in this section; or entries under the [Define Package Model] keyword described in Section 7. Note, if several [Define Package Model] keywords exist, the EDA tool may have to select which one to use. Also note, the Pad_Rail terminals have pad_name bus_label, or signal_name entries that may short the electrical connections at the die pad interface based on the information under the [Pin], [Pin Mapping], [Die Supply Pads] or [Bus Label] keywords in this section. If there are more rail pad_names than Pin_Rail pin_names, the EDA tool will have to short some pad_names to support existing package model formats.
- If there are no rail terminal names on a terminal line in any Interconnect Model in an Interconnect Model Group, then the EDA tool should use any other existing package model in this document, or ideal sources if the simulation does not need to include PDN effects.

Examples:

```
| Some [Interconnect Model Set] names used in Examples from Section 11 are
| referenced below:
|
| Example 1
|
[Interconnect Model Group] Full_ISS_PDN_1
| Interconnect Model Set   file_reference
Full_ISS_PDN_1             NA                | The [Interconnect Model Set] is
                                             | present in the .ibs file for
                                             | all pins
[End Interconnect Model Group]

| Example 2
|
[Interconnect Model Group] Full_ISS_PDN_sn_2
| Interconnect Model Set   file_reference
Full_ISS_PDN_sn_2          NA                | The [Interconnect Model Set] is
                                             | present in the .ibs file for
                                             | all I/O pins and PDN described
                                             | by signal_names (sn)
[End Interconnect Model Group]

| Example 3
|
[Interconnect Model Group] A1_TS
| Interconnect Model Set   file_reference
A1_TS                      touchstone/ts_sets.ims | [Interconnect Model Set] is
                                                  | in ts_sets.ims under the
                                                  | touchstone directory for A1
[End Interconnect Model Group]

| Example 4
|
[Interconnect Model Group] A1_ISS_buf_pad_TS_pad_pin
```

```
| Interconnect Model Set   file_reference
A1_ISS_buf_pad             NA     | Interconnect Model Sets combined from
A1_TS_pad_pin              NA     | buffer to pad and pad to pin Sets with
                                  | different file formats for A1
[End Interconnect Model Group]

| Example 5
|
[Interconnect Model Group] Full_ISS_split_IO_PDN_3
| Interconnect Model Set   file_reference
Full_ISS_buf_pin_IO_1      NA     | IO paths with common sn reference
Full_ISS_buf_pin_PDN_1     NA     | Detailed (by pin) PDN paths
                                  | PDN terminals G1-G4 get shorted
[End Interconnect Model Group]
```

Keyword: [**End Interconnect Model Group**]

Required: Yes, for each instance of the [Interconnect Model Group] keyword

Description: Indicates the end of the data for one [Interconnect Model Group].

Example:

```
[End Interconnect Model Group]
```

Keyword: **[PDN Domain]**

Required: No

Description: Marks the beginning of a PDN Domain description that is used to specify two Pad_Rail terminals connected by an on-die decoupling capacitance PDN model.

Sub-Params: Bus_label, Signal_name

Usage Rules: [PDN Domain] has a single argument, which is the name of the associated PDN Domain. The length of the PDN Domain name shall not exceed 40 characters. Blank characters are not allowed. The [PDN Domain]/[End PDN Domain] keyword pair is hierarchically scoped by the [Component] keyword. [Component] may contain zero or more [PDN Domain] keywords (each identified by a unique name).

Each [PDN Domain] keyword shall contain one or more [PDN Model] keywords and two sub-parameters consisting of two Bus_labels, two Signal_names, or one Bus_label and one Signal_name. See the [PDN Model] keyword section for a description of the content of each PDN Model.

Bus_label rules:

The Bus_label sub-parameter is followed by the name of a bus_label declared in the [Pin], [Pin Mapping], [Bus Label], or [Die Supply Pads] section of the .ibs file. If there are two or more die pads associated with the bus_label, the die pads shall be considered as shorted only when a PDN Model in the PDN Domain is enabled.

Signal_name rules:

The Signal_name sub-parameter is followed by the name of a signal_name declared in the [Pin] section of the .ibs file. Only a signal_name associated with POWER or GND can be used. If there are two or more die pads associated with the signal_name, the die pads shall be considered as shorted. In addition, if there are two or more die pads associated with the signal_name and the signal_name is associated with two or more Bus_labels, the die pads shall be considered as shorted only when a PDN Model in the PDN Domain is enabled.

A bus_label and a signal_name may appear on more than one entry under different [PDN Domain] keywords. This allows for multiple unique on-die decoupling capacitance PDN models to be placed between any arbitrary Pad_Rail pair combinations. It is not permitted to include the same pin in both terminals in a PDN Domain to avoid shorting the two terminals.

Note that it is allowed for two or more PDN Domains to be placed between the same two terminals. In this case, all PDN Domains are connected in parallel in a simulation. However, only one PDN Model from a PDN Domain is used, even though multiple PDN Models can be defined within one [PDN Domain]/[End PDN Domain] keyword section.

Note that it is possible to have a bus_label or signal_name listed under the [PDN Domain] keyword that does not have a path to the buffer rail terminals. In this case, the on-die decoupling capacitance PDN models can be used for power integrity (PI) analysis such as core power.

Example:

```
|
| PDN 1
[PDN Domain] PDN_X
Bus_label    VCC1    | assume the bus_label VCC1 includes B1 and B2 pins
Signal_name  VSS     | assume the signal_name VSS includes C1 pin
[PDN Model] PDN_model_A
…
[End PDN Model]
[End PDN Domain]

| PDN 2
[PDN Domain] PDN_Y
Signal_name  VCC2    | assume the signal_name VCC2 includes A1, B1, and B2 pins
Signal_name  VSSA    | assume the signal_name VSSA includes C2 pin
[PDN Model] PDN_model_B
…
[End PDN Model]
|
| Note: Bus_label VCC1 and Signal_name VCC2 shall not be defined as terminals
| under the same PDN Domain, because B1 and B2 pins are associated with
| both terminals and the terminals are shorted.
|
| Note: Even though A1, B1 and B2 pins are associated with two
| bus_labels VCC1 and VCC2, these pins are considered as shorted at the die
| pads in a simulation when the PDN Model under PDN_Y is enabled.
|
| Note: Even though PDN_Y and PDN_X have different power terminal names VCC1
| and VCC2, PDN Models in these PDN Domains are shorted together at die power
| pads when the PDN Models under PDN_X and PDN_Y are both enabled.
| This is because the power terminal of PDN_Y is signal_name VCC2 associated
| with bus_label VCC1.
|
| Figure: PDN_X and PDN_Y
|
```

```
|  pin(signal_name)  pad(bus_label)
|  A1(VCC2) ---------(VCC2)---------+
|                                   | shorted by PDN_model in PDN_Y
|  B1(VCC2) ---------(VCC1)+--------+
|                          |        | shorted by PDN_models in PDN_X and PDN_Y
|  B2(VCC2) ---------(VCC1)+--------+
|                          |        |
|                        PDN_X    PDN_Y
|                          |        |
|  C1(VSS)  ---------(VSS) +        |
|                                   |
|  C2(VSSA) ---------(VSSA)---------+
|
[End PDN Domain]

| PDN 3
[PDN Domain] PDN_for_VCC1_MIM
Bus_label   VCC1       | assume the bus_label VCC1 includes B1 and B2 pins
Signal_name VSS        | assume the signal_name VSS includes C1 pin
[PDN Model] PDN_model_MIM
…
[End PDN Model]
| Note: This [PDN Domain] has the same terminals as PDN_X.  In this case,
| a simulation may contain multiple on-die decoupling capacitance PDN models
| between bus_label VCC1 and signal_name VSS when the PDN Models under PDN_X
| and PDN_for_VCC1_MIM are both enabled.
[End PDN Domain]
```

Keyword: **[End PDN Domain]**

Required: Yes, for each instance of the [PDN Domain] keyword

Description: Indicates the end of the PDN Domain data.

Example:

```
[End PDN Domain]
```

Keyword: **[PDN Model]**

Required: Yes, for each instance of the [PDN Domain] keyword

Description: Marks the beginning of a PDN Model description that is used to define an on-die decoupling capacitance PDN model consisting of three RC values. These values represent MOS capacitor, MIM capacitor, metal resistance, parasitic RC, leakage current, etc. An on-die decoupling capacitance PDN model connects between the two terminals specified by the [PDN Domain] keyword.

Sub-Params: R_pdn, C_pdn, R_leak

Usage Rules: [PDN Model] has a single argument, which is the name of the associated PDN Model. The length of the PDN Model name shall not exceed 40 characters in length. Blank characters are not allowed. The [PDN Model]/[End PDN Model] keyword pair is hierarchically

scoped by the [PDN Domain]/[End PDN Domain] keywords. PDN Domain shall contain one or more [PDN Model] keywords (each identified by a unique name).

Each PDN Model shall contain R_pdn, C_pdn and R_leak sub-parameters. All three sub-parameters are required.

The EDA tool may disable all PDN Models that are contained in a PDN Domain. If two or more PDN Models are contained in one PDN Domain, the EDA tool may select one of them. The first [PDN Model] keyword entry under the [PDN Domain] keyword shall be considered the default by the EDA tool.

For each of the sub-parameters, the three columns contain values whose order does not depend on magnitude. The three entries shall be placed on a single line and shall be separated by at least one whitespace character. All three values are required for these sub-parameters. C_pdn and R_pdn shall be non-negative numbers (positive or zero). R_leak shall be a positive number (zero is not allowed). If a value of C_pdn is zero, the EDA tool may ignore it. "NA" is allowed for the second and third column only. If the second and/or third column value is NA, then the EDA tool shall use the first column value for simulation.

The electrical circuit model for the three sub-parameters is shown in Figure **2**. Terminal 1 is defined by the first Bus_label or Signal_name sub-parameter under the [PDN Domain] keyword. Terminal 2 is defined by the second Bus_label or Signal_name sub-parameter under the [PDN Domain] keyword. If two or more die pads are associated with a terminal, the PDN Model shorts the die pads when it is enabled. When the EDA tool disables all PDN Models in a PDN Domain, the [PDN Domain] keyword and its [PDN Model] keywords will not short die pads associated with the terminal.

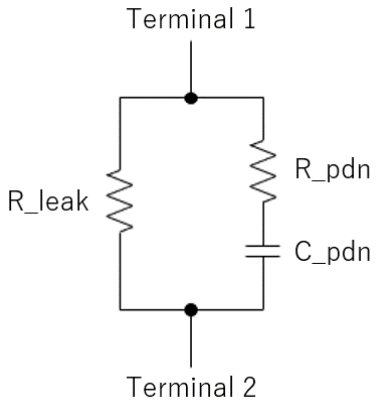


Figure 2 **– [PDN Model] Circuit**

Note that in simulation, the EDA tool may select one column from the typical, minimum or maximum data of the buffer models from the same .ibs file. At the same time, the EDA tool may select the same column of PDN Model. However, the on-die decoupling capacitance PDN model characteristics do not necessarily depend on the same variations of the buffer such as voltage, temperature and process. For example, a MIM capacitor variation may have little dependence on the same process, voltage, and temperature conditions. In such cases, the model maker may list the same three values for the entry of the PDN Model sub-parameters. In addition, the on-die decoupling capacitance PDN model characteristics can vary depending on many other factors

unrelated to the variation of the buffer. For example, the on-die capacitance of a gated power supply can vary due to the state of the gate. In such cases, the model maker may include multiple PDN Models in one PDN Domain. Based on the condition chosen by the user, the EDA tool may select one of them for simulation.

Other Notes: The Interconnect Model and Series Model can also be used to represent on-die decoupling capacitance PDN characteristics and can co-exist with PDN Model. The model maker should ensure that on-die decoupling capacitance PDN characteristics are not double counted.

When a PDN Model is used together with an Interconnect Model that does not have die pad (pin to buffer, pin only or buffer only) interfaces, there is no connection between them at the die pads. For example, when an Interconnect Model is intended for use in the pin to buffer rail path and a PDN Model is intended for use as the die pad to die pad decoupling of the rail, the PDN Model does not affect the I/O buffer in an unintended manner.

Example:

```
[PDN Domain] PDN_for_VDDQ
Bus_label VDDQ | VDDQ is an IO power supply for a DDR3/4 combo PHY
Signal_name VSS
[PDN Model] DDR3
|VDDQ Voltage Range    1.5    1.425  1.575
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  5n     4n     6n
R_pdn                  20m    30m    10m
R_leak                 15k    17k    11k
[End PDN Model]
|
[PDN Model] DDR4
|VDDQ [Voltage Range]  1.2    1.14   1.26
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  1.5n   1n     1.8n
R_pdn                  20m    30m    10m
R_leak                 15k    17k    11k
[End PDN Model]
[End PDN Domain]

[PDN Domain] MOS_capacitor_for_VCC
Bus_label VCC
Signal_name VSS
[PDN Model]
|VCC Voltage Range     0.9    0.84   0.96
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  200n   150n   250n
R_pdn                  3m     4m     1m
R_leak                 5k     8k     2k
[End PDN Model]
[End PDN Domain]

[PDN Domain] MIM_capacitor_for_VCC
Bus_label VCC
Signal_name VSS
| MIM cap does not depend on MOS PVT variations,
```

```
| but instead varies with intermetal dielectric and metal.
[PDN Model] Medium_MIM
|VCC Voltage Range     0.9    0.84   0.96
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  70n    70n    70n
R_pdn                  0      0      0   | R_pdn: Short
R_leak                 1G     1G     1G  | R_leak: Open
[End PDN Model]
|
[PDN Model] Large_MIM
|VCC Voltage Range     0.9    0.84   0.96
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  72n    72n    72n
R_pdn                  0      0      0   | R_pdn: Short
R_leak                 1G     1G     1G  | R_leak: Open
[End PDN Model]
|
[PDN Model] Small_MIM
|VCC Voltage Range     0.9    0.84   0.96
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  67n    67n    67n
R_pdn                  0      0      0   | R_pdn: Short
R_leak                 1G     1G     1G  | R_leak: Open
[End PDN Model]
[End PDN Domain]

[PDN Domain] Gated_area_for_VCC
Bus_label VCC
Signal_name VSS
[PDN Model] Gate_off
|VCC Voltage Range     0.9    0.84   0.96
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  0n     0n     0n  | C_pdn: zero (ignored)
R_pdn                  0      0      0   | R_pdn: Short
R_leak                 1G     1G     1G  | R_leak: Open
[End PDN Model]
|
[PDN Model] Gate_on
|VCC Voltage Range     0.9    0.84   0.96
|Temperature           25     125    -40
|MOS Process Corner    TT     SS     FF
C_pdn                  21n    18n    22n
R_pdn                  15m    18m    11m
R_leak                 17k    20k    14k
[End PDN Model]
[End PDN Domain]
```

Keyword: **[End PDN Model]**

Required: Yes, for each instance of the [PDN Model] keyword

Description: Indicates the end of the PDN Model data.

Example:

```
[End PDN Model]
```

Keyword: **[Pin Mapping]**

Required: No

Description: Used to indicate the power and/or ground buses to which a given driver, receiver or terminator is connected.

Sub-Params: pulldown_ref, pullup_ref, gnd_clamp_ref, power_clamp_ref, ext_ref

Usage Rules: The [Pin Mapping] keyword names the connections between POWER and/or GND pins and buffer and/or terminator voltage supply references using unique bus labels. All buses with identical labels are assumed to be connected with an ideal short. Each label must be associated with at least one pin whose model_name is POWER or GND. If a bus label defined in [Pin Mapping] is associated with more than one pin whose model_name is POWER or GND, then all of these associated pins shall have the same signal_name. Bus labels must not exceed 15 characters.

Each line must contain either three, five or six entries. Use the reserved word NC where an entry is required but a bus connection is not made.

The first column contains a pin name. Each pin name must match one of the pin names declared in the [Pin] section of the [Component].

For buffers and terminators, the remaining columns correspond to the voltage supply references for the named pin. Each [Model] supply reference is connected to a particular bus through a bus label in the corresponding column.

The second column, pulldown_ref, designates the ground bus connections for the buffer or termination associated with that pin. The bus named under pulldown_ref is associated with the [Pulldown] I-V table for non-ECL [Model]s. This is also the bus associated with the [GND Clamp] I-V table and the [Rgnd] model unless overridden by a label in the gnd_clamp_ref column.

The third column, pullup_ref, designates the power bus connection for the buffer or termination. The bus named under pullup_ref is associated with the [Pullup] table for non-ECL [Model]s (for ECL models, this bus is associated with the [Pulldown] table). This is also the bus label associated with the [POWER Clamp] I-V table and the [Rpower] model unless overridden by a label in the power_clamp_ref column.

The fourth and fifth columns, gnd_clamp_ref and power_clamp_ref, contain entries, if needed, to specify additional ground bus and power bus connections for clamps. Finally, the sixth column, ext_ref, contains entries to specify external reference supply bus connections.

The usage of the columns changes for GND and POWER pins. For GND pins, the pulldown_ref column contains the name of the bus to which the pin connects; the pullup_ref column in this case must contain the reserved word NC. Similarly, for POWER (including external reference) pins, the pullup_ref column contains the name of the bus to which the pin connects; the pulldown_ref column in this case must contain the reserved word NC.

If the [Pin Mapping] keyword is present, then the bus connections for *every* pin listed under the [Pin] keyword whose model_name is not POWER, GND or NC shall be given. If a pin has model

name POWER or GND and there is no entry for this pin under the [Pin Mapping], [Bus Label], or [Die Supply Pads] keywords then the bus_label for that pin will be its signal name.

If a pin has no connection, then both the pulldown_ref and pullup_ref subparameters for it will be NC.

The column length limits are:

[Pin Mapping]	20 characters max
pulldown_ref	15 characters max
pullup_ref	15 characters max
gnd_clamp_ref	15 characters max
power_clamp_ref	15 characters max
ext_ref	15 characters max

For compatibility with models developed under previous IBIS versions, [Pin Mapping] lines which contain ext_ref column entries must also explicitly include entries for the pulldown_ref, pullup_ref, gnd_clamp_ref and power_clamp_ref columns. These entries can be NC.

When six columns of data are specified, the headings gnd_clamp_ref, power_clamp_ref and ext_ref must be used on the line containing the [Pin Mapping] keyword. Otherwise, these headings can be omitted.

Example:

```
[Pin Mapping] pulldown_ref pullup_ref gnd_clamp_ref power_clamp_ref ext_ref
|
1               GNDBUS1    PWRBUS1   | Signal pins and their associated
2               GNDBUS2    PWRBUS2   | ground, power and external
|                                    | reference connections
3               GNDBUS1    PWRBUS1     GNDCLMP        PWRCLAMP
4               GNDBUS2    PWRBUS2     GNDCLMP        PWRCLAMP
5               GNDBUS2    PWRBUS2     NC             PWRCLAMP REFBUS1
6               GNDBUS2    PWRBUS2     GNDCLMP        NC
7               GNDBUS2    PWRBUS2     GNDCLMP        NC       REFBUS2
|                                    | Some possible clamping
|                                    | connections are shown above
|  .                                 | for illustration purposes
|  .
11              GNDBUS1    NC        | One set of ground connections.
12              GNDBUS1    NC        | NC indicates no connection to
13              GNDBUS1    NC        | power bus.
|  .
21              GNDBUS2    NC        | Second set of ground connections
22              GNDBUS2    NC
23              GNDBUS2    NC
|  .
31              NC         PWRBUS1   | One set of power connections.
32              NC         PWRBUS1   | NC indicates no connection to
33              NC         PWRBUS1   | ground bus.
|  .
41              NC         PWRBUS2   | Second set of power connections
42              NC         PWRBUS2
43              NC         PWRBUS2
|  .
51              GNDCLMP    NC        | Additional power connections
```

```
52              NC          PWRCLMP    | for clamps
|
|  .
71              NC          REFBUS1    | External reference connections
72              NC          REFBUS2
|
| The following [Pin] list corresponds to the [Pin Mapping] shown above.
|
[Pin] signal_name model_name R_pin L_pin C_pin
|
1     OUT1          output_buffer1      | Output buffers
2     OUT2          output_buffer2      |
3     IO3           io_buffer1          | Input/output buffers
4     IO4           io_buffer2          |
5     SPECIAL1      ref_buffer1         | Buffers with POWER CLAMP but no
6     SPECIAL2      io_buffer_term1     | GND CLAMP I-V tables; two use
7     SPECIAL3      ref_buffer2         | external reference voltages
11    VSS1           GND
12    VSS1           GND
13    VSS1           GND
21    VSS2           GND
22    VSS2           GND
23    VSS2           GND
31    VCC1           POWER
32    VCC1           POWER
33    VCC1           POWER
41    VCC2           POWER
42    VCC2           POWER
43    VCC2           POWER
51    VSSCLAMP       GND                | Power connections for clamps
52    VCCCLAMP       POWER              |
71    V_EXTREF1      POWER              | External reference voltage pins
72    V_EXTREF2      POWER              |
```

Keyword: **[Bus Label]**

Required: No

Description: Defines bus_label names and associates a POWER or GND signal_name with one or more bus_label names within a Component. The bus_label names can be used to define connection points for Interconnect Model terminals.

Sub-Params: signal_name

Usage Rules: The first column shall contain a bus_label. The second column, signal_name, shall be a corresponding signal_name entry for a pin under the [Pin] keyword that uses the model_name POWER or GND.

The [Bus Label] keyword shall be followed by the string “signal_name” as a column heading.

Duplicate bus_labels are not permitted. A bus_label may be defined also by the [Pin Mapping] keyword, by a signal_name under the [Pin] keyword, and/or by the [Die Supply Pads] keyword below.

Column length limits are:

[Bus Label]	15 characters max
signal_name	40 characters max

Example:

```
[Bus Label] signal_name
VDD1        VDD
VDD2        VDD
VDD3        VDD
VSS1        VSS
VSS2        VSS
```

Keyword: **[Die Supply Pads]**

Required: No

Description: Defines supply rail die pads and associates signal_names and bus_labels with those die pads.

Sub-Params: signal_name, bus_label

Usage Rules: Only die pads with signal_names that occur on POWER or GND pins are allowed. Each line shall contain either two or three columns. The first column shall contain the supply die pad name (the column entry is also referred to as "pad_name" elsewhere in this document). The second column, signal_name, shall contain the signal name as given under the [Pin] keyword. The third column is optional. If it exists, it is a bus_label. If the third column does not exist, then the bus_label shall be the signal_name.

The [Die Supply Pads] keyword shall be followed by the strings "signal_name" and "bus_label" as column headings.

Other Notes: The data in this section consists of a list of pad_names and their corresponding signal_names and bus_labels, which can be used to mate package and on-die power delivery networks.

The keywords described above ([Pin Mapping], [Pin], [Bus Label], and [Die Supply Pads]) describe several ways to name the bus_label entries. Briefly, they are listed here:

[Pin Mapping] associates each rail pin_name with a bus_label for all rail pin_names. For the listed buffer I/O pin_names (in the first column), the bus_label entries are listed under the pulldown_ref, pullup_ref, gnd_clamp_ref, power_clamp_ref, and ext_ref columns of [Pin Mapping]. This listing of any or all POWER and/or GND pin_names (also referred to as rails) is optional.

[Pin] associates each pin_name with a signal_name. The signal_name can be used as a bus_label for rail pin_names that are not listed under [Pin Mapping] or not described by the [Bus Label] and [Die Supply Pads] keywords.

[Bus Label] also associates signal_names with bus_labels.

[Die Supply Pads] is used to define rail pad_names and to associate them with signal_name, but the second and third columns can provide another way to associate signal_names with bus_labels in a manner that may not be covered above.

Such entries can be used as terminals at designated locations in [Interconnect Model] terminal lines described later in Section 11. The keywords can also be used to describe how different

Terminal_type_qualifiers (described later) can be associated with each other. For example, a POWER or GND pin_name with a bus_label entry in [Pin Mapping] would find its corresponding signal_name from the [Pin] keyword for the same pin_name.

With these four keywords, it is possible to create bus_label names for rails in four different ways, and any or all of the four ways can be used at once.

These keywords also support the usage of each rail terminal individually, or the creation of a single terminal connecting rails with the same bus_label or signal_name, or the designation of rail pad_names that might be different than rail pin_names. With these keywords, the number of rail nets can be reduced. Also, a different number of rail terminals can be entered at each boundary to support few-to-many or many-to-few connection terminals.

Example:

```
[Die Supply Pads] signal_name bus_label
VDDQ              VDDQ
VDD1              VDD         VDDa
VDD2              VDD         VDDa
VDD3              VDD         VDDb
VSS1              VSS
VSS2              VSS
```

Keyword: **[Diff Pin]**

Required: No

Description: Associates differential pins and defines their differential receiver threshold voltages and differential driver timing delays.

Sub-Params: inv_pin, vdiff, tdelay_typ, tdelay_min, tdelay_max

Usage Rules: Enter only differential pin pairs. The first column, [Diff Pin], contains a non-inverting pin name. The second column, inv_pin, contains the corresponding inverting pin name for I/O output. Each pin name must match the pin names declared previously in the [Pin] section of the .ibs file. The third column, vdiff, contains the specified differential receiver threshold voltage between the inverting and non-inverting pins for Input or I/O model types. The fourth, fifth, and sixth columns, tdelay_typ, tdelay_min, and tdelay_max, contain launch delays of the non-inverting pins relative to the inverting pins. Each of the numerical entries may be a positive, zero, or negative number.

For differential Input or I/O model types, the differential input threshold (vdiff) overrides and supersedes the need for Vinh and Vinl.

Other Notes: The output pin polarity specification in the table overrides the [Model] Polarity specification such that the pin in the [Diff Pin] column is Non-Inverting and the pin in the inv_pin column is Inverting. This convention enables one [Model] to be used for both pins.

The column length limits are:

[Diff Pin]	20 characters max
inv_pin	20 characters max
vdiff	9 characters max
tdelay_typ	9 characters max
tdelay_min	9 characters max
tdelay_max	9 characters max

Each line must contain either four or six columns. Using four columns is an equivalent of entering "NA"s in the fifth and sixth columns. An "NA" in the vdiff column will be interpreted as a 200 mV default differential receiver threshold. "NA"s in the tdelay_typ, or tdelay_min columns are interpreted as 0 ns. If "NA" appears in the tdelay_max column, its value is interpreted as the tdelay_typ value. When using six columns, the headers tdelay_min and tdelay_max must be listed. Entries for the tdelay_min column are based on minimum magnitudes; and tdelay_max column, maximum magnitudes. One entry of vdiff, regardless of its polarity, is used for difference magnitudes.

When a [Model] that is associated with any of the pins listed under the [Diff Pin] keyword contains the [Algorithmic Model] keyword, the tdelay_*** parameters in the fourth, fifth and sixth columns of the [Diff Pin] keyword are ignored in algorithmic model interface (AMI) channel characterization simulations, i.e., they are treated as if their value would be zero.

The positioning of numerical entries and/or "NA" must not be used as an indication for the model type. The model type is determined by the model type parameter inside the [Model]s referenced by the [Diff Pin] keyword, regardless of what the [Diff Pin] keyword's entries are. The EDA tool may ignore the vdiff or the tdelay_*** parameters if not needed by the model type of the [Model], or use the default values defined above if they are needed but not provided in the [Diff Pin] keyword.

For example, an “NA” in the third column (vdiff) does not imply that the model type is Output, or three “NA”s in the tdelay columns does not mean that the model type is Input.

Note that the starting point of the flight time measurements will occur when the differential driver’s output waveforms are crossing, i.e., when the differential output voltage is zero, and consequently Vmeas, if defined, will be ignored.

Example:

```
[Diff Pin]  inv_pin  vdiff  tdelay_typ tdelay_min tdelay_max
|
 3           4       150mV    -1ns       0ns      -2ns
| For Input,  tdelay_typ/min/max ignored
| For Output, vdiff ignored
|
 7           8         0V      1ns        NA        NA
16          15       200mV     1ns
| For Input,  tdelay_typ ignored
| For Output, vdiff ignored and tdelay_min = 0ns and tdelay_max = 1ns
| For I/O,    tdelay_min = 0ns and tdelay_max = 1ns
|
 9          10         NA       NA        NA        NA
22          21         NA       NA
| For Input,  vdiff = 200 mV
| For Output, tdelay_typ/min/max = 0ns
| For I/O,    vdiff = 200 mV and tdelay_typ/min/max = 0ns
|
20          19         0V       NA
| For Output, vdiff ignored and tdelay_typ/min/max = 0ns
| For I/O,    tdelay_typ/min/max = 0ns
```

Keyword: **[Clock Pins]**

Required: No

Description: This optional keyword identifies clocking relationships between pins for a specific [Component] keyword.

Sub-Params: clocked_pins, relationship

Usage Rules: The keyword arguments consist of three columns of entries, beginning on the same line as the keyword itself. The keyword [Clock Pins] shall be followed by the string “clocked_pins” and “relationship”, all separated by whitespace. The second and any following lines consist of two columns of pin names and a string column as detailed below.

Column 1 is required and lists clock pins, one clock pin per line. These are pins from the [Pin] keyword pin_name column that correspond to clocks, either inputs or outputs, for the component.

Column 2 is required and lists clocked pins. These are the pins from the [Pin] keyword pin_name column that correspond to clocked data pins for the component. The pin named in the second column of each line is assumed to be “clocked” by the pin named in the first column.

Alternatively, column 2 may contain the pin_name of a clock pin where a timing relationship (for example, a timing skew) exists with respect to the clock pin_name in column 1. The pins listed in

columns 1 and 2 are assumed to be Inputs, Outputs, I/Os, 3-state, or Open variants of these Model_types. Included in these are the "_ECL" and "_diff" variants.

Column 3 is required and contains a string identifying the timing relationship between the pins listed in columns 1 and 2. The only permitted entry is "Unspecified", which is case-sensitive. This column is intended for future expansion.

Three entries, separated by whitespace, are required per line.

Entire lines may not be duplicated. The same pin_name may not appear in both columns of any single line.

[Clock Pins] is compatible with [Diff Pin], in that the clock pins and data pins may be single-ended or differential. For differential pins, the [Clock Pins] line shall list only the non-inverting pin(s) of the differential pair(s) as listed under the first column of the [Diff Pin] keyword.

Some Model_types are incompatible with the [Clock Pins] keyword. For both columns, prohibited Model_types include Series, Series_switch, and Terminator. Pins associated with [Model]s of these types shall not appear under [Clock Pins]. [Clock Pins] is not compatible with pin_names in [Series Pin Mapping] or function_table_group states in [Series Switch Groups].

Other Notes: [Clock Pins] is hierarchically under the [Component] keyword.

The structure of [Clock Pins] assumes that the clocking relationships cannot be redefined dynamically for the given [Component] (for example, the number of data pins supported by any one clock pin is fixed).

[Clock Pins] is compatible with [Algorithmic Model], [External Model], [External Circuit] and their associated keywords. [Clock Pins] is also compatible with [Model Selector].

[Clock Pins] is also compatible with the [Component] keyword Timing_location subparameter.

Example:

```
[Clock Pins]      clocked_pins   relationship
A1          B1     Unspecified | Data pin B1 uses clock information from Pin A1
A2          B2     Unspecified | Data pin B2 uses clock information from Pin A2
A3          B3     Unspecified
A3          B4     Unspecified | Pins B3, B4, B5 use clock information from A3
A3          B5     Unspecified | case-sensitive entry
```

Keyword: **[Series Pin Mapping]**

Required: No

Description: Used to associate two pins joined by a Series Model.

Sub-Params: pin_2, model_name, function_table_group

Usage Rules: Enter only series pin pairs. The first column, [Series Pin Mapping], contains the series pin for which input impedances are measured. The second column, pin_2, contains the other connection of the Series Model. Each pin must match the pin names declared previously in the [Pin] section of the .ibs file. The third column, model_name, associates models of type Series or Series_switch, or model selectors containing references to models of type Series or Series_switch with the pair of pins in the first two columns. Each model_name must have a corresponding model or model selector name listed in a [Model] or [Model Selector] keyword below. The usage of reserved model names (POWER, GND, or NC) within the [Series Pin Mapping] keyword is not

allowed. The fourth column, function_table_group, contains a name to associate those sets of Series_switch pins that are switched together.

Each line must contain either three or four columns. When using four columns, the header function_table_group must be listed.

One possible application is to model crossbar switches where the straight through On paths (see [Series Switch Groups]) are indicated by one designator and the cross over On paths are indicated by another designator. If the model referenced is a Series Model, then the function_table_group entry shall be omitted.

The column length limits are:

[Series Pin Mapping]	20 characters max
pin_2	20 characters max
model_name	40 characters max
function_table_group	20 characters max

Other Notes: If the model_name is for a non-symmetrical Series Model, then the order of the pins is important. The [Series Pin Mapping] and pin_2 entries must be in the columns that correspond with Pin 1 and Pin 2 of the referenced model.

This mapping covers only the series paths between pins. The package parasitics between the series pins and the terminals of the Series Model are defined by the package modeling keywords. Any other elements, such as additional capacitance or clamping circuitry, are defined by the [Model] whose name is referenced in the model_name column of the [Pin] keyword for the corresponding pins. The model_names under the [Pin] keyword that are also referenced by the [Series Pin Mapping] keyword may include any legal model or reserved model except for Series and Series_switch models. Normally the pins will reference a [Model] whose Model_type is "Terminator". For example, a Series_switch model may contain Terminator models on EACH of the pins to describe both the capacitance on each pin and some clamping circuitry that may exist on each pin. In a similar manner, Input, I/O or Output models may exist on each pin of a Series Model that is serving as a differential termination.

Also, a pin name may appear on more than one entry under the [Series Pin Mapping] keyword. This allows for multiple and perhaps different models or model selectors to be placed between the same, or any arbitrary pin pair combinations.

Example:

```
[Series Pin Mapping]  pin_2    model_name      function_table_group
|
  2                    3       CBTSeries       1    | Four independent groups
  5                    6       CBTSeries       2
  9                    8       CBTSeries       3
  12                  11       CBTSeries       4
|
  22                  23       CBTSeries       5    | Straight through path
  25                  26       CBTSeries       5
  22                  26       CBTSeries       6    | Cross over path
  25                  23       CBTSeries       6
|
  32                  33       Fixed_series         | No group needed
```

Keyword: **[Series Switch Groups]**

Required: Yes, if function_table_group column data are present under [Series Pin Mapping]

Description: Used to define allowable switching combinations of series switches described using the names of the groups in the [Series Pin Mapping] keyword function_table_group column.

Sub-Params: On, Off

Usage Rules: Each state line contains an allowable configuration. A typical state line will start with "On" followed by all of the on-state group names or an "Off" followed by all of the off-state group names. Only one of "On" or "Off" is required since the undefined states are presumed to be opposite of the explicitly defined states. The state line is terminated with the slash "/", even if it extends over several lines to fit within the 1024-character column width restriction.

The group names in the function_table_group are used to associate switches whose switching action is synchronized by a common control function. The first line defines the assumed (default) state of the set of series switches. Other sets of states are listed and can be selected through a user interface or through automatic control.

Example:

```
[Series Switch Groups]
| Function Group States
On 1 2 3 4 /           | Default setting is all switched On
|
Off 1 2 3 4 /          | All Off setting
On 1 /                 | Other possible combinations below
On 2 /
On 3 /
On 4 /
On 1 2 /
On 1 3 /
On 1 4 /
On 2 3 /
On 2 4 /
On 3 4 /
On 1 2 3 /
On 1 2 4 /
On 1 3 4 /
On 2 3 4 /
| Off 4 /              | The last four lines above could have been replaced
| Off 3 /              | with these four lines with the same meaning.
| Off 2 /
| Off 1 /
|
On 5 /                 | Crossbar switch straight through connection
On 6 /                 | Crossbar cross over connection
Off 5 6 /              | Crossbar open switches
```

Keyword: **[Model Selector]**

Required: No

Description: Used to pick a [Model] from a list of [Model]s for a pin which uses a programmable buffer.

Usage Rules: A programmable buffer must have an individual [Model] section for each one of its modes used in the .ibs file. The names of these [Model]s must be unique and are listed under the [Model Selector] keyword and/or pin list. The name of the [Model Selector] keyword must match the corresponding model name listed under the [Pin] or [Series Pin Mapping] keyword and must not contain more than 40 characters. A .ibs file must contain enough [Model Selector] keywords to cover all of the model selector names specified under the [Pin] and [Series Pin Mapping] keywords.

The section under the [Model Selector] keyword must have two fields. The two fields must be separated by at least one whitespace character. The first field lists the [Model] name (up to 40 characters long). The second field contains a short description of the [Model] shown in the first field. The contents and format of this description is not standardized. However, it shall be limited in length so that none of the descriptions exceed the 1024-character length of the line that it started on. The purpose of the descriptions is to aid the user of the EDA tool in making intelligent buffer mode selections and it can be used by the EDA tool in a user interface dialog box as the basis of an interactive buffer selection mechanism.

The first entry under the [Model Selector] keyword shall be considered the default by the EDA tool for all those pins which call this [Model Selector].

The operation of this selection mechanism implies that a group of pins which use the same programmable buffer (i.e., model selector name) will be switched together from one [Model] to another. Therefore, if two groups of pins, for example an address bus and a data bus, use the same programmable buffer, and the user must have the capability to configure them independently, one can use two [Model Selector] keywords with unique names and the same list of [Model] keywords; however, the usage of the [Model Selector] is not limited to these examples. Many other combinations are possible.

Example:

```
[Pin]   signal_name     model_name      R_pin   L_pin   C_pin
|
  1     RAS0#           Progbuffer1     200.0m  5.0nH   2.0pF
  2     EN1#            Input1          NA      6.3nH   NA
  3     A0              3-state
  4     D0              Progbuffer2
  5     D1              Progbuffer2     320.0m  3.1nH   2.2pF
  6     D2              Progbuffer2
  7     RD#             Input2          310.0m  3.0nH   2.0pF
|  .
|  .
|  .
  18     Vcc3            POWER
|
[Model Selector]        Progbuffer1
|
OUT_2       2 mA buffer without slew rate control
OUT_4       4 mA buffer without slew rate control
OUT_6       6 mA buffer without slew rate control
OUT_4S      4 mA buffer with slew rate control
OUT_6S      6 mA buffer with slew rate control
|
```

```
[Model Selector]        Progbuffer2
|
OUT_2       2 mA buffer without slew rate control
OUT_6       6 mA buffer without slew rate control
OUT_6S      6 mA buffer with slew rate control
OUT_8S      8 mA buffer with slew rate control
OUT_10S    10 mA buffer with slew rate control
```

6 BUFFER MODELING

6.1 MODEL STATEMENT

Keyword: **[Model]**

Required: Yes

Description: Used to define a model, and its attributes.

Sub-Params: Model_type, Polarity, Enable, Vinl, Vinh, C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, C_comp_gnd_clamp, Vmeas, Cref, Rref, Vref

Usage Rules: Each model type must begin with the keyword [Model]. The model name shall match one that is listed under a [Pin], [Model Selector] or [Series Pin Mapping] keyword and must not contain more than 40 characters. A .ibs file must contain enough [Model] keywords to cover all of the model names specified under the [Pin], [Model Selector] and [Series Pin Mapping] keywords, except for those model names that use reserved words (POWER, GND and NC).

Model_type must be one of the following:

Input, Output, I/O, 3-state, Open_drain, I/O_open_drain, Open_sink, I/O_open_sink, Open_source, I/O_open_source, Input_ECL, Output_ECL, I/O_ECL, 3-state_ECL, Terminator, Series, and Series_switch.

For true differential models documented under Section 6.3, Model_type must be one of the following:

Input_diff, Output_diff, I/O_diff, and 3-state_diff

Special usage rules for particular model types are provided in Table 1. Some definitions are included for clarification.

Table 1 – Special Rules for Keyword [Model]

Model Type	Definition
Input I/O I/O_open_drain I/O_open_sink I/O_open_source	These model types must have Vinl and Vinh defined. If they are not defined, the parser issues a warning and the default values of Vinl = 0.8 V and Vinh = 2.0 V are assumed.
Input_ECL I/O_ECL	These model types must have Vinl and Vinh defined. If they are not defined, the parser issues a warning and the default values of Vinl = 0.8 V and Vinh = 2.0 V are assumed.

Model Type	Definition
Terminator	This model type is an input-only model that can have analog loading effects on the circuit being simulated but has no digital logic thresholds. Examples of terminators are: capacitors, termination diodes, and pullup resistors.
Output	This model type indicates that an output always sources and/or sinks current and cannot be disabled.
3-state	This model type indicates that an output can be disabled, i.e., put into a high impedance state.
Open_sink Open_drain	These model types indicate that the output has an OPEN side (do not use the [Pullup] keyword, or if it must be used, set I = 0 mA for all voltages specified) and the output SINKS current. Open_drain model type is retained for backward compatibility.
Open_source	This model type indicates that the output has an OPEN side (do not use the [Pulldown] keyword, or if it must be used, set I = 0 mA for all voltages specified) and the output SOURCES current.
Input_ECL Output_ECL I/O_ECL 3-state_ECL	These model types specify that the model represents an ECL type logic that follows different conventions for the [Pulldown] keyword.
Series	This model type is for Series Models that can be described by [R Series], [L Series], [Rl Series], [C Series], [Lc Series], [Rc Series], [Series Current] and [Series MOSFET] keywords.
Series_switch	This model type is for series switch models that can be described by [On], [Off], [R Series], [L Series], [Rl Series], [C Series], [Lc Series], [Rc Series], [Series Current] and [Series MOSFET] keywords.
Input_diff Output_diff I/O_diff 3-state_diff	These model types specify that the model defines a true differential model available directly through the [External Model] keyword documented in Section 6.3.

The Model_type subparameter is required.

The C_comp subparameter is required only when C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, and C_comp_gnd_clamp are not present. If the C_comp subparameter is not present, at least one of the C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, or C_comp_gnd_clamp subparameters is required. It is permitted to include the C_comp subparameter together with one or more of the remaining C_comp_* subparameters, but in that case the EDA tool will have to make a decision whether to use C_comp or the C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, and C_comp_gnd_clamp subparameters. Under no circumstances should the EDA tool use the value of C_comp simultaneously with the values of the other C_comp_* subparameters.

The C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, and C_comp_gnd_clamp (referred to hereinafter as "C_comp_*") subparameters define die capacitance. These values should not include the capacitance of the package. These are intended to represent the parasitic capacitances of those structures whose I-V characteristics are described by the [Pullup], [Pulldown], [POWER Clamp] and [GND Clamp] I-V tables, respectively. For this reason, the EDA tool should generate a circuit netlist so that, if defined, each of the C_comp_* capacitors are connected in parallel with their corresponding I-V tables, whether or not the I-V table exists. That is, the C_comp_* capacitors are positioned between the signal pad and the nodes defined by the [Pullup Reference], [Pulldown Reference], [POWER Clamp Reference] and [GND Clamp Reference] keywords, or the [Voltage Range] keyword and GND.

C_comp and C_comp_* are allowed to use "NA" for the min and max values only.

The Polarity, Enable, Vinl, Vinh, Vmeas, Cref, Rref, and Vref subparameters are optional.

Also, optional Rref_diff and Cref_diff subparameters discussed further in Section 6.3 support true differential buffer timing test loads. They are used only when the [Diff Pin] keyword connects two models, and each buffer references the same model. The Rref_diff and Cref_diff subparameters can be used with the Rref, Cref, and Vref subparameters for a combined differential and signal-ended timing test load. Single-ended test loads are permitted for differential applications.

The Rref_diff and Cref_diff subparameters are recognized only when the [Diff Pin] keyword connects the models. This applies to the true differential buffers in Section 6.3 and to differential buffers using identical single-ended models.

The Polarity subparameter can be defined as either Non-Inverting or Inverting, and the Enable subparameter can be defined as either Active-High or Active-Low.

The Cref and Rref subparameters correspond to the test load that the semiconductor vendor uses when specifying the propagation delay and/or output switching time of the model. The Vmeas subparameter is the timing reference voltage level that the semiconductor vendor uses for measuring the model timing. Include Cref, Rref, Vref, and Vmeas information to facilitate board-level timing simulation. The assumed connections for Cref, Rref, and Vref are shown in Figure 3.

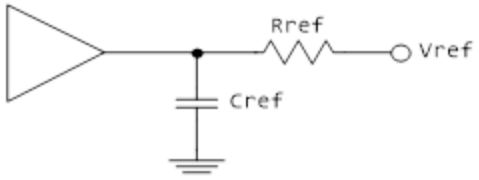


Figure 3 – Reference Load Connections

A single-ended or true differential buffer can have Rref_diff and Cref_diff (Figure 4).

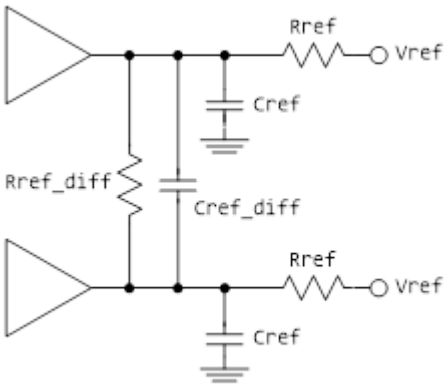


Figure 4 – Single-Ended or True Differential Buffer

Other Notes: A complete [Model] description normally contains the following keywords: [Voltage Range], [Pullup], [Pulldown], [GND Clamp], [POWER Clamp], and [Ramp]. A Terminator model may use the [Rgnd] and/or [Rpower] keywords, as well as the [Rac] and [Cac] keyword pair. The [Rgnd] and [Rpower] keywords may appear in [Model] descriptions using other Model_types. However, some models may have only a subset of these keywords. For example, an input structure normally only needs the [Voltage Range], [GND Clamp], and possibly the [POWER Clamp] keywords. If the [Rac] and [Cac] keyword pair is used, then the Model_type must be Terminator.

Examples:

```
| Signals       CLK1, CLK2,...          | Optional signal list, if desired
[Model]         Clockbuffer
Model_type      I/O
Polarity        Non-Inverting
Enable          Active-High
Vinl = 0.8V                             | Input logic "low" DC voltage, if any
Vinh = 2.0V                             | Input logic "high" DC voltage, if any
Vmeas = 1.5V              | Reference voltage for timing measurements (unused)
Cref = 50pF               | Timing specification test load capacitance value
Rref = 500                | Timing specification test load resistance value
Vref = 0                  | Timing specification test load voltage
| variable         typ             min             max
```

```
C_comp              7.0pF           5.0pF           9.0pF
C_comp_pullup       3.0pF           2.5pF           3.5pF | These four can be
C_comp_pulldown     2.0pF           1.5pF           2.5pF | used instead of
C_comp_power_clamp  1.0pF           0.5pF           1.5pF | C_comp
C_comp_gnd_clamp    1.0pF           0.5pF           1.5pF
```

For a single-ended or true differential buffer (Section 6.3):

```
[Model]          External_Model_Diff
Model_type       I/O_diff               | Requires [External Model]
Polarity         Non-Inverting
Enable           Active-High
| The [Diff Pin] vdiff value overrides the thresholds below
Vinl = 0.8V                             | Input logic "low" DC voltage, if any
Vinh = 2.0V                             | Input logic "high" DC voltage, if any
|                          | The true differential measurement point is at
|                          | the crossover voltage
|                          | The Vmeas value is overridden
Vmeas = 1.5V               | Reference voltage for timing measurements
|                          | Single-ended timing test load is still permitted
Cref = 5pF                 | Timing specification test load capacitance value
Rref = 500                 | Timing specification test load resistance value
Vref = 0                   | Timing specification test load voltage
|                          | These new subparameters are permitted for
|                          | single-ended differential operation based on the
|                          | [Diff Pin] keyword
Rref_diff = 100            | Timing specification differential resistance value
Cref_diff = 5pF            | Timing specification differential capacitance value
```

Keyword: **[Model Spec]**

Required: No

Sub-Params: Vinh, Vinl, Vinh+, Vinh-, Vinl+, Vinl-, S_overshoot_high, S_overshoot_low, D_overshoot_high, D_overshoot_low, D_overshoot_time, D_overshoot_area_h, D_overshoot_area_l, D_overshoot_ampl_h, D_overshoot_ampl_l, Pulse_high, Pulse_low, Pulse_time, Vmeas, Vref, Cref, Rref, Cref_rising, Cref_falling, Rref_rising, Rref_falling, Vref_rising, Vref_falling, Vmeas_rising, Vmeas_falling, Rref_diff, Cref_diff, Weak_R, Weak_I, Weak_V

Description: The [Model Spec] keyword defines four columns under which specification subparameters are defined.

The following subparameters are defined:

Vinh	Input voltage threshold high
Vinl	Input voltage threshold low
Vinh+	Hysteresis threshold high max Vt+
Vinh-	Hysteresis threshold high min Vt+
Vinl+	Hysteresis threshold low max Vt-
Vinl-	Hysteresis threshold low min Vt-
S_overshoot_high	Static overshoot high voltage
S_overshoot_low	Static overshoot low voltage

D_overshoot_high	Dynamic overshoot high voltage
D_overshoot_low	Dynamic overshoot low voltage
D_overshoot_time	Dynamic overshoot time
D_overshoot_area_h	Dynamic overshoot high area (in V-s)
D_overshoot_area_l	Dynamic overshoot low area (in V-s)
D_overshoot_ampl_h	Dynamic overshoot high max amplitude
D_overshoot_ampl_l	Dynamic overshoot low max amplitude
Pulse_high	Pulse immunity high voltage
Pulse_low	Pulse immunity low voltage
Pulse_time	Pulse immunity time
Vmeas	Measurement voltage for timing measurements
Vref	Timing specification test load voltage
Cref	Timing specification capacitive load
Rref	Timing specification resistance load
Cref_rising	Timing specification capacitive load for rising edges
Cref_falling	Timing specification capacitive load for falling edges
Rref_rising	Timing specification resistance load for rising edges
Rref_falling	Timing specification resistance load for falling edges
Vref_rising	Timing specification test load voltage for rising edges
Vref_falling	Timing specification test load voltage for falling edges
Vmeas_rising	Measurement voltage for rising edge timing measurements
Vmeas_falling	Measurement voltage for falling edge timing measurements
Rref_diff	Timing specification differential resistance load
Cref_diff	Timing specification differential capacitive load
Weak_R	Weak tie-up or tie-down resistance
Weak_I	Weak tie-up or tie-down current
Weak_V	Weak tie-up or tie-down voltage

Usage Rules: [Model Spec] shall follow any and all subparameters under the [Model] keyword.

For each subparameter contained in the first column, the remaining three columns hold its typical, minimum and maximum values. The entries of typical, minimum, and maximum must be placed on a single line and must be separated by at least one whitespace character. All four columns are required under the [Model Spec] keyword. However, data are required only in the typical column. If minimum and/or maximum values are not available, the reserved word "NA" must be used indicating the typical value by default.

The minimum and maximum values are used for specifying subparameter values that may track the min and max operation conditions of the [Model]. Usually it is related to the Voltage Range settings.

Unless noted below, no subparameter requires the presence of any other subparameter.

Vinh, Vinl rules:

The threshold subparameter lines provide additional min and max column values, if needed. The typ column values are still required and would be expected to override the Vinh and Vinl subparameter values specified elsewhere. Note that the syntax rule that requires inserting Vinh

and Vinl under models remains unchanged even if the values are defined under the [Model Spec] keyword.

Vinh+, Vinh-, Vinl+, Vinl- rules:

The four hysteresis subparameters (used for Schmitt trigger inputs for defining two thresholds for the rising edges and two thresholds for falling edges) must all be defined before independent input thresholds for rising and falling edges of the hysteresis threshold rules become effective. Otherwise, the standard threshold subparameters remain in effect. The hysteresis thresholds shall be at the Vinh+ and Vinh- values for a low-to-high transition, and at the Vinl+ and Vinl- values for a high-to-low transition. See Figure 5.

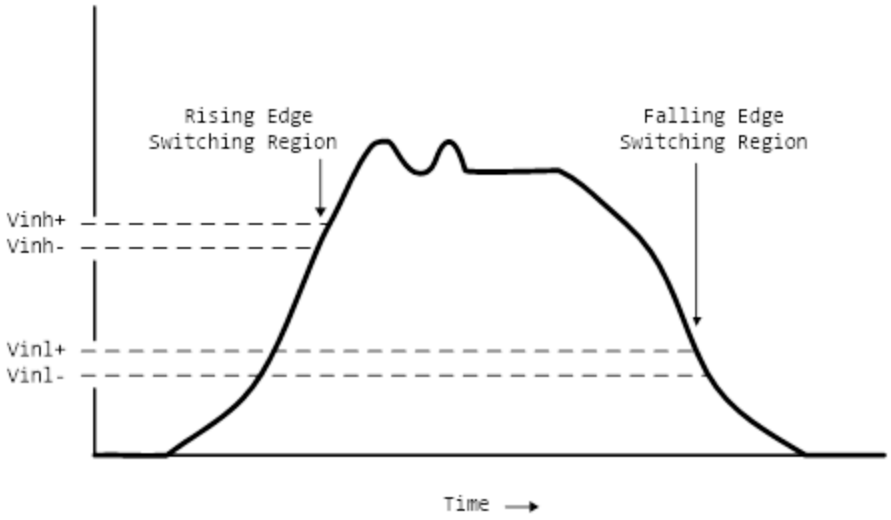


Figure 5 – Receiver Voltage with Hysteresis Thresholds

S_overshoot_high, S_overshoot_low rules:

The static overshoot subparameters provide the DC voltage values beyond which the model is no longer guaranteed to function correctly. Often these voltages are given as absolute maximum ratings. However, if any lower *_overshoot_high or higher *_overshoot_low limit for functional specification compliance exists, that limit should be used.

D_overshoot_high, D_overshoot_low, D_overshoot_time rules:

The dynamic overshoot values provide a time window during which the overshoot may exceed the static overshoot limits but be below the dynamic overshoot limits and still guarantee functional specification compliance. D_overshoot_time is required for dynamic overshoot testing. In addition, if D_overshoot_high is specified, then S_overshoot_high is necessary for testing beyond the static limit. Similarly, if D_overshoot_low is specified, then S_overshoot_low is necessary for testing beyond the static limit. See Figure 6.

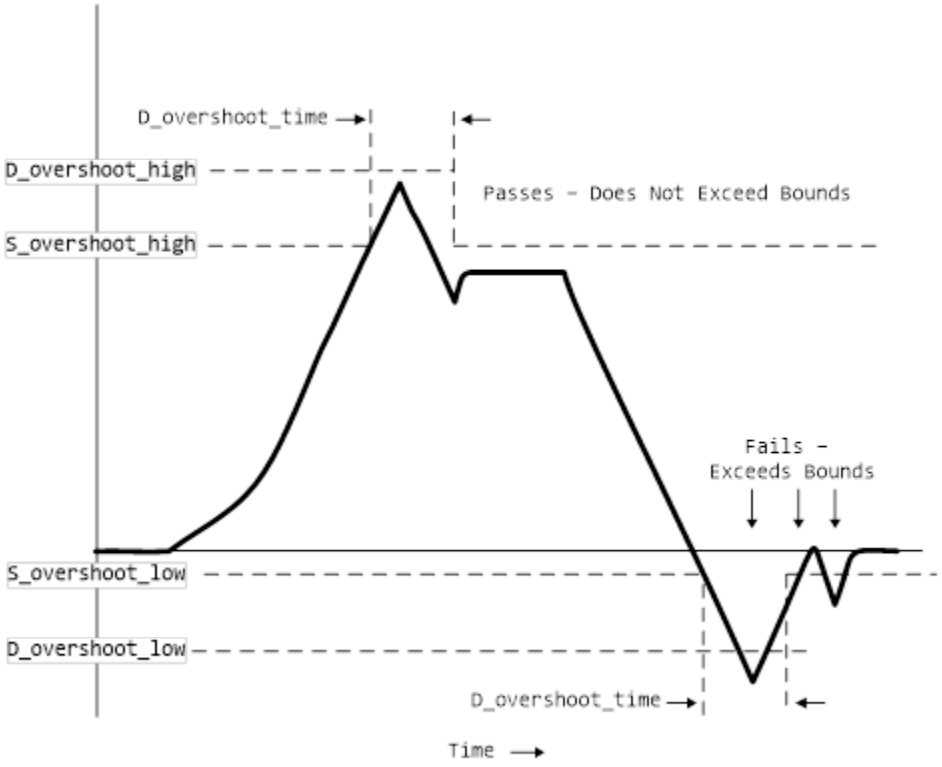


Figure 6 – Receiver Voltage with Static and Dynamic Overshoot Limits

D_overshoot_area_h, D_overshoot_area_l, D_overshoot_ampl_h, D_overshoot_ampl_l rules:

The dynamic overshoot area values define a maximum V-s area that an overshooting signal must not exceed. The high area is calculated from the point that a signal overshoots above the voltage defined by the [Power Clamp Reference] keyword until the point that the signal crosses back through this same voltage. Note that the area is defined as the complete area-under-the-curve as bounded by the limits defined above and not a "triangular" area, as shown in Figure 7. If [Power Clamp Reference] is not defined, then this crossing voltage is assumed to be defined by the [Voltage Range] keyword. The low area is calculated from the point that a signal overshoots below the voltage defined by the [GND Clamp Reference] keyword until the point that the signal crosses back through this same voltage. If [GND Clamp Reference] is not defined, then this crossing voltage is assumed to be 0.0 V. If D_overshoot_area_h is specified, then D_overshoot_ampl_h must also be specified. D_overshoot_ampl_h provides a maximum amplitude allowed for the overshoot area and is measured as voltage above the [Power Clamp Reference] voltage. Similarly, if D_overshoot_area_l is specified, then D_overshoot_ampl_l must also be specified. D_overshoot_ampl_l is measured as voltage below the [GND Clamp Reference] voltage. Both amplitude parameters should be listed as absolute (non-negative) values. Also, if D_overshoot_area_h, D_overshoot_area_l, D_overshoot_ampl_h, and D_overshoot_ampl_l are specified, then the other static and dynamic overshoot parameters are optional.

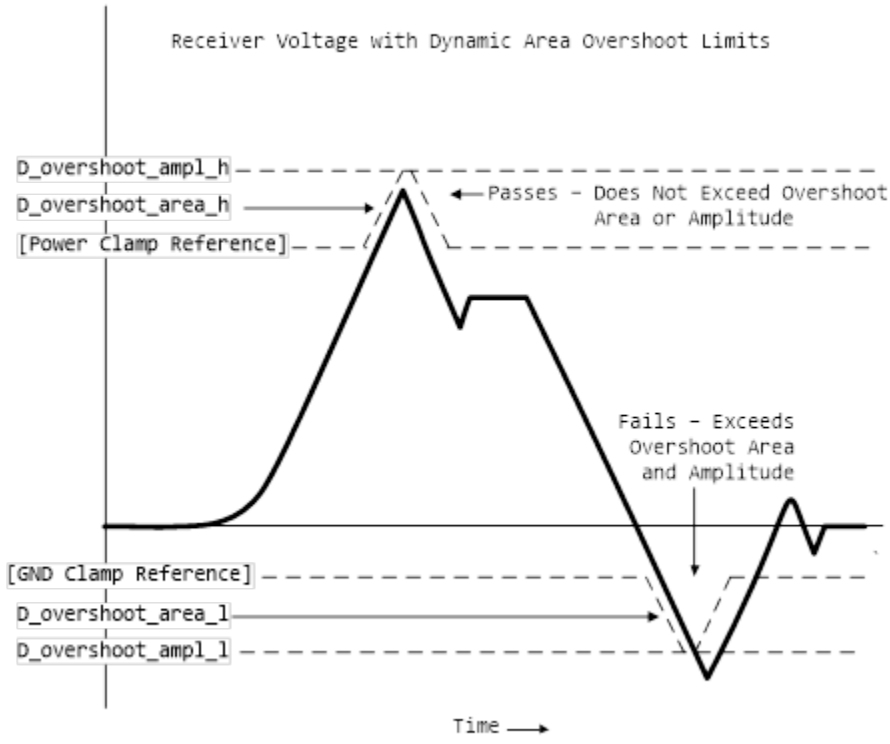


Figure 7 – Receiver Voltage with Dynamic Area Overshoot Limits

Pulse_high, Pulse_low, Pulse_time rules:

The pulse immunity values provide a time window during which a rising pulse may exceed the nearest threshold value but not exceed the pulse voltage value and still not cause the input to switch. Pulse_time is required for pulse immunity testing. A rising response is tested only if Pulse_high is specified. Similarly, a falling response is tested only if Pulse_low is specified. The rising response may exceed the Vinl value, but remain below the Pulse_high value.

Similarly, the falling response may drop below the Vinh value, but remain above the Pulse_low value. In either case, the input is regarded as immune to switching if the responses are within these extended windows. If the hysteresis thresholds are defined, then the rising response shall use Vinh- as the reference voltage, and the falling response shall use Vinl+ as the reference voltage. See Figure 8.

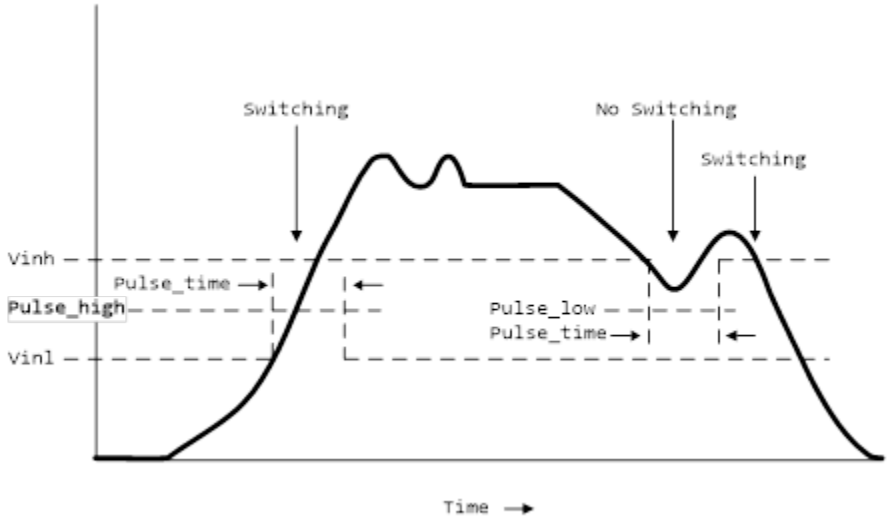


Figure 8 – Receiver Voltage with Pulse Immunity Thresholds

Vmeas, Vref, Cref, Rref rules:

The Vmeas, Vref, Cref and Rref values under the [Model Spec] keyword override their respective values entered elsewhere. Note that a Vmeas, Vref, Cref or Rref subparameter may not be used if its edge-specific version (*_rising or *_falling) is used.

Cref_rising, Cref_falling, Rref_rising, Rref_falling, Vref_rising, Vref_falling, Vmeas_rising, Vmeas_falling rules:

Use these subparameters when specifying separate timing test loads and voltages for rising and falling edges. If one “rising” or “falling” subparameter is used, then the corresponding “rising” or “falling” subparameter must be present. The values listed in these subparameters override any corresponding Cref, Vref, Rref or Vmeas values entered elsewhere.

Rref_diff, Cref_diff rules:

The Rref_diff and Cref_diff values under the [Model Spec] keyword override their respective values entered elsewhere. These subparameters are used only when the model is referenced by the [Diff Pin] keyword. These follow the same rules as the corresponding subparameters documented under the [Model] keyword. See Section 6.3 for more discussion on true and single-ended differential operation.

Weak_R, Weak_I, and Weak_V rules:

If an I/O circuit uses a simple weak tie-up or tie-down device (resistor or transistor) between the chip I/O pad and a power supply, Weak_R defines the resistance of this device and Weak_V defines the voltage of the power supply to which the device is connected. A Weak_I defines the approximate current into the device and Weak_V defines the voltage of the power supply. They apply to both static and configurable tie-up and tie-down devices.

The Weak_R, Weak_I, and Weak_V subparameters are optional and separate from the current-voltage table keywords, e.g., [Pullup], [GND Clamp], etc. They do not affect how the current-voltage tables are extracted.

(Weak_R and Weak_V) or (Weak_I and Weak_V) must always be used as a pair. Weak_R and Weak_I must not be used together.

The current flow convention for Weak_I is similar to that of [GND Clamp] and [POWER Clamp] tables. A positive sign documents a weak tie-down current. A negative sign documents a weak tie-up current.

Examples:

```
[Model Spec]
|   Subparameter          typ        min        max
|
| Thresholds
|
Vinh                      3.5        3.15       3.85     | 70% of Vcc
Vinl                      1.5        1.35       1.65     | 30% of Vcc
|
| Vinh                    3.835       3.335     4.335    | Offset from Vcc
| Vinl                    3.525       3.025     4.025    | for PECL
|
| Hysteresis
|
Vinh+                     2.0        NA         NA       | Overrides the
Vinh-                     1.6        NA         NA       | thresholds
Vinl+                     1.1        NA         NA
Vinl-                     0.6        NA         NA       | All 4 are required
|
| Overshoot
|
S_overshoot_high          5.5        5.0        6.0      | Static overshoot
S_overshoot_low          -0.5        NA         NA
D_overshoot_high          6.0        5.5        6.5      | Dynamic overshoot
D_overshoot_low          -1.0       -1.0       -1.0      | requires
|                                                        | D_overshoot_time
D_overshoot_time          20n        20n        20n      | & static overshoot
|
| Overshoot defined by area in V-s (Values from DDR2 specification)
|
D_overshoot_ampl_h        0.9        NA         NA       | Dynamic overshoot
D_overshoot_ampl_l        0.9        NA         NA       | requires area
D_overshoot_area_h        0.38n      NA         NA       | and amplitude
D_overshoot_area_l        0.38n      NA         NA       | parameters
|
| Pulse Immunity
|
Pulse_high                3V         NA         NA       | Pulse immunity
Pulse_low                 0          NA         NA       | requires
Pulse_time                3n         NA         NA       | Pulse_time
|
| Timing Thresholds
|
Vmeas                     3.68       3.18       4.68     | A 5-volt PECL
|                                                        | example
```

```
|
| Timing test load voltage reference example
|
Vref                    1.25       1.15       1.35    | An SSTL-2 example
|
|
| Rising and falling timing test load example (values from PCI-X
| specification)
|
Cref_falling            10p        10p         10p
Cref_rising             10p        10p         10p
Rref_rising             25         500         25  | typ value not specified
Rref_falling            25         500         25  | typ value not specified
Vref_rising             0          1.5         0
Vref_falling            3.3        1.5         3.6
Vmeas_rising            0.941      0.885       1.026 | vmeas = 0.285(vcc)
Vmeas_falling           2.0295     1.845       2.214 | vmeas = 0.615(vcc)
|
| Differential timing test load for true or single-ended differential model
|
Rref_diff               100        90          110
Cref_diff               5pF        NA          NA
|
| Weak tie-up examples:
|
Weak_R                  10k        NA          NA
Weak_V                  1.5V       NA          NA
|
Weak_I                  -10u       NA          NA   | negative sign for
Weak_V                  1.5V       NA          NA   | tie-up current
```

Keyword: **[Receiver Thresholds]**

Required: No

Sub-Params: Vth, Vth_min, Vth_max, Vinh_ac, Vinh_dc, Vinl_ac, Vinl_dc, Threshold_sensitivity, Reference_supply, Vcross_low, Vcross_high, Vdiff_ac, Vdiff_dc, Tslew_ac, Tdiffslew_ac

Description: The [Receiver Thresholds] keyword defines both a set of receiver input thresholds as well as their sensitivity to variations in a reference supply. The subparameters are defined as follows:

Vth, Vth_min, and Vth_max are the ideal input threshold voltages at which the output of a digital logic receiver changes state. Vth is the nominal input threshold voltage under the voltage, temperature and process conditions that define “typ”. Vth_min is the minimum input threshold voltage at “typ” conditions while Vth_max is the maximum input threshold voltage at “typ” conditions.

Vinh_ac is the voltage that a low-to-high going input waveform must reach in order to guarantee that the receiver’s output has changed state. In other words, rising above Vinh_ac is sufficient to guarantee a receiver state change. Vinh_ac is expressed as an offset from Vth.

Vinh_dc is the voltage that an input waveform must remain above (more positive than) in order to guarantee that the receiver's output will *not* change state. Vinh_dc is expressed as an offset from Vth.

Vinl_ac is the voltage that a high-to-low going input waveform must reach in order to guarantee that the receiver's output has changed state. In other words, falling below Vinl_ac is sufficient to guarantee a receiver state change. Vinl_ac is expressed as an offset from Vth.

Vinl_dc is the voltage that an input waveform must remain below (more negative than) in order to guarantee that the receiver's output will *not* change state. Vinl_dc is expressed as an offset from Vth.

Threshold_sensitivity is a unit-less number that specifies how Vth varies with respect to the supply voltage defined by the Reference_supply subparameter. Threshold_sensitivity is defined as:

$$Threshold_sensitivity = \frac{change\ in\ input\ threshold\ voltage}{change\ in\ referenced\ supply\ voltage}$$

Threshold_sensitivity must be entered as a whole number or decimal, not as a fraction.

Reference_supply indicates which supply voltage Vth tracks; i.e., it indicates which supply voltage change causes a change in input threshold. The legal arguments to this subparameter are as follows:

Power_clamp_ref	The supply voltage defined by the [POWER Clamp Reference] keyword
Gnd_clamp_ref	The supply voltage defined by the [GND Clamp Reference] keyword
Pullup_ref	The supply voltage defined by the [Pullup reference] keyword
Pulldown_ref	The supply voltage defined by the [Pulldown reference] keyword
Ext_ref	The supply voltage defined by the [External Reference] keyword

Tslew_ac and Tdiffslew_ac measure the absolute difference in time between the point at which an input waveform crosses Vinl_ac and the point it crosses Vinh_ac. The purpose of this parameter is to document the maximum amount of time an input signal may take to transition between Vinh_ac and Vinl_ac and still allow the device to meet its input setup and hold specifications. Tslew_ac is the parameter used for single-ended receivers while Tdiffslew_ac must be used for receivers with differential inputs.

Vcross_low is the least positive voltage at which a differential receiver's input signals may cross while switching and still allow the receiver to meet its timing and functional specifications. Vcross_low is specified with respect to 0 V.

Vcross_high is the most positive voltage at which a differential receiver's input signals may cross while switching and still allow the receiver to meet its timing and functional specifications. Vcross_high is specified with respect to 0 V.

Vdiff_dc is the minimum voltage difference between the inputs of a differential receiver that guarantees the receiver will not change state.

Vdiff_ac is the minimum voltage difference between the inputs of a differential receiver that guarantees the receiver will change state.

Usage Rules: [Receiver Thresholds] must follow all subparameters under the [Model] keyword and precede all other keywords of a model except [Model Spec].

The [Receiver Thresholds] keyword is valid if the model type includes any reference to input or I/O. For single-ended receivers the Vinh_ac, Vinh_dc, Vinl_ac, Vinh_dc, Vth and Tslew_ac subparameters are required and override the Vinh, Vinl, Vinh+/- and Vinl+/- subparameters declared under the [Model] or [Model Spec] keywords. For single-ended receivers the Vth_min, Vth_max, Threshold_sensitivity and Reference_supply subparameters are optional. However, if the Threshold_sensitivity subparameter is present then the Reference_supply subparameter must also be present.

For differential receivers (i.e., the [Receiver Thresholds] keyword is part of a [Model] statement that describes a pin listed in the [Diff Pin] keyword), the Vcross_low, Vcross_high, Vdiff_ac, Vdiff_dc and Tdiffslew_ac subparameters are required. The rest of the subparameters are not applicable. The Vdiff_ac and Vdiff_dc values override the value of the vdiff subparameter specified by the [Diff Pin] keyword. Note that Vcross_low and Vcross_high are valid over the device's minimum and maximum operating conditions.

Subparameter Usage Rules:

Numerical arguments are separated from their associated subparameter by an equals sign (=); whitespace around the equals sign is optional. The argument to the Reference_supply subparameter is separated from the subparameter by whitespace.

Vth at Minimum or Maximum Operating Conditions:

As described above, the Vth_min and Vth_max subparameters define the minimum and maximum input threshold values under typical operating conditions. There is no provision for directly specifying Vth under minimum or maximum operating conditions. Instead, these values are calculated using the following equation:

Vth(min/max) = Vth + [(Threshold_sensitivity) X (change in supply voltage)]*

where Vth* is either Vth, Vth_min or Vth_max as appropriate, and the supply voltage is the one indicated by the Reference_supply subparameter.

Examples:

A basic 3.3 V single-ended receiver using only the required subparameters:

```
[Receiver Thresholds]
Vth = 1.5V
Vinh_ac = +225mV
Vinh_dc = +100mV
Vinl_ac = -225mV
Vinl_dc = -100mV
Tslew_ac = 1.2ns
```

A single-ended receiver using an external threshold reference. In this case, the input threshold is the external reference voltage, so Threshold_sensitivity equals 1.

```
[Receiver Thresholds]
Vth = 1.0V
Threshold_sensitivity = 1
Reference_supply Ext_ref
Vinh_ac = +200mV
Vinh_dc = +100mV
```

```
Vinl_ac = -200mV
Vinl_dc = -100mV
Tslew_ac = 400ps
```

A fully specified single-ended 3.3 V CMOS receiver:

```
[Receiver Thresholds]
Vth = 1.5V
Vth_min = 1.45V
Vth_max = 1.53V
Threshold_sensitivity = 0.45
Reference_supply Power_clamp_ref
Vinh_ac = +200mV
Vinh_dc = +100mV
Vinl_ac = -200mV
Vinl_dc = -100mV
Tslew_ac = 400ps
```

A differential receiver:

```
[Receiver Thresholds]
Vcross_low = 0.65V
Vcross_high = 0.90V
Vdiff_ac = +200mV
Vdiff_dc = +100mV
Tdiffslew_ac = 200ps
```

Keyword: **[Add Submodel]**

Required: No

Description: References a submodel to be added to an existing model.

Usage Rules: The [Add Submodel] keyword is invoked within the [Model] keyword section to add the functionality that is contained in the submodel or list of submodels in each line that follows. Top-level model I-V and V-T data extraction is self-consistent and done with all submodels removed or de-embedded. Submodel behaviors are added in simulation according to the submodel mode described below.

The first column of the [Add Submodel] keyword contains the submodel name argument for a [Submodel] keyword defined in the same .ibs file. The second column contains a submodel mode under which the submodel is used.

If the top-level model type is one of the I/O or 3-state models, the submodel mode may be Driving, Non-Driving, or All. For example, if the submodel mode is Non-Driving, then the submodel is used only in the high-Z state of a 3-state model. Set the submodel mode to All if the submodel is to be used for all modes of operation.

The submodel mode cannot conflict with the top-level model type. For example, if the top-level model type is an Open or Output type, the submodel mode cannot be set to Non-Driving. Similarly, if the top-level model type is Input, the submodel mode cannot be set to Driving.

The submodel mode can be set to All to cover all permitted modes for any top-level model type including, for example, Input, Output, and I/O.

The [Add Submodel] keyword is not defined for Series or Series_switch model types.

Refer to the Add Submodel description in Section 6.2 of this document for the descriptions of available submodels.

Example:

```
[Add Submodel]
| Submodel_name        Mode
Bus_Hold_1             Non-Driving  | Adds the electrical characteristics of
                                    | [Submodel] Bus_Hold_1 for receiver or
                                    | high-Z mode only.
Dynamic_clamp_1        All          | Adds the Dynamic_clamp_1 model for
                                    | all modes of operation.
```

Keyword: **[Driver Schedule]**

Required: No

Description: Describes the relative model switching sequence for referenced models to produce a multi-staged driver.

Usage Rules: The [Driver Schedule] keyword establishes a hierarchical order between models and should be placed under the [Model] which acts as the top-level model. The scheduled models are then referenced from the top-level model by the [Driver Schedule] keyword.

When a multi-staged buffer is modeled using the [Driver Schedule] keyword, all of its stages (including the first stage, or normal driver) are modeled by references to [Model] keywords, each activated according to a stated schedule.

If there is support for this feature in a EDA tool, the [Driver Schedule] keyword will cause it to use the [Pulldown], [Pulldown Reference], [Pullup], [Pullup Reference], [Voltage Range], [Ramp], [Rising Waveform] and [Falling Waveform] keywords from the scheduled models instead of the top-level model, according to the timing relationships described in the [Driver Schedule] keyword. Consequently, the keywords in the above list will be ignored in the top-level model. All of the remaining keywords not shown in the above list, and all of the subparameters will be used from the top-level model and should be ignored in the scheduled model(s).

However, both the top-level and the scheduled model(s) have to be complete models, i.e., all of the required keywords must be present and follow the syntactical rules.

For backward compatibility reasons and for EDA tools which do not support multi-staged switching, the keywords in the above list can be used in the top-level [Model] to describe the overall characteristics of the buffer as if it was a composite model. It is not guaranteed, however, that such a top-level model will yield the same simulation results as a full multi-stage model. It is recommended that a “golden waveform” for the device consisting of a [Rising Waveform] table and a [Falling Waveform] table be supplied in the top-level model to serve as a reference for validation.

Even though some of the keywords are ignored in the scheduled model, it may still make sense in some cases to supply correct data with them. One such situation would arise when a [Model] is used both as a regular top-level model as well as a scheduled model.

The [Driver Schedule] table consists of five columns. The first column contains the model names of other models that exist in the .ibs file. The remaining four columns describe delays:

Rise_on_dly, Rise_off_dly, Fall_on_dly, and Fall_off_dly. The t = 0 time of each delay is the event when the EDA tool's internal pulse initiates a rising or falling transition. All specified delay values must be equal to or greater than 0. There are only five valid combinations in which these delay values can be defined:

1) Rise_on_dly with Fall_on_dly
2) Rise_off_dly with Fall_off_dly
3) Rise_on_dly with Rise_off_dly
4) Fall_on_dly with Fall_off_dly
5) All four delays defined

Note: Be careful about correct sequencing.

The four delay parameters have the meaning as described below (note that this description applies to buffer types which have both pullup and pulldown structures; for those buffer types which have only a pullup or pulldown structure, the description for the missing structure can be omitted).

Rise_on_dly is the amount of time that elapses from the internal EDA tool pulse initiating a RISING edge to the t = 0 time of the waveform or ramp that turns the I-V table of the PULLUP device ON, and the t = 0 time of the waveform or ramp that turns the I-V table of the PULLDOWN device OFF (if they were not already turned ON and OFF, respectively, by another event).

Rise_off_dly is the amount of time that elapses from the internal EDA tool pulse initiating a RISING edge to the t = 0 time of the waveform or ramp that turns the I-V table of the PULLUP device OFF, and the t = 0 time of the waveform or ramp that turns the I-V table of the PULLDOWN device ON (if they were not already turned ON and OFF, respectively, by another event).

Fall_on_dly is the amount of time that elapses from the internal EDA tool pulse initiating a FALLING edge to the t = 0 time of the waveform or ramp that turns the I-V table of the PULLDOWN device ON, and the t = 0 time of the waveform or ramp that turns the I-V table of the PULLUP device OFF (if they were not already turned ON and OFF, respectively, by another event).

Fall_off_dly is the amount of time that elapses from the internal EDA tool pulse initiating a FALLING edge to the t = 0 time of the waveform or ramp that turns the I-V table of the PULLDOWN device OFF, and the t = 0 time of the waveform or ramp that turns the I-V table of the PULLUP device ON (if they were not already turned ON and OFF, respectively, by another event).

In the above four paragraphs, the word "event" refers to the moment in time when the delay is triggered by the stimulus. This stimulus is provided to the top-level model by the simulation tool. The expiration of delays cannot generate events.

Note that some timing combinations may only be possible if the two halves of a complementary buffer are modeled separately as two open_* models.

No [Driver Schedule] table may reference a model which itself has within it a [Driver Schedule] keyword.

Use "NA" when no delay value is applicable. For each scheduled model the transition sequence must be complete, i.e., the scheduled model must return to its initial state.

Only certain numerical entry combinations are permitted to define a complete transition sequence. Table 2 gives the initial scheduled model states for each permitted set of numerical entries. The numerical delay entries, r, r1, and r2 are relative to the internal EDA tool pulse rising edge, and f, f1, and f2 are the numerical delay entries relative to internal EDA tool pulse falling edge. For the cases where two delays are given relative to the same edge, the r2 entry is larger than the r1 entry, and the f2 entry is larger than the f1 entry. For cases below, the interchanging of such values corresponds to opposite direction switching. Once the scheduled model is set to its initial state, the switching is controlled by the internal EDA tool pulse and delays relative to it.

In Table 2, the scheduled model initial states depend on the initial state of the [Model]. This top-level [Model] state ("Low" or "High") is a function of the stimulus pulse (or simulation control method) and the [Model] Polarity subparameter. For example, if a [Model] Polarity is Inverting and its stimulus pulse starts high, the [Model] initial state is "Low" and all scheduled model initial states follow the settings under the "Low" column. Two possible four-data ordering combinations are omitted because their initial states are ambiguous. Special rules to select the initial states would produce sequencing equivalent to the two-data combinations shown in the first two lines of the table.

Table 2 – Scheduled Model Initial State

Table Numerical Delay Entries				**[Model] Initial State**	
Rise_on	**Rise_off**	**Fall_on**	**Fall_off**	**Low**	**High**
r	NA	f	NA	Low	High
NA	r	NA	f	High	Low
r1	r2	NA	NA	Low	Low
r2	r1	NA	NA	High	High
NA	NA	f1	f2	High	High
NA	NA	f2	f1	Low	Low
r1	r2	f2	f1	Low	Low
r2	r1	f1	f2	High	High

The delay numbers r, r1, r2, and f, f1, f2 plus the associated model transitions should fit within the corresponding pulse width durations. Smaller pulse width stimuli may change the switching sequencing and may not support completion of the full driver sequence.

Other Notes: The added models typically consist of Open_sink (Open_drain) or Open_source models to provide sequentially increased drive strengths. The added drive may be removed within the same transition for a momentary boost or during the opposite transition.

The syntax also allows for reducing the drive strength.

Note that the Rise_on_dly, Rise_off_dly, Fall_on_dly, Fall_off_dly parameters are single value parameters, so typical, minimum and maximum conditions cannot be described with them directly. In order to account for those effects, one can refer to the fastest waveform table with the delay number and then insert an appropriate amount of horizontal lead in sections of those waveforms which need more delay.

Notice that the C_comp parameter of a multi-stage buffer is defined in the top-level model. The value of C_comp therefore includes the total capacitance of the entire buffer, including all of its stages. Since the rising and falling waveform measurements include the effects of C_comp, each of these waveforms must be generated with the total C_comp present, even if the various stages of the buffer are characterized individually.

Example:

```
[Driver Schedule]
| Model_name     Rise_on_dly  Rise_off_dly  Fall_on_dly  Fall_off_dly
  MODEL_OUT      0.0ns        NA            0.0ns        NA
|
| Examples of added multi-staged transitions
  M_O_SOURCE1    0.5ns        NA            0.5ns        NA
|              low (high-Z) to high        high to low (high-Z)
  M_O_SOURCE2    0.5n         1.5n          NA           NA
|              low to high to low          low (high-Z)
  M_O_DRAIN1     1.0n         NA            1.5n         NA
|              low to high (high-Z)        high (high-Z) to low
  M_O_DRAIN2     NA           NA            1.5n         2.0n
|              high (high-Z)               high to low to high
```

Keyword: **[Temperature Range]**

Required: Yes, if other than the default 0-, 50-, 100-degree-Celsius range

Description: Defines the temperature range over which the model is to operate.

Usage Rules: List the actual die temperatures (not percentages) in the typ, min, max format. "NA" is allowed for min and max only.

Other Notes: The [Temperature Range] keyword also describes the temperature range over which the various I-V tables and ramp rates were derived. Refer to Section 9, "NOTES ON DATA DERIVATION METHOD" for rules on which temperature values to put in the "min" and "max" columns.

Example:

```
| variable              typ            min            max
[Temperature Range]     27.0           -50            130.0
```

Keyword: **[Voltage Range]**

Required: Yes, if [Pullup Reference], [Pulldown Reference], [POWER Clamp Reference], and [GND Clamp Reference] are not present

Description: Defines the power supply voltage tolerance over which the model is intended to operate. It also specifies the default voltage rail to which the [Pullup] and [POWER Clamp] I-V data are referenced.

Usage Rules: Provide actual voltages (not percentages) in the typ, min, max format. "NA" is allowed for the min and max values only.

Other Notes: If the [Voltage Range] keyword is not present, then all four of the keywords described below must be present: [Pullup Reference], [Pulldown Reference], [POWER Clamp Reference], and [GND Clamp Reference]. If the [Voltage Range] keyword is present, the other keywords are optional and may or may not be used as required. It is legal (although redundant) for an optional keyword to specify the same voltage as specified by the [Voltage Range] keyword.

Example:

```
| variable              typ             min             max
[Voltage Range]         5.0V            4.5V            5.5V
```

Keyword: **[Pullup Reference]**

Required: Yes, if the [Voltage Range] keyword is not present

Description: Defines a voltage rail other than that defined by the [Voltage Range] keyword as the reference voltage for the [Pullup] I-V data.

Usage Rules: Provide actual voltages (not percentages) in the typ, min, max format. "NA" is allowed for the min and max values only.

Other Notes: This keyword, if present, also defines the voltage range over which the typ, min, and max dV/dt_r values are derived.

Example:

```
| variable              typ             min             max
[Pullup Reference]      5.0V            4.5V            5.5V
```

Keyword: **[Pulldown Reference]**

Required: Yes, if the [Voltage Range] keyword is not present

Description: Defines a power supply rail other than 0 V as the reference voltage for the [Pulldown] I-V data. If this keyword is not present, the voltage data points in the [Pulldown] I-V table are referenced to 0 V.

Usage Rules: Provide actual voltages (not percentages) in the typ, min, max format. "NA" is allowed for the min and max values only.

Other Notes: This keyword, if present, also defines the voltage range over which the typ, min, and max dV/dt_f values are derived.

Example:

```
| variable              typ             min             max
[Pulldown Reference]    0V              0V              0V
```

Keyword: **[POWER Clamp Reference]**

Required: Yes, if the [Voltage Range] keyword is not present

Description: Defines a voltage rail other than that defined by the [Voltage Range] keyword as the reference voltage for the [POWER Clamp] I-V data.

Usage Rules: Provide actual voltages (not percentages) in the typ, min, max format. "NA" is allowed for the min and max values only.

Other Notes: Refer to the "Other Notes" section of the [GND Clamp Reference] keyword.

Example:

```
| variable              typ            min            max
[POWER Clamp Reference] 5.0V           4.5V           5.5V
```

Keyword: **[GND Clamp Reference]**

Required: Yes, if the [Voltage Range] keyword is not present

Description: Defines a power supply rail other than 0 V as the reference voltage for the [GND Clamp] I-V data. If this keyword is not present, the voltage data points in the [GND Clamp] I-V table are referenced to 0 V.

Usage Rules: Provide actual voltages (not percentages) in the typ, min, max format. "NA" is allowed for the min and max values only.

Other Notes: Power Supplies: It is intended that standard TTL and CMOS models be specified using only the [Voltage Range] keyword. However, in cases where the output characteristics of a model depend on more than a single supply and ground, or a [Pullup], [Pulldown], [POWER Clamp], or [GND Clamp] table is referenced to something other than the default supplies, use the additional "reference" keywords.

Example:

```
| variable              typ            min            max
[GND Clamp Reference]   0V             0V             0V
```

Keyword: **[External Reference]**

Required: Yes, if a receiver's input threshold is determined by an external reference voltage

Description: Defines a voltage source that supplies the reference voltage used by a receiver for its input threshold reference.

Usage Rules: Provide actual voltages (not percentages) in the typ, min max format. "NA" is allowed for the min and max values only. Note that the numerically largest value should be placed in "max" column, while the numerically smallest value should be placed in the "min" column.

Example:

```
| variable              typ          min           max
[External Reference]   1.00V        0.95V         1.05V
```

Keyword: **[C Comp Corner]**

Required: Yes, if the [C Comp Model] keyword is present

Description: Used to define C_comp values associated with the typ/min/max corner

Sub-Params: C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, C_comp_gnd_clamp

Usage Rules: If [C Comp Corner] is present, its subparameters take precedence over any and all C_comp, C_comp_* subparameters of [Model]. The entries are values associated with each of the typ/min/max corners rather than entered by magnitude as with the other C_comp subparameters.

The C_comp subparameter under [C Comp Corner] is required only when C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, and C_comp_gnd_clamp are not present. If the C_comp subparameter is not present, at least one of the C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, or C_comp_gnd_clamp subparameters is required. It is not illegal to include the C_comp subparameter together with one or more of the remaining C_comp_* subparameters, but in that case the EDA tool will have to make a decision whether to use C_comp or the C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, and C_comp_gnd_clamp subparameters. Under no circumstances should the EDA tool use the value of C_comp simultaneously with the values of the other C_comp_* subparameters.

C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, and C_comp_gnd_clamp are intended to represent the parasitic capacitances of those structures whose I-V characteristics are described by the [Pullup], [Pulldown], [POWER Clamp] and [GND Clamp] I-V tables. For this reason, the EDA tool should generate a circuit netlist so that, if defined, each of the C_comp_* capacitors is connected in parallel with its corresponding I-V table(s), whether or not the I-V table(s) exist(s). That is, the C_comp_* capacitors are positioned between the signal pad and the nodes defined by the [Pullup Reference], [Pulldown Reference], [POWER Clamp Reference] and [GND Clamp Reference] keywords, or the [Voltage Range] keyword and GND.

The C_comp and C_comp_* subparameters define die capacitance. These values should not include the capacitance of the package. C_comp and C_comp_* are allowed to use "NA" for the min and max values only.

Other Notes: When C_comp values are obtained by extraction under the corner process, voltage, and temperature conditions, the C_comp* entries are often positioned with the maximum values under the min column and the minimum values under the max column. C_comp* entries under other keywords are entered into columns by numerical magnitude. The [C Comp Corner] entries override all other C_comp* entries under other keywords.

Example:

```
[C Comp Corner]
| variable         typ             min             max
C_comp             7.0pF           9.0pF           5.0pF
|
C_comp_pullup      3.0pF           3.5pF           2.5pF | These four can be
C_comp_pulldown    2.0pF           2.5pF           1.5pF | used instead of
C_comp_power_clamp 1.0pF           1.5pF           0.5pF | C_comp
C_comp_gnd_clamp   1.0pF           1.5pF           0.5pF
|
```

Keyword: **[C Comp Model]**, **[End C Comp Model]**

Required: No

Description: Defines an enhanced C_comp model referenced in an external file using either the Touchstone or IBIS-ISS languages. The [C Comp Model] has terminals compatible with the [Model] keyword or has terminals compatible with the [Model] keyword plus an additional terminal connecting between the [Model] and an input buffer probing location. Up to two [C Comp Model]/[End C Comp Model] sections may exist within a single [Model].

Sub-Params: C_comp_model_mode, Param, File_IBIS-ISS, File_TS, Number_of_terminals

Usage Rules: If [C Comp Model] is present, it overrides [C Comp Corner] or any other C_comp* representations. If [C Comp Model] is present, [C Comp Corner] is required, so that [C Comp Corner] values can be used by the EDA tool for a C_comp compensation algorithm. [C Comp Corner] values should represent the capacitance of the [C Comp Model] in Driving mode. [C Comp Corner] values may be ignored by EDA tools for a [C Comp Model] in Non-Driving mode.

Other Notes:
Some I/O or input buffers may contain series, filtering circuit elements between the buffer signal terminal and the location of the input buffer circuitry. A C Comp Model may include a unique probe point at this internal location where signal integrity and/or timing measurements could be made for the component. This location may be referenced as a Si_location and/or Timing_location by [Component]. Note that [Power Clamp] and [GND Clamp] I-V table data, if present in a [Model], is assumed to be applied at the Buffer I/O terminal and not at this internal probing location.

The following subparameters are defined:

C_comp_model_mode
Param
File_IBIS-ISS
File_TS
Number_of_terminals = <value>

In addition to these subparameters, the [C Comp Model]/[End C Comp Model] section may contain lines describing terminals and their connections. No specific subparameter name or other string is used to identify terminal lines.

Unless noted below, no C Comp Model subparameter requires the presence of any other subparameter.

C_comp_model_mode rules:

The subparameter C_comp_model_mode is required and may be either Driving, Non-Driving, or All. If the top-level model type is one of the I/O or 3-state models, C_comp_model_mode may be Driving, Non-Driving, or All, and up to two C Comp Models may be defined, one for Driving mode and one for Non-Driving mode. For example, if C_comp_model_mode is set to Non-Driving, then the C Comp Model is used only in the high-Z state of a 3-state model.

If the top-level model type is I/O or 3-state, and there is only one C Comp Model with C_comp_model_mode set to Driving or Non-Driving, the EDA tool shall use the [C Comp Corner] values for the undefined mode in the simulation.

The C_comp_model_mode cannot conflict with the top-level model type. For example, if the top-level model type is Open or Output, C_comp_model_mode cannot be set to Non-

Driving. Similarly, if the top-level model type is Input, C_comp_model_mode cannot be set to Driving.

The C_comp_model_mode can be set to All to cover all permitted modes for any top-level model type including, for example, Input, Output, and I/O.

Example:

```
C_comp_model_mode  Driving
```

Param rules:

The subparameter Param is optional and only legal with the File_IBIS-ISS subparameter documented below. Param is illegal with the File_TS subparameter documented below. Param shall be followed by several arguments: an unquoted string argument giving the name of the parameter to be passed into the IBIS-ISS subcircuit, a reserved word for the parameter format, and other arguments based on the parameter format to be passed into the IBIS-ISS subcircuit. Valid entries for format are:

Value – A single numerical value or string value.

Corner – Three numerical values or three string values (surrounded by double quotes) located in the typ, min, and max columns. A typ value is required. Either or both the min and max entries may be NA, in which cases the typ entry is used. The typ, min, and max parameters are associated with the corner_name Typ, Min, and Max files and their corresponding circuit_names, respectively.

Several Param lines are permitted as long as each of the parameter names is unique within the [C Comp Model]/[End C Comp Model] section.

The numerical value rules follow the scaling conventions in Section 3.2, “SYNTAX RULES”. The EDA tool is responsible for translating IBIS specified parameters into IBIS-ISS parameters. For example, 1 megaohm, would be represented as 1M in Param value according to the Section 3 rules, but would be converted by the EDA tool to case-insensitive 1meg (1X is not recommended) or 1E6 for IBIS-ISS use. Quoted string parameters in IBIS are converted to the string parameter syntax in IBIS-ISS subcircuits. For example, the Param value “typ.s1p” would be converted to str(‘typ.s1p’) in IBIS-ISS subcircuits.

Examples:

```
| Param  param_name format    typ       min       max
Param    R_esr      Corner    4.0       6.0       2.0
Param    C_123      Value     425f
Param    ts_file    Corner    "typ.s1p" "min.s1p" "max.s1p"
```

File_IBIS-ISS rules:

Either File_IBIS-ISS or File_TS is required for a [C Comp Model]/[End C Comp Model] section. The File_IBIS-ISS subparameter is followed by three unquoted string arguments consisting of corner_name, file_reference, and circuit_name (.subckt name) for an IBIS-ISS file. The IBIS-ISS file under file_reference shall be located in the same directory as the referencing .ibs file or in a specified directory under the referencing file as determined by the directory path (i.e., a file reference containing a relative path to a directory below that of the referencing .ibs file is permitted). The corner_name shall be Typ, Min or Max. File_IBIS-ISS for the Typ corner_name is required, and File_IBIS-ISS for the Min and Max

corner_names are optional. If present, each File_IBIS-ISS shall have a unique corner_name. If File_IBIS-ISS for either the Min or Max corner_name is missing, the File_IBIS-ISS for the Typ corner_name shall be used to describe the missing corner_name file reference. The Min and Max file_names should represent minimum (slow) and maximum (fast) model conditions respectively.

Examples:

```
| file_type    corner_name  file_reference  circuit_name (.subckt name)
File_IBIS-ISS  Typ          c_comp.iss      c_comp_typ
File_IBIS-ISS  Min          c_comp.iss      c_comp_min | same file as Typ
File_IBIS-ISS  Max          c_comp_max.iss  c_comp_max | in separate file
```

File_TS rules:

Either File_TS or File_IBIS-ISS is required for a [C Comp Model]/[End C Comp Model] section. File_TS is followed by three unquoted string arguments for typ, min, and max Touchstone file references. The typ entry is required and shall point to a Touchstone file representing typical conditions and located in the same directory as the referencing .ibs file or in a specified directory under the referencing file as determined by the directory path (i.e., a file reference containing a relative path to a directory below that of the referencing .ibs file is permitted). The min and max entries may point to the same file or other files representing minimum (slow) and maximum (fast) models or contain NA. If the entry is NA, the typical file entry shall be used.

Examples:

```
| file_type  typ             min             max
File_TS      c_comp_typ.s5p  c_comp_min.s5p  c_comp_max.s5p

| file_type  typ             min             max
File_TS      c_comp_typ.s4p  c_comp_min.s4p  NA
```

Number_of_terminals rules:

The Number_of_terminals subparameter is required and defines the number of Terminals associated with the C Comp Model. The subparameter name shall be followed by a single integer argument on the same line. The argument shall be separated from the subparameter name by the "=" character. The subparameter name, "=" character, and argument may optionally be separated by whitespace.

Only one Number_of_terminals subparameter may appear for a given [C Comp Model] keyword. The Number_of_terminals subparameter shall appear before any terminal lines and after all other subparameters for a given C Comp Model.

For File_IBIS-ISS, the Number_of_terminals value shall be equal to the number of subcircuit terminals for an IBIS-ISS subcircuit. Because an IBIS-ISS subcircuit requires at least two terminals including a reference, the Number_of_terminals value shall be 2 or greater. The IBIS-ISS subcircuit terminals shall not contain an ideal reference node (SPICE node 0 or its synonyms).

For File_TS, the Number_of_terminals value shall be a value equal to N+1 (where N is the number of ports in the Touchstone file). Because a Touchstone file requires at least one port, the Number_of_terminals value shall be 2 or greater.

Example:
```
Number_of_terminals = 2
```

Terminal line rules:

Terminal lines shall appear after the Number_of_terminals subparameter and before the [End C Comp Model] keyword. No reserved word identifies terminal lines. Each terminal line contains information on a terminal of an IBIS-ISS subcircuit (or Touchstone file). Two or more terminal lines may appear under a given [C Comp Model] keyword. At least one signal and one reference terminal line is required.

Terminal lines are of the following form, with each identifier separated by whitespace:

<Terminal_number> <Terminal_type>

Terminal_number

The Terminal_number is the identifier for a specific terminal. The value shall be 1 or greater and less than or equal to the Number_of_terminals. The same Terminal_number shall not appear more than once for a given C Comp Model.

For File_IBIS-ISS, the Terminal_number entry shall match the IBIS-ISS terminal (node) position. The Terminal_number entries may be listed in any order so long as there are no duplicate entries. Each IBIS-ISS terminal shall have a terminal line entry.

For File_TS, the Terminal_number entry shall match the Touchstone file port number or reference terminal line, as shown below. The Terminal_number entries may be listed in any order so long as there are no duplicate entries. The terminal line for Terminal_number N+1 is required as a reference terminal for each port and shall be connected to a rail terminal or A_gnd in the C Comp Model. All ports of the Touchstone file shall have a terminal line entry.

Terminal_number	Port
1	1
2	2
…	
N	N
N+1	Reference terminal for the Touchstone file

Terminal_type

The Terminal_type is a string that identifies whether the terminal is a reference, supply or I/O terminal (note that "I/O" in this context is a synonym for "signal", as opposed to "supply" or "rail"; it is not intended to imply model type as used in the "Model_type" subparameter). Furthermore, if the terminal is connected to a buffer supply rail, the Terminal_type identifies to which specific buffer rail the terminal is connected. The Terminal_type shall be one of the following:

- Buffer_I/O
- Buffer_I
- Pullup_ref
- Pulldown_ref
- Power_clamp_ref

- Gnd_clamp_ref
- Ext_ref
- A_gnd

Any given Terminal_type may appear only once in any C Comp Model.

Terminal_type A_gnd defines a connection to the simulator global reference node. The A_gnd node can be used at any interface.

Terminal_type A_gnd is not required under File_TS or File_IBIS-ISS.

If present under File_TS, Terminal_type A_gnd may be used only once on the terminal line with Terminal_number N+1.

Terminal_type entries are summarized in Table 3.

Table 3 – Terminal_type Definitions

Terminal_type	Definition
Buffer_I/O	Connects to the [Model]'s signal terminal.
Buffer_I	Available when there is a series circuit element between the Buffer_I/O terminal and the input buffer, where signal integrity and/or timing measurements could be made for the component. This location may be referenced as a Si_location and/or Timing_location by [Component].
Pullup_ref	Connects to the [Model]'s pullup reference.
Pulldown_ref	Connects to the [Model]'s pulldown reference.
Power_clamp_ref	Connects to the [Model]'s power clamp reference.
Gnd_clamp_ref	Connects to the [Model]'s ground clamp reference.
Ext_ref	Connects to the [Model]'s external reference.
A_gnd	Connects to the simulator global reference node.

A C Comp Model can replace C_comp by connecting a single terminal of the C Comp Model at the same location that the [Model]'s C_comp connects (see Figure 9). If it is desired to view the analog input waveform at the input buffer, the C Comp Model can contain the terminal Buffer_I as seen in Figure 9. Buffer_I may be referenced as an Si_location and/or Timing_location by [Component]. The terminal Buffer_I is analogous to the terminal my_receive of an [External Model] as seen in Figure 27.

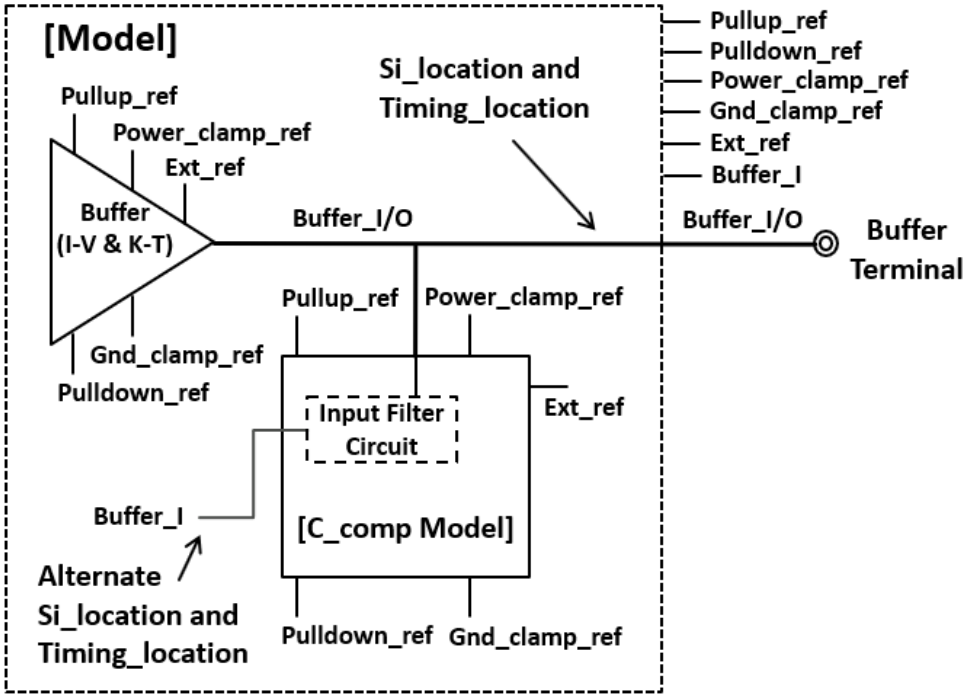


Figure 9 – C Comp Model Structure

EDA tool usage guidelines:

The EDA tool shall either use C_comp* or [C Comp Model], but not both. The user and EDA tool may assume that [C Comp Model] is more realistic than C_comp*. The EDA tool may use the [C Comp Corner] values for C_comp compensation during simulation.

Examples:

```
| Example 1 for a Model with model type I/O using one C Comp Model for
|   Driving mode and one C Comp Model for Non-Driving mode
|
[C Comp Model]
File_IBIS-ISS  Typ  A.iss  A
C_comp_model_mode Non-Driving
Param  C  Corner  1p  1.5p  0.5p
Number_of_terminals = 2
1 Buffer_I/O
2 Gnd_clamp_ref
[End C Comp Model]
|
[C Comp Model]
File_IBIS-ISS  Typ  A.iss  A
C_comp_model_mode Driving
Param  C  Corner  0.9p  1.4p  0.4p
Number_of_terminals = 2
1 Buffer_I/O
2 Gnd_clamp_ref
[End C Comp Model]
|
| Example 2 for a Model with model type I/O using one C Comp Model for
```

```
|   all modes
|
[C Comp Model]
File_IBIS-ISS  Typ  B.iss  B
C_comp_model_mode All
Number_of_terminals = 6
1 Buffer_I/O
2 Buffer_I
3 Pullup_ref
4 Pulldown_ref
5 Power_clamp_ref
6 Gnd_clamp_ref
[End C Comp Model]
```

Keywords: **[TTgnd]**, **[TTpower]**

Required: No

Description: These keywords specify the transit time parameters used to estimate the transit time capacitances or develop transit time capacitance tables for the [GND Clamp] and [POWER Clamp] tables.

Usage Rules: For each of these keywords, the three columns hold the transit time values corresponding to the typical, minimum and maximum [GND Clamp] or [POWER Clamp] tables, respectively. The entries for TT(typ), TT(min), and TT(max) must be placed on a single line and must be separated by at least one whitespace character. All three columns are required under these keywords. However, data are required only in the typical column. If minimum and/or maximum values are not available, the reserved word "NA" must be used indicating the TT(typ) value by default.

Other Notes: The transit time capacitance is added to C_comp. It is in a SPICE reference model as Ct = TT * d(Id)/d(Vd) where d(Id)/d(Vd) defines the DC conductance at the incremental DC operating point of the diode, and TT is the transit time. This expression does not include any internal series resistance. Such a resistance is assumed to be negligible in practice. Assume that the internal diode current (Id) - voltage (Vd) relationship is Id = Is * (exp(q(Vd)/kT) - 1) where Is is the saturation current, q is electron charge, k is Boltzmann's constant, and T is temperature in kelvin. Then d(Id)/d(Vd) is approximately (q/kT) * Id when the diode is conducting, and zero otherwise. This yields the simplification Ct = TT * (q/kT) * Id. The Id is found from the [GND Clamp] and [POWER Clamp] operating points, and the corresponding TTgnd or TTpower is used to calculate the Ct value. If the [Temperature Range] keyword is not defined, then use the default "typ" temperature for all Ct calculations.

The effective TT parameter values are intended to APPROXIMATE the effects. They may be different from the values found in the SPICE diode equations. Refer to Section 9, "NOTES ON DATA DERIVATION METHOD" for extracting the effective values.

Examples:

```
| variable      TT(typ)         TT(min)         TT(max)
[TTgnd]         10n             12n             9n
[TTpower]       12n             NA              NA
```

Keywords: **[Pulldown]**, **[Pullup]**, **[GND Clamp]**, **[POWER Clamp]**

Required: Yes, if they exist in the design

Description: The data points under these keywords define the I-V tables of the pulldown and pullup structures of an output buffer and the I-V tables of the clamping diodes connected to the GND and the POWER pins, respectively. Currents are considered positive when their directions are into the component.

Usage Rules: In each of these sections, the first column contains the voltage value, and the three remaining columns hold the typical, minimum, and maximum current values. The four entries, Voltage, I(typ), I(min), and I(max) must be placed on a single line and must be separated by at least one whitespace character.

All four columns are required under these keywords. However, data are only required in the typical column. If minimum and/or maximum current values are not available, the reserved word "NA" must be used. "NA" can be used for currents in the typical column, but numeric values MUST be specified for the first and last voltage points on any I-V table. Each I-V table must have at least 2, but not more than 100, rows.

Other Notes: The I-V table of the [Pullup] and the [POWER Clamp] structures are "Vcc-relative", meaning that the voltage values are referenced to the Vcc pin (note that, under these keywords, all references to "Vcc" refer to the voltage rail defined by the [Voltage Range], [Pullup Reference], or [POWER Clamp Reference] keywords, as appropriate). The voltages in the data tables are derived from the equation:

$$Vtable = Vcc - Voutput$$

Therefore, for a 5 V model, -5 V in the table actually means 5 V above Vcc, which is +10 V with respect to ground; and 10 V means 10 V below Vcc, which is -5 V with respect to ground. Vcc-relative data are necessary to model a pullup structure properly, since the output current of a pullup structure depends on the voltage between the output and Vcc pins and not the voltage between the output and ground pins. Note that the [GND Clamp] I-V table can include quiescent input currents, or the currents of a 3-stated output, if so desired.

When tabulating data for ECL models, the data in the [Pulldown] table is measured with the output in the "logic low" state. In other words, the data in the table represents the I-V characteristics of the output when the output is at the most negative of its two logic levels. Likewise, the data in the [Pullup] table is measured with the output in the "logic one" state and represents the I-V characteristics when the output is at the most positive logic level. Note that in *both* of these cases, the data are referenced to the Vcc supply voltage, using the equation:

$$Vtable = Vcc - Voutput$$

Monotonicity Requirements:

To be monotonic, the I-V table data must meet any one of the following 8 criteria:

1. The current axis either increases or remains constant as the voltage axis is increased.
2. The current axis either increases or remains constant as the voltage axis is decreased.

3. The current axis either decreases or remains constant as the voltage axis is increased.
4. The current axis either decreases or remains constant as the voltage axis is decreased.
5. The voltage axis either increases or remains constant as the current axis is increased.
6. The voltage axis either increases or remains constant as the current axis is decreased.
7. The voltage axis either decreases or remains constant as the current axis is increased.
8. The voltage axis either decreases or remains constant as the current axis is decreased.

Data may be monotonic if currents from both the output stage and the clamp diode are added together as most EDA tools do.

It is intended that, for monotonicity checks, the [POWER Clamp] and [GND Clamp] tables are summed together and then added to the appropriate [Pullup] or [Pulldown] table when a buffer is driving high or low, respectively.

From this assumption and the nature of 3-statable buffers, it follows that the data in the clamping table sections are handled as constantly present tables and the [Pullup] and [Pulldown] tables are used only when needed in the simulation.

The clamp tables of an Input or I/O buffer can be measured directly with a curve tracer, with the I/O buffer 3-stated. However, sweeping enabled buffers results in tables that are the sum of the clamping tables and the output structures. Based on the assumption outlined above, the [Pullup] and [Pulldown] tables of an IBIS model must represent the difference of the 3-stated and the enabled buffer's tables (note that the resulting difference table can demonstrate a non-monotonic shape). This requirement enables the EDA tool to sum the tables, without the danger of double counting, and arrive at an accurate model in both the 3-stated and enabled conditions.

Since in the case of a non-3-statable buffer, this difference table cannot be generated through lab measurements (because the clamping tables cannot be measured alone), the [Pullup] and [Pulldown] tables of an IBIS model can contain the sum of the clamping characteristics and the output structure. In this case, the clamping tables must contain all zeroes, or the keywords must be omitted.

Examples:

```
[Pulldown]
|  Voltage    I(typ)     I(min)     I(max)
|
   -5.0V     -40.0m     -34.0m     -45.0m
   -4.0V     -39.0m     -33.0m     -43.0m
|     .
     0.0V       0.0m       0.0m       0.0m
|     .
     5.0V      40.0m      34.0m      45.0m
    10.0V      45.0m      40.0m      49.0m
|
[Pullup]                                  | Note: Vtable = Vcc - Voutput
|
|  Voltage    I(typ)     I(min)     I(max)
|
   -5.0V      32.0m      30.0m      35.0m
   -4.0V      31.0m      29.0m      33.0m
|     .
     0.0V       0.0m       0.0m       0.0m
```

```
|    .
    5.0V    -32.0m    -30.0m    -35.0m
   10.0V    -38.0m    -35.0m    -40.0m
|
[GND Clamp]
|
|  Voltage   I(typ)    I(min)    I(max)
|
   -5.0V  -3900.0m  -3800.0m  -4000.0m
   -0.7V    -80.0m    -75.0m    -85.0m
   -0.6V    -22.0m    -20.0m    -25.0m
   -0.5V     -2.4m     -2.0m     -2.9m
   -0.4V      0.0m      0.0m      0.0m
    5.0V      0.0m      0.0m      0.0m
|
[POWER Clamp]                          | Note: Vtable = Vcc - Voutput
|
|  Voltage   I(typ)    I(min)    I(max)
|
   -5.0V   4450.0m       NA        NA
   -0.7V     95.0m       NA        NA
   -0.6V     23.0m       NA        NA
   -0.5V      2.4m       NA        NA
   -0.4V      0.0m       NA        NA
    0.0V      0.0m       NA        NA
```

Keywords: **[ISSO PD]**, **[ISSO PU]**

Required: No

Description: The data points under the keyword [ISSO PD] define the effective current of the pulldown structure of a buffer as a function of the voltage on the pulldown reference node (the ground node), whereas the points under the keyword [ISSO PU] define the effective current of the pullup structure as a function of the voltage on the pullup reference node (the power node).

Usage Rules: The first column contains the voltage value at which the currents of the remaining three columns are obtained. The three remaining columns contain the typical, minimum, and maximum effective current values to be defined below of pullup/pulldown stage.

All four columns are required under this keyword. However, data are only required in the typical column. If minimum and/or maximum current values are not available, the reserved word "NA" must be used. "NA" can be used for currents in the typical column, but numeric values MUST be specified for the first and last voltage points in any table. Each table must have at least 2, but not more than 100, rows.

The [ISSO PD] table voltages are relative to the [Pulldown Reference] typ/min/max values (usually ground). The [ISSO PU] table voltages are relative to the [Pullup Reference] typ/min/max values (also usually the [Voltage Range] voltages). In the case of the [ISSO PU] table, the voltages follow the same Vtable = Vcc - Vmeasured convention as the [Pullup] table. Each of the tables are aligned with and span the typical -Vcc to Vcc voltages.

If the [ISSO PD] and [ISSO PU] keywords are not present, the effect of power supply variations on the I-V tables is not explicitly defined by the model.

The effective current table for the Isso_pd current is extracted by the following process. The buffer is set to “logic zero.” A Vtable voltage source is inserted between the [Pulldown Reference] node and the buffer as shown in Figure 10. This Vtable voltage is swept from -Vcc (typical) to +Vcc (typical) and is relative to the [Pulldown Reference] typ/min/max values for the corresponding columns. The output is connected to the VCC (typical) value as shown in Figure 10.

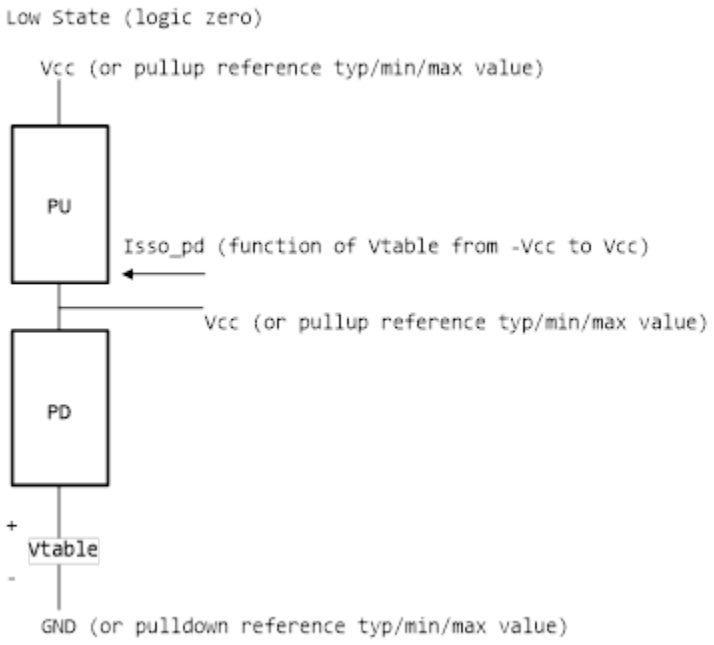


Figure 10 – Low State (Logic Zero) Isso_pd Data Collection

The effective current table for the Isso_pu current is extracted by the following process. The buffer is set to "logic one". A Vtable voltage source is inserted between the [Pullup Reference] node and the buffer as shown in Figure 11. This Vtable voltage is swept from -Vcc (typical) to +Vcc (typical) and is relative to the [Pullup Reference] typ/min/max values for the corresponding columns. The output is connected to the GND (typical) value as shown in Figure 11.

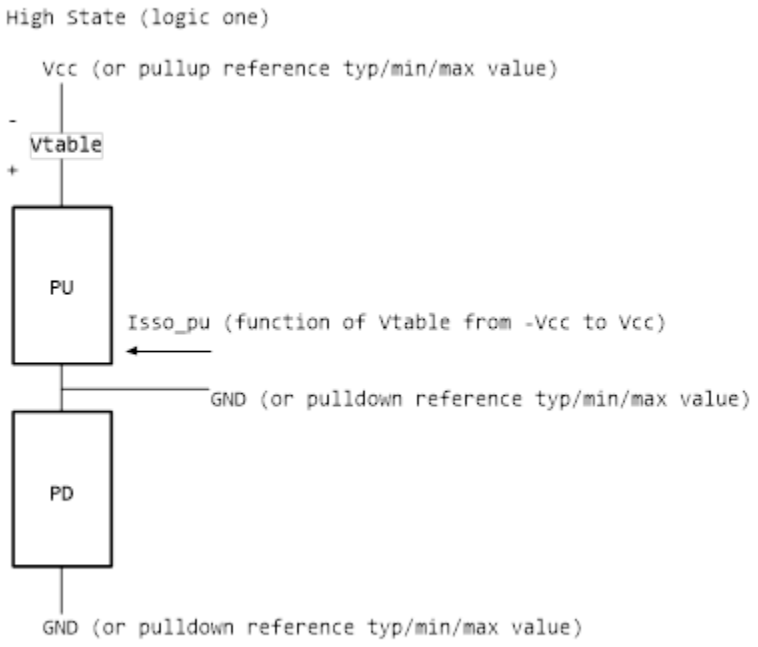


Figure 11 – High State (Logic One) Isso_pu Data Collection

For each of these extractions, the corresponding [GND Clamp] and [POWER Clamp] currents need to be removed. Normally these are negligible. However, if on-die terminators exist, the extra currents that are associated with them should be removed from the [ISSO PD] and [ISSO PU] tables. The process details are not discussed here, but need to be solved by the modeler. Such details may depend upon the contents of the [GND Clamp] and [POWER Clamp] tables and the [GND Clamp Reference] and [POWER Clamp Reference] values.

Currents are considered positive when their direction is into the component.

Other Notes: EDA tools can use such tables to calculate modulation coefficients to modulate the original pulldown and pullup currents when a voltage variation on the pullup and pulldown reference nodes is revealed during power and/or ground bounce, and/or SSO simulation events.

To describe the modulation coefficients, a reference algorithm to generate an output response producing Vout(t) for a given load including clamp currents that requires an Iout(t) is shown in terms of pullup table currents Ipu(Vcc-Vout(t)) and pulldown table currents Ipd(Vout(t)). See Figure 12.

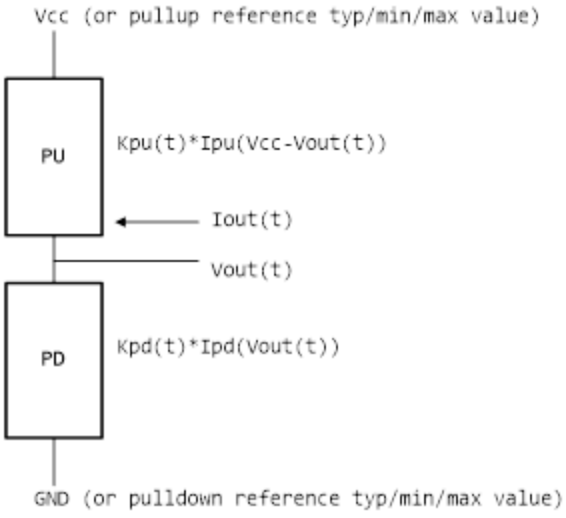


Figure 12 – Reference Data Collection

When the supplies are modulated during simulation, the modulation coefficients Ksso_pu(Vtable_pu) and Ksso_pd(Vtable_pd) modify the equations as shown in Figure 13.

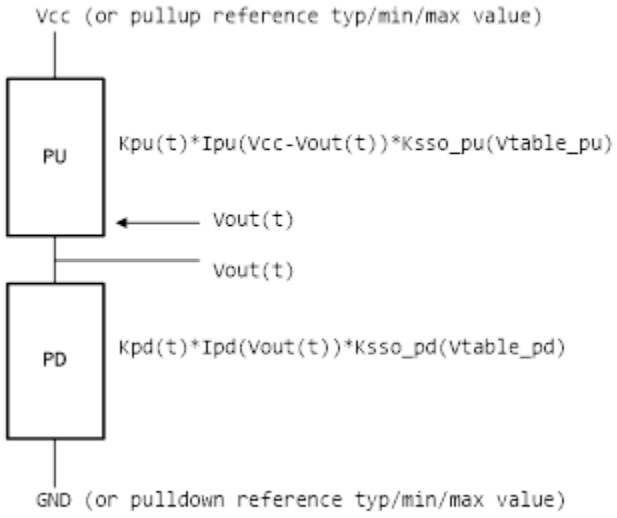


Figure 13 – Reference Data Collection with Supply Modulation

The Vtable_pd and Vtable_pu values may change at each time step. The Ksso_pd(Vtable_pd) and Ksso_pu(Vtable_pu) values are derived from the dynamic reference voltage variation and [ISSO PD] and [ISSO PU] table entries according to the equations below:

Ksso_pd(Vtable_pd) = Isso_pd(Vtable_pd)/Isso_pd(0)
Ksso_pu(Vtable_pu) = Isso_pu(Vtable_pu)/Isso_pu(0)

Note that the extraction setup equates the currents for each column at Vtable = 0 lines to the corresponding pulldown and pullup table currents:

Isso_pd(0) = Ipd(Vcc)
Isso_pu(0) = Ipu(Vcc)

where Vcc refers to the typ/min/max value used for the corresponding typ/min/max column.

For example, for a typ/min/max [Voltage Range] of 5.0V, 4.5V and 5.5V, and with the negative reference set to GND, the Isso_pu(0) and Isso_pd(0) values for typ/min/max should be equal to the column values as shown in Table 4.

Table 4 – Example of Setting Isso_pu and Isso_pd Values

	Typ	**Min**	**Max**
Isso_pd(0)	Ipd(5.0)	Ipd(4.5)	Ipd(5.5)
Isso_pu(0)	Ipu(5.0)	Ipu(4.5)	Ipu(5.5)

With no modulation, Ksso_pd(0) = 1 and Ksso_pu(0) = 1. However, if during simulation of the typical corner the Vcc voltage drops from 5.0 to 4.7, then Vtable_pu = 5.0 - 4.7 = 0.3, and Ksso_pu(0.3) is calculated. If at the same time the ground reference voltage at the buffer increases to 0.2 V, then Ksso_pd(0.2) is calculated. These two modulation factors are used in the reference model calculations to account for gate modulation effects associated with both output transistors.

These modulation factors are updated at each time step.

Note that the [ISSO PD] and [ISSO PU] keywords are designed for CMOS technology and may not be appropriate for bipolar or ECL technologies. A single [ISSO PU] or [ISSO PD] keyword table is appropriate for open technologies such as Open_drain, Open_source, Open_sink, etc.

As a minor source of error, actual modulation effects may lag slightly from simulated modulation effects due to internal delays within the physical device.

Examples:

```
| Assume [Voltage Range] is 1.8V (typ), 1.7V (min) and 1.95V (max).
|
| The table voltage entries are relative to the typ/min/max of the
| corresponding reference voltage for each table.
[ISSO PD]    |  Relative to the [Pulldown Reference] voltage
|
|  Voltage    I(typ)     I(min)     I(max)
|
   -1.8V     10.0m       7.0m       13.0m
|     .
```

```
|     .
   -0.5V     24.0m     18.0m     31.0m
   -0.2V     27.0m     20.0m     37.0m
    0.0V     25.0m     19.0m     34.0m
    0.2V     18.0m     13.0m     26.0m
    0.5V     10.0m      7.0m     16.0m
    0.7V      5.0m      3.0m      9.0m
    1.0V      1.0m      0.7m      3.0m
|     .
|     .
    1.8V      0.0m      0.0m      0.0m
|
[ISSO_PU]   |  Relative to the [Pullup Reference] voltage)
|
|  Voltage   I(typ)    I(min)    I(max)
|
   -1.8V    -10.0m     -9.0m    -14.0m
|     .
|     .
   -0.6V    -28.0m    -19.0m    -40.0m
   -0.4V    -31.0m    -22.0m    -43.0m
   -0.2V    -29.0m    -21.0m    -40.0m
    0.0V    -27.0m    -19.0m    -38.0m
    0.2V    -21.0m    -14.0m    -31.0m
    0.4V    -14.0m     -9.0m    -22.0m
|     .
|     .
    1.8V      0.0m      0.0m      0.0m
```

Keywords: **[Rgnd]**, **[Rpower]**, **[Rac]**, **[Cac]**

Required: Yes, if they exist in the design

Description: The data for these keywords define the resistance values of Rgnd and Rpower connected to GND and the POWER pins, respectively, and the resistance and capacitance values for an AC terminator. See Figure 14.

Usage Rules: For each of these keywords, the three columns hold the typical, minimum, and maximum resistance values. The three entries for R(typ), R(min), and R(max), or the three entries for C(typ), C(min), and C(max), must be placed on a single line and must be separated by at least one whitespace character. All three columns are required under these keywords. However, data are only required in the typical column. If minimum and/or maximum values are not available, the reserved word "NA" must be used indicating the R(typ) or C(typ) value by default. Note that only one instance of any one of these keywords is permitted within any single [Model]. For example, [Rgnd] may not be used twice under the same [Model] description.

Other Notes: [Rpower] is connected to "Vcc" and [Rgnd] is connected to "GND". However, [GND Clamp Reference] voltages, if defined, apply to [Rgnd]. [POWER Clamp Reference] voltages, if defined, apply to [Rpower]. Either or both [Rgnd] and [Rpower] may be defined and may coexist with [GND Clamp] and [POWER Clamp] tables. If the terminator consists of a series R and C (often referred to as either an AC or RC terminator), then both [Rac] and [Cac] are required; these two keywords shall only be used together for any given [Model]. When [Rac] and [Cac] are specified, the Model_type shall be Terminator. [Rgnd] and [Rpower] may be used with

Model_type Terminator, Model_type Input, or Model_type Input_ECL. The C_comp subparameter is required for [Model]s using any or all of these keywords; the C_comp_* subparameters may be used in the same [Model] with any or all of these keywords.

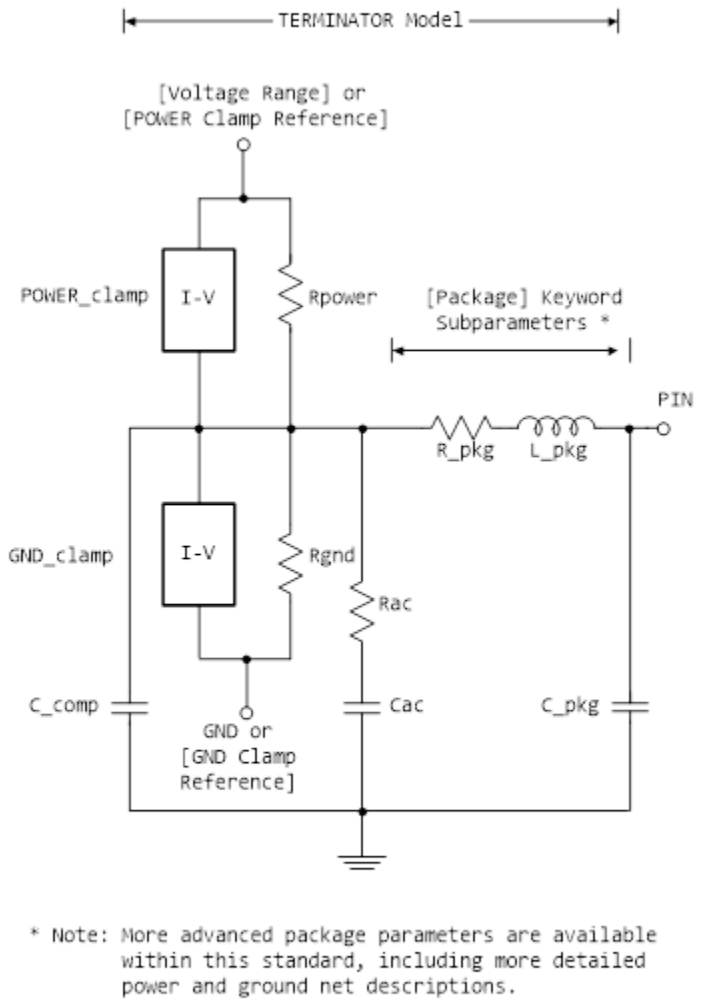


Figure 14 – [Rgnd], [Rpower], [Rac], [Cac] in Relation to Package and Buffer Data

Examples:

```
| variable      R(typ)          R(min)          R(max)
[Rgnd]          330ohm          300ohm          360ohm    | Parallel Terminator
[Rpower]        220ohm          200ohm          NA
|
[Rac]           30ohm           NA              NA
|
| variable      C(typ)          C(min)          C(max)    | AC terminator
[Cac]           50pF            NA              NA
```

Keywords: **[On]**, **[Off]**

Required: Yes, both [On] and [Off] for Series_switch Model_types only.

Description: The "On" state electrical models are positioned under [On]. The "Off" state electrical models are positioned under [Off].

Usage Rules: These keywords are only valid for Series_switch Model_types. Only keywords associated with Series_switch electrical models are permitted under [On] or [Off]. The Series electrical models describe the path for one state only and do not use the [On] and [Off] keywords.

In Series_switch models, [On] or [Off] must be positioned before any of the [R Series], [L Series], [Rl Series], [C Series], [Lc Series], [Rc Series], [Series Current], and [Series MOSFET] keywords. There is no provision for any of these keywords to be defined once, but to apply to both states.

Examples:

```
[On]
| ...  On state keywords such as [R Series], [Series Current], [Series MOSFET]
[Off]
| ...  Off state keywords such as [R Series], [Series Current]
```

Keywords: **[R Series]**, **[L Series]**, **[Rl Series]**, **[C Series]**, **[Lc Series]**, **[Rc Series]**

Required: Yes, if they exist in the design

Description: The data for these keywords allow the definition of Series or Series_switch R, L or C paths.

Usage Rules: For each of these keywords, the three columns hold the typical, minimum, and maximum resistance values. The three entries must be placed on a single line and must be separated by at least one whitespace character. All three columns are required for these keywords. However, data are only required in the typical column. If minimum and/or maximum values are not available, the reserved word "NA" must be used.

Note that only one instance of any one of these keywords is permitted within any single [On] or [Off] keyword for [Model]s of type Series_switch. For example, [L Series] may not be used twice under the same [Off] description. Similarly, only one instance of any one of these keywords is permitted within any single [Model] of type Series.

Other Notes: This series RLC model is defined to allow IBIS to model simple passive models and/or parasitics.

These keywords are valid only for Series or Series_switch Model_types.

The electrical circuit model for these keywords is shown in Figure 15.

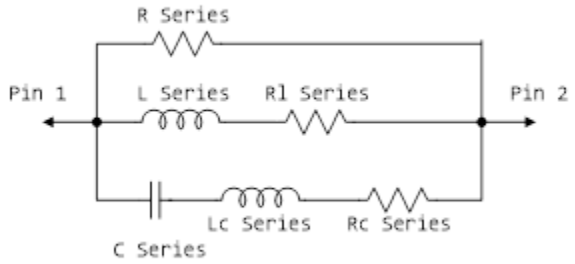


Figure 15 – Series Element Associations

[Rl Series] shall be defined only if [L Series] exists. [Rl Series] is 0 ohms if it is not defined in the path.

[Rc Series] and [Lc Series] shall be defined only if [C Series] exists. [Rc Series] is 0 ohms if it is not defined in the path. [Lc Series] is 0 henries if it is not defined in the path.

C_comp values are ignored for Series Models.

Examples:

```
| variable      R(typ)          R(min)          R(max)
[R Series]      8ohm            6ohm            12ohm
|
| variable      L(typ)          L(min)          L(max)
[L Series]      5nH             NA              NA
| variable      R(typ)          R(min)          R(max)
[Rl Series]     4ohm            NA              NA
|
| variable      C(typ)          C(min)          C(max)   | The other elements
[C Series]      50pF            NA              NA       | are 0 impedance
```

Keyword: **[Series Current]**

Required: Yes, if they exist in the design

Description: The data points under this keyword define I-V tables, with voltages measured at Pin 1 with respect to Pin 2. Currents are considered positive if they flow into Pin 1. Pins 1 and 2 are listed under the [Series Pin Mapping] keyword under columns [Series Pin Mapping] and pin_2, respectively.

Usage Rules: The first column contains the voltage value, and the remaining columns hold the typical, minimum, and maximum current values. The four entries, Voltage, I(typ), I(min), and I(max) must be placed on a single line and must be separated by at least one whitespace character.

All four columns are required under these keywords. However, data are only required in the typical column. If minimum and/or maximum current values are not available, the reserved word

"NA" must be used. "NA" can be used for currents in the typical column, but numeric values MUST be specified for the first and last voltage points on any I-V table. Each I-V table must have at least 2, but not more than 100 rows.

Other Notes: There is no monotonicity requirement. However, the model supplier should realize that it may not be possible to derive a behavioral model from non-monotonic data. This keyword is valid only for Series or Series_switch Model_types.

The model is shown in Figure 16.

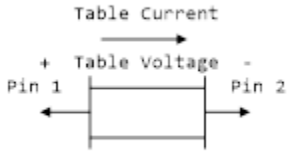


Figure 16 – [Series Current] Voltage Polarity and Current Direction

C_comp values are ignored for [Series Current] models.

Example:

```
[Series Current]
|  Voltage   I(typ)    I(min)    I(max)
   -5.0V  -3900.0m  -3800.0m  -4000.0m
   -0.7V    -80.0m    -75.0m    -85.0m
   -0.6V    -22.0m    -20.0m    -25.0m
   -0.5V     -2.4m     -2.0m     -2.9m
   -0.4V      0.0m      0.0m      0.0m
    5.0V      0.0m      0.0m      0.0m
```

Keyword: **[Series MOSFET]**

Required: Yes, for series MOSFET switches

Description: The data points under this keyword define the I-V tables for voltages measured at Pin 2 for a given Vds setting. Currents are considered positive if they flow into Pin 1. Pins 1 and 2 are listed under the [Series Pin Mapping] keyword under [Series Pin Mapping] and pin_2 columns, respectively. See Figure 17.

Sub-Params: Vds

Usage Rules: The first column contains the voltage value, and the three remaining columns hold the typical, minimum, and maximum current values. The four entries, Voltage, I(typ), I(min), and I(max) must be placed on a single line and must be separated by at least one whitespace character.

All four columns are required under this keyword. However, data are only required in the typical column. If minimum and/or maximum current values are not available, the reserved word "NA" must be used. "NA" can be used for currents in the typical column, but numeric values MUST be

specified for the first and last voltage points on any I-V table. Each I-V table must have at least 2, but not more than 100 rows.

Other Notes: There is no monotonicity requirement. However, the model supplier should realize that it may not be possible to derive a behavioral model from non-monotonic data.

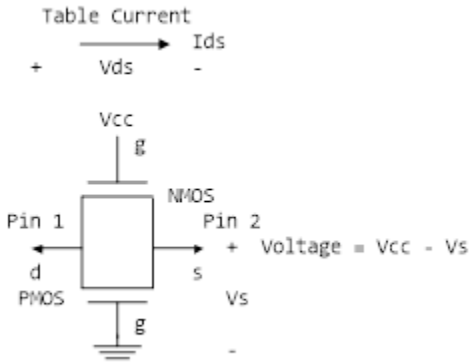


Figure 17 – [Series MOSFET] Voltage Polarities and Current Direction

Either of the FETs could be removed (or have zero current contribution). Thus, this model covers all four conditions: off, single NMOS, single PMOS, and parallel NMOS/PMOS.

Voltage = Table Voltage = Vtable = Vcc - Vs
Ids = Table Current for a given Vcc and Vds

Internal logic that is generally referenced to the power rail is used to set the NMOS MOSFET switch to its "On" state. Internal logic, likewise referenced to ground, is used to set the PMOS device to its "On" state if the PMOS device is present. Thus, the [Voltage Range] settings provide the assumed gate voltages. If the [POWER Clamp Reference] exists, it overrides the [Voltage Range] value. The table voltage entries are actually Vgs values of the NMOS device and Vcc - Vgs values of the PMOS device, if present. The polarity conventions are identical to those used for other tables that are referenced to power rails. Thus, the voltage column can be viewed as defining the source voltage Vs points according to the convention: Vtable = Vcc - Vs. This convention remains even without the NMOS device.

If the switch is used in an application such as interfacing between 3.3 V and 5.0 V logic, the Vcc may be biased at a voltage (such as 4.3 V) that is different from the power rail voltage (such as 5.0 V) used to create the model. Just readjust the [Voltage Range] entries (or [POWER Clamp Reference] entries).

One fundamental assumption in the MOSFET switch model is that it operates in a symmetrical manner. The tables and expressions are given assuming that Vd >= Vs. If Vd < Vs, then apply the same relationships under the assumption that the source and drain nodes are interchanged. A consequence of this assumption is that the Vds subparameter is constrained to values Vds > 0. It is assumed that with Vds = 0 the currents will be 0 mA. A further consequence of this assumption is

that the voltage table is based on the side of the model with the lowest voltage (and that side is defined as the source). Thus, the analysis must allow current to flow in both directions, as would occur due to reflections when the switch is connected in series with an unterminated transmission line.

The model data are used to create an On state relationship between the actual drain to source current, ids, and the actual drain to source voltage, vds:

ids = f(vds).

This functional relationship depends on the actual source voltage Vs and can be expressed in terms of the corresponding table currents associated with Vs (and expressed as a function of Vtable).

If only one [Series MOSFET] table is supplied (as a first order approximation), the functional relationship is assumed to be linearly related to the table drain to source current, Ids, for the given Vds subparameter value and located at the existing gate to source voltage value Vtable. This table current is denoted as Ids(Vtable, Vds). The functional relationship becomes:

*ids = Ids(Vtable, Vds) * vds/Vds.*

More than one [Series MOSFET] table under a [Model] keyword is permitted. However, the usage of this data is EDA tool-dependent. Each table must begin with the [Series MOSFET] keyword and Vds subparameter. Each successive [Series MOSFET] table must have a different subparameter value for Vds. The number of tables for any specific [Model] must not exceed 100.

C_comp values are ignored for [Series MOSFET] models.

Examples:

```
| An NMOS Example
|
[On]

[Series MOSFET]
Vds = 1.0
|  Voltage   I(typ)    I(min)    I(max)
    5.0V    257.9m     153.3m    399.5m    | Defines the Ids current as a
    4.0V    203.0m     119.4m    317.3m    | function of Vtable, for Vds = 1.0
    3.0V    129.8m      74.7m    205.6m
    2.0V     31.2m      16.6m     51.0m
    1.0V     52.7p      46.7p     56.7p
    0.0V      0.0p       0.0p      0.0p
|
| A PMOS/NMOS Example
|
[On]
[Series MOSFET]
Vds = 0.5
|  Voltage   I(typ)    I(min)    I(max)
    0.0      48.6ma      NA        NA
    0.1      47.7ma      NA        NA
    0.2      46.5ma      NA        NA
    0.3      46.1ma      NA        NA
    0.4      45.3ma      NA        NA
```

```
0.5      44.4ma      NA          NA
0.6      42.9ma      NA          NA
0.7      42.3ma      NA          NA
0.8      41.2ma      NA          NA
0.9      39.7ma      NA          NA
1.0      38.6ma      NA          NA
1.1      38.1ma      NA          NA
1.2      38.6ma      NA          NA
1.3      40.7ma      NA          NA
1.4      45.0ma      NA          NA
1.5      49.2ma      NA          NA
1.6      52.3ma      NA          NA
1.7      55.1ma      NA          NA
1.8      57.7ma      NA          NA
1.9      58.8ma      NA          NA
2.0      58.9ma      NA          NA
2.1      59.2ma      NA          NA
2.2      59.3ma      NA          NA
2.3      59.4ma      NA          NA
2.4      59.8ma      NA          NA
2.5      60.1ma      NA          NA
2.6      61.8ma      NA          NA
2.7      62.3ma      NA          NA
2.8      63.4ma      NA          NA
2.9      64.4ma      NA          NA
3.0      65.3ma      NA          NA
3.1      66.0ma      NA          NA
3.2      66.8ma      NA          NA
3.3      68.2ma      NA          NA
```

Keyword: **[Ramp]**

Required: Yes, except for inputs, terminators, Series, and Series_switch model types

Description: Defines the rise and fall times of a buffer. The ramp rate does not include packaging but does include the effects of the C_comp or C_comp_* parameters.

Sub-Params: dV/dt_r, dV/dt_f, R_load

Usage Rules: The rise and fall times are defined as the time taken by the output to go from 20% to 80% of the final value. The ramp rate is defined as:

$$\frac{dV}{dt} = \frac{20\%\ to\ 80\%\ voltage\ swing}{time\ it\ takes\ to\ swing\ the\ above\ voltage}$$

The ramp rate must be specified as an explicit fraction and must not be reduced. The [Ramp] values can use "NA" for the min and max values only. The R_load subparameter is optional if the default 50-ohm load is used. The R_load subparameter is required if a non-standard load is used.

Example:

```
[Ramp]
| variable      typ             min             max
dV/dt_r         2.20/1.06n      1.92/1.28n      2.49/650p
```

```
dV/dt_f          2.46/1.21n      2.21/1.54n      2.70/770p
R_load = 300ohms
```

Keywords: **[Rising Waveform]**, **[Falling Waveform]**

Required: No

Description: Describes the shape of the rising and falling edge waveforms of a driver.

Sub-Params: R_fixture, V_fixture, V_fixture_min, V_fixture_max, C_fixture, L_fixture, R_dut, L_dut, C_dut

Usage Rules: Each [Rising Waveform] and [Falling Waveform] keyword introduces a table of voltage versus time points that describe the shape of an output waveform. These voltage versus time points are taken under the conditions specified by the R/L/C/V_fixture and R/L/C_dut subparameters. The table itself consists of one column of time points, then three columns of voltage points in the standard typ, min, and max format. The four entries must be placed on a single line and must be separated by at least one whitespace character. All four columns are required. However, data are only required in the typical column. If minimum or maximum data are not available, use the reserved word "NA". The first value in the time column need not be "0". Time values must increase as one parses down the table. The waveform table can contain a maximum of 1000 data rows. A maximum of 100 waveform tables are allowed per model.

Note that for backward compatibility, the existing [Ramp] keyword is still required. The data in the waveform table is measured with the effects of the C_comp parameter included.

A waveform table must include the entire waveform; i.e., the first entry (or entries) in a voltage column must be the DC voltage of the output before switching and the last entry (or entries) of the column must be the final DC value of the output after switching. Each table must contain at least two entries. Thus, numerical values are required for the first and last entries of any column containing numerical data.

The data in all of the waveform tables should be time correlated. In other words, the edge data in each of the tables (rising and falling) should be entered with respect to a single point in time when the input stimulus is assumed to have initiated a logic transition. All waveform extractions should reference a common input stimulus time in order to provide a sufficiently accurate alignment of waveforms. The first line in each waveform table should be assumed to be the reference point in time corresponding to a logic transition. For example, assume that some internal rising edge logic transition starts at time = 0. Then a rising edge voltage-time table might be created starting at time zero. The first several table entries might be some "lead-in" time caused by some undefined internal buffer delay before the voltage actually starts transitioning. The falling edge stimulus (for the purpose of setting reference time for the voltage-time table) should also start at time = 0. And, the falling edge voltage-time table would be created starting at time zero with a possibly different amount of "lead-in" time caused by a possibly different but corresponding falling edge internal buffer delay. Any actual device differences in internal buffer delay time between rising and falling edges should appear as differing lead-in times between the rising and the falling waveforms in the tables just as any differences in actual device rise and fall times appear as differing voltage-time entries in the tables.

A [Model] specification can contain more than one rising edge or falling edge waveform table. However, each new table must begin with the appropriate keyword and subparameter list as shown

below. If more than one rising or falling edge waveform table is present, then the data in each of the respective tables must be time correlated. In other words, the rising (falling) edge data in each of the rising (falling) edge waveform tables must be entered with respect to a common reference point on the input stimulus waveform.

The "fixture" subparameters specify the loading conditions under which the waveform is taken. The R_dut, C_dut, and L_dut subparameters are analogous to the package parameters R_pkg, C_pkg, and L_pkg and are used if the waveform includes the effects of pin inductance/capacitance. Figure 18 shows the interconnection of these elements.

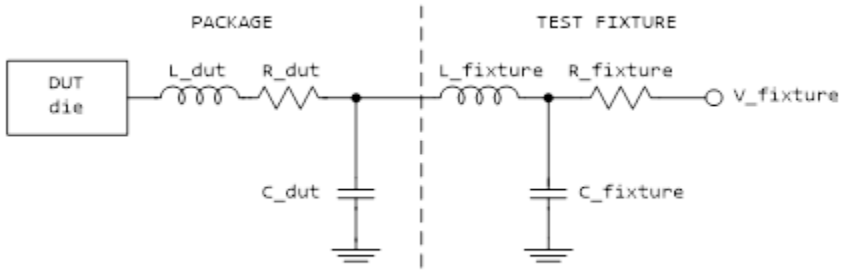


Figure 18 – [Rising Waveform] and [Falling Waveform] Fixtures

Note: The use of L_dut, R_dut, and C_dut is strongly discouraged in developing waveform data from simulation models. Some EDA tools may ignore these parameters because they may introduce numerical time constant artifacts.

Only the R_fixture and V_fixture subparameters are required; the rest of the subparameters are optional. If a subparameter is not used, its value defaults to zero. The subparameters must appear in the text after the keyword and before the first row of the waveform table.

V_fixture defines the voltage for typ, min, and max supply conditions. However, when the fixture voltage is related to the power supply voltages, then the subparameters V_fixture_min and V_fixture_max can be used to further specify the fixture voltage for min and max supply voltages.

Note: Test fixtures with R_fixture and V_fixture, V_fixture_min, and V_fixture_max only are strongly encouraged because they provide the BEST set of data needed to produce the best model for simulation. C_fixture and L_fixture can be used to produce waveforms which describe the typical test case setups for reference.

Note: In most cases two [Rising Waveform] tables and two [Falling Waveform] tables will be necessary for accurate modeling.

All tables assume that the die capacitance is included. Potential numerical problems associated with processing the data using the effective C_comp (or C_comp_* values as appropriate) for effective die capacitance may be handled differently among EDA tools.

Examples:

```
[Rising Waveform]
R_fixture = 50
V_fixture = 0.0
```

```
| C_fixture = 50p        | These are shown, but are generally not recommended
| L_fixture = 2n
| C_dut = 7p
| R_dut = 1m
| L_dut = 1n
| Time            V(typ)              V(min)              V(max)
   0.0000s       25.2100mV           15.2200mV           43.5700mV
   0.2000ns       2.3325mV           -8.5090mV           23.4150mV
   0.4000ns       0.1484V            15.9375mV            0.3944V
   0.6000ns       0.7799V             0.2673V             1.3400V
   0.8000ns       1.2960V             0.6042V             1.9490V
   1.0000ns       1.6603V             0.9256V             2.4233V
   1.2000ns       1.9460V             1.2050V             2.8130V
   1.4000ns       2.1285V             1.3725V             3.0095V
   1.6000ns       2.3415V             1.5560V             3.1265V
   1.8000ns       2.5135V             1.7015V             3.1600V
   2.0000ns       2.6460V             1.8085V             3.1695V
| ...
  10.0000ns       2.7780V             2.3600V             3.1670V
|
[Falling Waveform]
R_fixture = 50
V_fixture = 5.5
V_fixture_min = 4.5
V_fixture_max = 5.5
| Time            V(typ)              V(min)              V(max)
   0.0000s        5.0000V             4.5000V             5.5000V
   0.2000ns       4.7470V             4.4695V             4.8815V
   0.4000ns       3.9030V             4.0955V             3.5355V
   0.6000ns       2.7313V             3.4533V             1.7770V
   0.8000ns       1.8150V             2.8570V             0.8629V
   1.0000ns       1.1697V             2.3270V             0.5364V
   1.2000ns       0.7539V             1.8470V             0.4524V
   1.4000ns       0.5905V             1.5430V             0.4368V
   1.6000ns       0.4923V             1.2290V             0.4266V
   1.8000ns       0.4639V             0.9906V             0.4207V
   2.0000ns       0.4489V             0.8349V             0.4169V
| ...
  10.0000ns       0.3950V             0.4935V             0.3841V
```

Keyword: **[Composite Current]**

Required: No

Description: Describes the shape of the rising and falling edge current waveforms from the power reference terminal of the buffer.

Usage Rules: The [Composite Current] keyword is positioned under the last row of the [Rising Waveform] table (for rising waveform currents) or [Falling Waveform] table (for falling waveform currents). The keywords are followed by a table of current versus time rows (I-T) that describe the shape of a current waveform. These I-T tables inherit the test fixture load of the [Rising Waveform] or [Falling Waveform] R/L/C/V_fixture and R/L/C_dut subparameters.

The [Composite Current] keyword is optional. It can be omitted, or it can be positioned under some or all of the rising and falling waveform tables.

The table itself consists of one column of time points, then three columns of current points in the standard typ, min, and max format. The four entries must be placed on a single line and must be separated by at least one whitespace character. All four columns are required. However, data are only required in the typical column. If minimum or maximum data are not available, use the reserved word "NA". The first value in the time column need not be "0". Time values must increase as one parses down the table. The waveform table can contain a maximum of 1000 data points.

The I-T table data must be time-correlated with the V-T data above it. That is, the I-T data should be entered with respect to the same point in time that the V-T table above it references and for the given *_fixture load. See the [Rising Waveform] and [Falling Waveform] section for more information about the common input stimulus time. Note that additional "lead-in" time may need to be added to all V-T waveforms, as a portion of the I-T waveform data describes pre-driver current that may occur earlier in time than the V-T rising or falling edge transitions.

Figure 18 illustrates a general configuration from which a [Rising Waveform] or [Falling Waveform] is extracted. The DUT die shows all of the available power and ground pin reference voltage terminals. For many buffers, only one power pin and one common ground terminal are used. GND is the reference for the V_fixture voltage and the package model equivalent networks defined by [Package], [Pin] and/or [Define Package Model]. GND also serves as the reference node for C_comp, unless C_comp is optionally split into one or more C_comp_* elements which are connected to the [Model] reference rails [Pullup Reference], [POWER Clamp Reference], [Pulldown Reference] and/or [GND Clamp Reference].

The [Composite Current] I-T table includes all of the current through the [Pullup Reference] terminal. If the [POWER Clamp Reference] terminal is the same as the [Pullup Reference] terminal (according to the [Pin Mapping] keyword table), the [Composite Current] entries include the currents through both the [POWER Clamp] and [Pullup] sections of the DUT (for example, when an on-die terminator is connected to the power reference terminal). Note that the terminals are shown in terms of separately defined reference voltages, but still exist even if they are defined with default [Voltage Range] or 0 V settings.

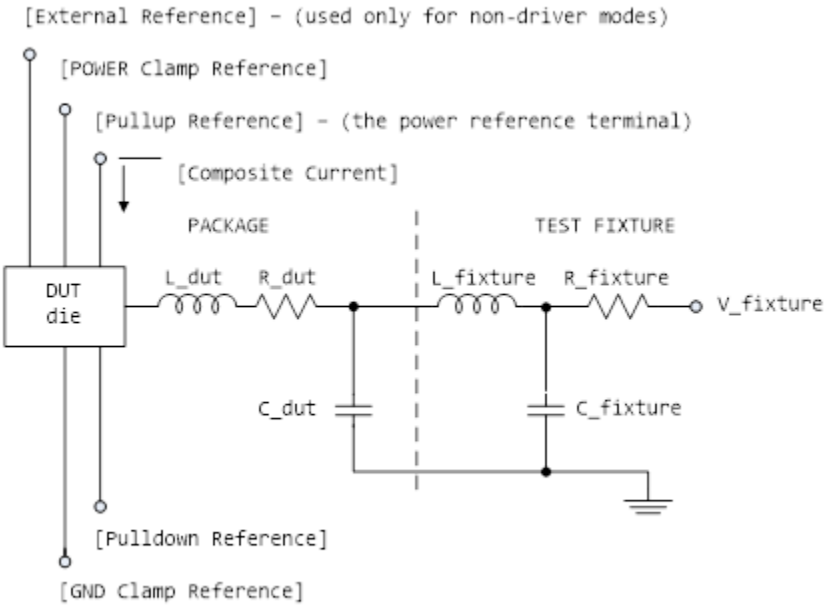


Figure 19 – [External Reference] - Used Only for Non-driver Modes

For *_ECL model types, the [Pullup] and [Pulldown] sections of the DUT share the same power reference terminal. The [Composite Current] includes the currents through both sections.

Other Notes: Figure 20 documents some expected internal paths for a useful special case where only one common power pin (VDDQ) and one common ground exists (GND).

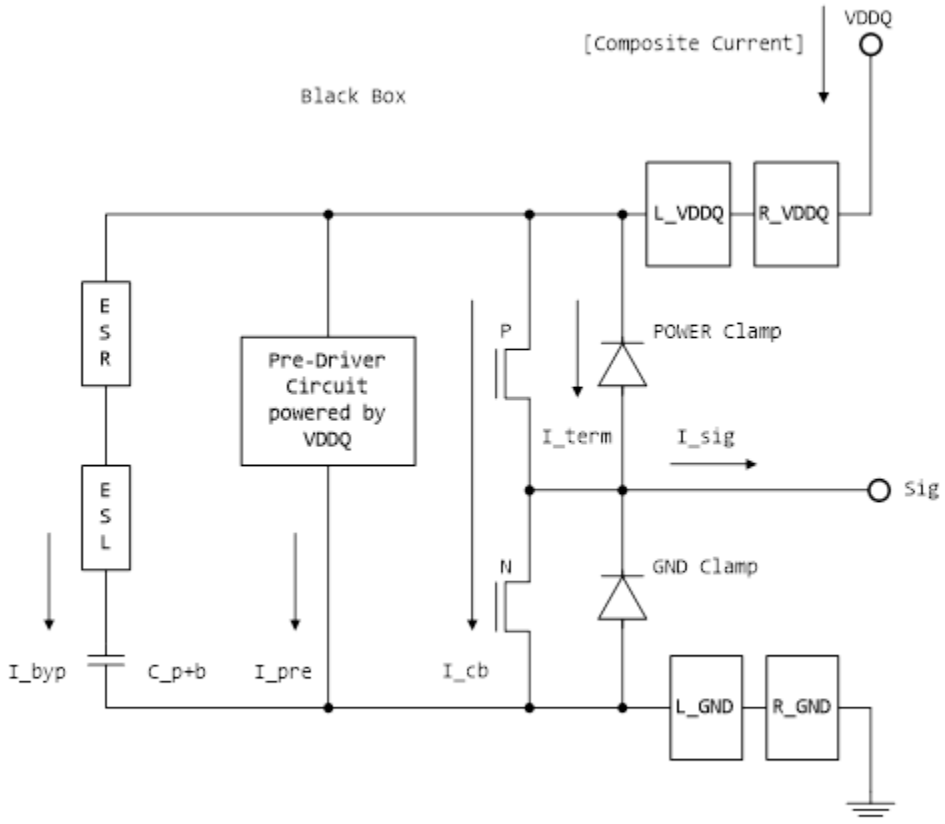


Figure 20 – [Composite Current] Internal Current Paths

Other elements in a more detailed typical (per buffer) model are:

I_byp	- Bypass current
I_pre	- Pre-Driver current
I_cb	- Crow-bar current
I_term	- Termination current (optional)
L_VDDQ	- On-die inductance of I/O Power
R_VDDQ	- On-die resistance of I/O Power
L_GND	- On-die inductance of Ground
R_GND	- On-die resistance of Ground
C_p+b	- Bypass + Parasitic Capacitance
ESR	- Equivalent Series Resistance for on-die Decap
ESL	- Equivalent Series Inductance for on-die Decap

While the [Composite Current] already includes the buffer I_byp current, some Series Model type elements may be used to document an equivalent bypass impedance to improve simulation results. Such an equivalent impedance can be extracted on a per buffer basis, but summed and expressed as a total equivalent impedance between the power and ground pins of the component with the Series Model type keywords, including [C Series], [Lc Series], [Rc Series], and [R Series] under a separate [Model]. These elements are connected using the [Series Pin Mapping] keyword. Paths between several voltage rails can be modeled in this manner. The [Pin Mapping] keyword documents which buffers share common and often isolated power rails.

The C_p+b value might include the detailed distribution of C_comp when C_comp* is attached to several rails. If the C_comp value and the C_p+b value are about the same magnitude, the [C Series] value should be adjusted to avoid double counting.

The power reference terminal (VDDQ) is usually the [Pullup Reference], or the default [Voltage Range] terminal. The [Pulldown Reference] terminal is usually at the GND connection.

The [Composite Current] can still be defined for model types without the [Pullup] keywords (such as Open_drain) because the [Pullup Reference] or [Voltage Range] are still required. Pre-driver and other internal paths still can exist.

In most cases six [Composite Current] tables are recommended for accurate modeling. The first four tables correspond to the recommended fixture conditions for [Rising Waveform] and [Falling Waveform] tables (normally 50-ohm loads to Vdd and GND). Two additional waveforms for no load conditions (such as with an R_fixture of 1.0 Megaohm) are useful. However, some EDA tools process only the first four waveforms. So, the additional open load waveforms for I-T tables should be in [Rising Waveform] and [Falling Waveform] tables that are positioned after the other V-T tables to maintain the best output response simulation accuracy.

For Open_drain and Open_source technologies, two tables are often specified (one for the [Rising Waveform] and one for the [Falling Waveform]). The tables should be positioned in front of any other optional waveform tables because some EDA tools process just the first two tables. Also, the open load tables may not yield meaningful simulations unless internal on-die terminators exist.

When the [Model] is configured for differential operation with the [Diff Pin] keyword, the individual I-T currents for each [Model] are used as an approximation, and may not accurately conform to the measured currents under actual differential operation.

The [Composite Current] table can be derived from currents measured at the [Pulldown Reference] (GND) node, but adjusted for the current flowing through the output pin and at other terminals.

The [Pin Mapping] keyword is used to document how buffers with common voltage rails are connected. The effective impedances for each buffer between the [Pullup Reference] and [Pulldown Reference] are then combined to form the total effective impedance between the voltage rails.

The [Composite Current] keyword does not accurately document the effects of controlled switching buffers such as those with [Submodel] or [Driver Schedule] keywords. The currents associated with [Submodel] switching under specified test load conditions can occur at different times under other load conditions. The scheduled models under the [Driver Schedule] keyword can be attached to different voltage rails in an undocumented manner.

Example:

```
[Rising Waveform]
R_fixture = 50.0
V_fixture = 0.0
| ...
| ...             | Rising Waveform table
| ...
[Composite Current]
|
| Time       I(typ)    I(min)   I(max)
0            4.243E-05   NA      NA
4.00E-11     4.244E-05   NA      NA
8.00E-11     4.242E-05   NA      NA
1.20E-10     4.265E-05   NA      NA
1.60E-10     3.610E-05   NA      NA
2.00E-10     3.903E-03   NA      NA
..
..
..
3.80E-09     2.012E-02   NA      NA
3.84E-09     2.012E-02   NA      NA
3.88E-09     2.012E-02   NA      NA
3.92E-09     2.012E-02   NA      NA
3.96E-09     2.012E-02   NA      NA
4.00E-09     2.012E-02   NA      NA
|
[Falling Waveform]
R_fixture = 50.0
V_fixture = 1.8
| ...
| ...             | Falling Waveform table
| ...
[Composite Current]
|
| Time       I(typ)    I(min)   I(max)
0            4.302E-05   NA      NA
4.00E-11     4.299E-05   NA      NA
8.00E-11     4.304E-05   NA      NA
1.20E-10     4.287E-05   NA      NA
1.60E-10     4.782E-05   NA      NA
2.00E-10     1.459E-04   NA      NA
```

```
..
..
..
3.80E-09    4.933E-05   NA      NA
3.84E-09    5.211E-05   NA      NA
3.88E-09    5.490E-05   NA      NA
3.92E-09    5.441E-05   NA      NA
3.96E-09    4.842E-05   NA      NA
4.00E-09    4.244E-05   NA      NA
|
| ... etc.
```

Keyword: **[Initial Delay]**

Required: No

Description: Initial delay added to waveform tables.

Sub-Params: V-T, I-T

Usage Rules: The [Initial Delay] keyword can be specified only when at least one waveform table ([Rising Waveform], [Falling Waveform], or [Composite Current]) is included for the IBIS model under which this keyword is specified. The V-T subparameter can be specified only when at least one voltage waveform table ([Rising Waveform] or [Falling Waveform]) is specified. It applies to all the voltage waveforms present but only within the IBIS [Model] under which this keyword is specified. The I-T subparameter can be specified only when at least one current waveform table ([Composite Current]) is specified. It applies to all of the current waveforms present, but only within the IBIS [Model] under which this keyword is specified. Only one [Initial Delay] keyword can be specified for a model. It shall be followed by either one subparameter or both subparameters. Data specified for a subparameter shall be handled as described below.

Other Notes: Each subparameter shall be followed by three non-negative floating-point numbers representing the typ, min and max amounts of time delay in seconds. For the second and/or the third number NA can be specified. The meaning of the "NA" entry is equivalent to entering the same value as in the "typ" column (the first value following the subparameter name). The typ, min and max values are defined for the respective typ, min and max columns of the corresponding waveform tables.

EDA tool handling of [Initial Delay] data: Following user selection of the typ/min/max corner the EDA tool shall apply the value τ_V given in the corresponding column of the V-T subparameter to (1) modifying all the voltage waveform tables by subtracting τ_V from each value in the "time" column of the table, and (2) delaying every trigger event applied to the buffer by τ_V when any of the voltage waveform tables is to be used. Following user selection of the typ/min/max corner, the EDA tool shall apply the value τ_{CC} given in the corresponding column of the I-T subparameter to (1) modifying all the current waveform tables by subtracting τ_{CC} from each value in the "time" column of the table, and (2) delaying every trigger event applied to the buffer by τ_{CC} when any of the current waveform tables is to be used. When both subparameters are specified and the values τ_V and τ_{CC} are identical any single trigger event applied to the buffer becomes simply delayed. When both subparameters are specified and the values τ_V and τ_{CC} are different, any single trigger event applied to the buffer is split to create two separately delayed trigger events that are applied independently to either the voltage waveform data or to the current waveform data. For IBIS files

with [IBIS Ver] 6.1 or higher, if the delay information is missing (the [Initial Delay] keyword is not specified or when it is specified the subparameter(s) corresponding to the table(s) that is/are present is/are not specified), the assumption is that the corresponding delay value(s) (τ_V and/or τ_{CC}) is/are zero.

Example:

```
[Initial Delay] | This keyword provides information on removable delay(s)
| time table      typ         min         max
  V-T             0.20e-9     0.22e-9     0.18e-9
  I-T             0.05e-9     NA          NA
```

6.2 ADD SUBMODEL DESCRIPTION

The [Add Submodel] keyword can be used under a top-level [Model] keyword to add special-purpose functionality to the existing top-level model. This section describes the structure of the top-level model and the submodel.

Top-Level Model:

When special-purpose functional detail is needed, the top-level model can call one or more submodels. The [Add Submodel] keyword is positioned after the initial set of required and optional subparameters of the [Model] keyword and among the keywords under [Model].

The [Add Submodel] keyword lists the name of each submodel and the permitted mode (Driving, Non-Driving or All) under which each added submodel is used.

Submodel:

A submodel is defined using the [Submodel] keyword. It contains a subset of keywords and subparameters used for the [Model] keyword along with other keywords and subparameters that are needed for the added functionality.

The [Submodel] and [Submodel Spec] keywords are defined first since they are used for all submodels.

The only required subparameter in [Submodel] is Submodel_type to define the list of submodel types. No subparameters under [Model] are permitted under the [Submodel] keyword.

The following keywords that are defined under the [Model] keyword are supported by the [Submodel] keyword:

[Pulldown]
[Pullup]
[GND Clamp]
[POWER Clamp]
[Ramp]
[Rising Waveform]
[Falling Waveform]
[Initial Delay]

The [Voltage Range], [Pullup Reference], [Pulldown Reference], [GND Clamp Reference], and [POWER Clamp Reference] keywords are not permitted. The voltage settings are inherited from the top-level model. The following additional keywords are used only for the [Submodel] and are documented in this section:

[Submodel Spec]
[GND Pulse Table]
[POWER Pulse Table]

The application of these keywords depends upon the Submodel_type entries listed below:

Dynamic_clamp
Bus_hold
Fall_back

Permitted keywords that are not defined for any of these submodel types are ignored. The rules for what set of keywords are required are found under the Dynamic Clamp, Bus Hold, and Fall Back headings of this section.

Keyword: **[Submodel]**

Required: No

Description: Used to define a submodel, and its attributes.

Sub-Params: Submodel_type

Usage Rules: Each submodel must begin with the keyword [Submodel]. The submodel name shall match one that is listed under an [Add Submodel] keyword and must not contain more than 20 characters. A .ibs file must contain enough [Submodel] keywords to cover all of the model names specified under the [Add Submodel] keyword.

The Submodel_type subparameter is required and must be one of the following: Dynamic_clamp, Bus_hold, Fall_back.

The C_comp subparameter is not permitted under the [Submodel] keyword. The total effective die capacitance including the submodel contributions is provided in the top-level model.

Other Notes: The following list of keywords that are defined under the [Model] keyword can be used under [Submodel]:

[Pulldown]

[Pullup]

[GND Clamp]

[POWER Clamp]

[Ramp]

[Rising Waveform]

[Falling Waveform]

[Initial Delay]

The following list of additional keywords can be used:

[Submodel Spec]

[GND Pulse Table]

[POWER Pulse Table]

Example:

```
[Submodel]       Dynamic_clamp1
Submodel_type    Dynamic_clamp
```

Keyword: **[Submodel Spec]**

Required: No

Description: The [Submodel Spec] keyword defines four columns under which specification and information subparameters are defined for submodels.

Sub-Params: V_trigger_r, V_trigger_f, Off_delay

Usage Rules: The [Submodel Spec] is to be used only with submodels.

The following subparameters are used:

V_trigger_r	Rising edge trigger voltage
V_trigger_f	Falling edge trigger voltage
Off_delay	Turn-off delay from V_trigger_r or V_trigger_f

For each subparameter contained in the first column, the remaining three hold its typical, minimum and maximum values. The entries of typical, minimum, and maximum must be placed on a single line and must be separated by at least one whitespace character. All four columns are required under the [Submodel Spec] keyword. However, data are required only in the typical column. If minimum and/or maximum values are not available, the reserved word "NA" must be used to indicate the typical value by default.

The values in the minimum and maximum columns usually correspond to the values in the same columns for the inherited top-level voltage range or reference voltages in the top-level model. The V_trigger_r and V_trigger_f subparameters should hold values in the minimum and maximum columns that correspond to the voltage range or reference voltages of the top-level model. The Off_delay subparameter, however, is an exception to this rule because in some cases it may be completely or partially independent from supply voltages and/or manufacturing process variations. Therefore, the minimum and maximum entries for the Off_delay subparameter should be ordered simply by their magnitude.

Unless otherwise noted, each [Submodel Spec] subparameter is independent of any other subparameter.

V_trigger_r, V_trigger_f rules:

The voltage trigger values for the rising and falling edges provide the starting time when an action is initiated.

Off_delay rules:

The functionality of the Off_delay subparameter is to provide an additional time related mechanism to turn off circuit elements.

Examples:

```
| Dynamic Clamp Example:
|
[Submodel Spec]
|   Subparameter          typ        min        max
|
V_trigger_r               3.6        2.9        4.3 | Starts power pulse table
V_trigger_f               1.4        1.2        1.6 | Starts gnd pulse table
|
| Bus Hold Example:
|
[Submodel Spec]
```

```
|   Subparameter          typ        min        max
V_trigger_r               3.1        2.4        3.7 | Starts low to high
                                                    | bus hold transition
V_trigger_f               1.8        1.6        2.0 | Starts high to low
                                                    | bus hold transition
|
| Bus_hold application with pullup structure triggered on and then clocked
| off:
|
[Submodel Spec]
|   Subparameter          typ        min        max
V_trigger_r               3.1        2.4        3.7 | Low to high transition
                                                    | triggers the turn on
                                                    | process of the pullup
V_trigger_f             -10.0      -10.0      -10.0 | Not used, so trigger
                                                    | voltages are set out
                                                    | of range
Off_delay                 5n         4n         6n  | Time from rising edge
                                                    | trigger at which the
                                                    | pullup turned off
```

Dynamic Clamp:

When the Submodel_type subparameter under the [Submodel] keyword is set to Dynamic_clamp, the submodel describes the dynamic clamp functionality.

The [GND Pulse Table] and [POWER Pulse Table] keywords are defined below.

Keywords: **[GND Pulse Table]**, **[POWER Pulse Table]**

Required: No

Description: Used to specify the offset voltage versus time of [GND Clamp] and [POWER Clamp] tables within submodels.

Usage Rules: Each [GND Pulse Table] and [POWER Pulse Table] keyword introduces a table of voltage vs. time points that describe the shape of an offset voltage from the [GND Clamp Reference] voltage (or default ground) or the [POWER Clamp Reference] voltage (or default [Voltage Range] voltage). Note that these voltage values are inherited from the top-level model.

The table itself consists of one column of time points, then three columns of voltage points in the standard typ, min, and max format. The four entries must be placed on a single line and must be separated by at least one whitespace character. All four columns are required. However, data are only required in the typical column. If minimum or maximum data are not available, use the reserved word "NA". Time values must increase as one parses down the table. The waveform table can contain a maximum of 100 rows.

Each table must contain at least two entries. Thus, numerical values are required for the first and last entries of any column containing numerical data.

The voltage entries in both the [GND Pulse Table] and [POWER Pulse Table] tables are directly measured offsets. At each instance, the [GND Pulse Table] voltage is ADDED to the [GND Clamp] table voltages to provide the shifted table voltages. At each instance, the [POWER Pulse

Table] voltage is SUBTRACTED (because of polarity conventions) from the [POWER Clamp] table voltages to provide the shifted table voltages.

Only one [GND Pulse Table] and one [POWER Pulse Table] are allowed per model.

The [GND Pulse Table] and [POWER Pulse Table] interact with [Submodel Spec] subparameters V_trigger_f and V_trigger_r. Several modes of operation exist based on whether a pulse table and its corresponding trigger subparameter are given. These modes are classified as triggered and static.

Triggered Mode:

For triggered mode, a pulse table must exist and include the entire waveform; i.e., the first entry (or entries) in a voltage column must be equal to the last entry.

Also, a corresponding [Submodel Spec] V_trigger_* subparameter must exist. The triggered interaction is described:

The V_trigger_f subparameter under [Submodel Spec] is used to detect when the falling edge waveform at the buffer pad on the die passes the trigger voltage. At that time, the [GND Pulse Table] operation starts. Similarly, the V_trigger_r subparameter is used to detect when the rising edge waveform at the buffer pad on the die passes the trigger voltage. At that time, [POWER Pulse Table] operation starts. The [GND Pulse Table] dependency is shown in Figure 21.

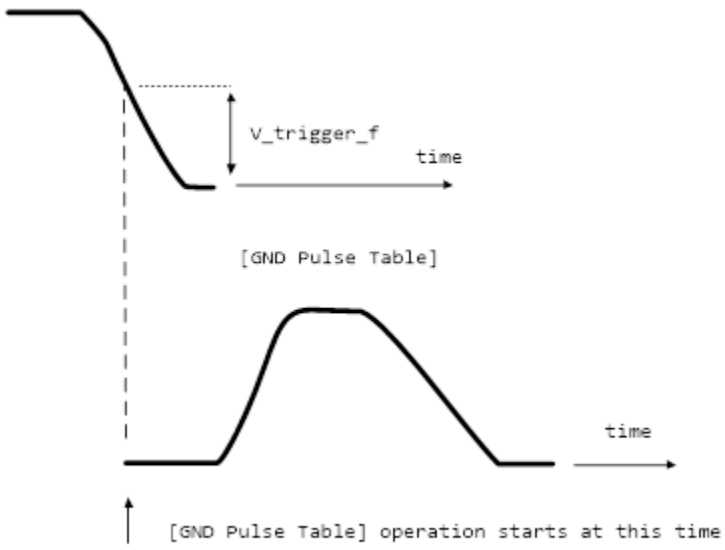


Figure 21 – [GND Pulse Table] Waveforms at Die

The V_trigger_r and [POWER Pulse Table] operate in a similar manner. When the V_trigger_r voltage value is reached on the rising edge, the [POWER Pulse Table] is started. Normally the offset voltage entries in the [POWER Pulse Table] are negative.

Static Mode:

When the [GND Pulse Table] keyword does not exist, but the added model [GND Clamp] table does exist, the added model [GND Clamp] is used directly. Similarly, when the [POWER Pulse

Table] keyword does not exist, but the added model [POWER Clamp] table does exist, the added model [POWER Clamp] is used directly.

This mode provides additional fixed clamping to an I/O_* buffer or a 3-state buffer when it is used as a driver.

Examples:

```
| Dynamic_clamp Model with both dynamic GND and POWER clamps
|
[Submodel]       Dynamic_Clamp_1
Submodel_type    Dynamic_clamp
|
[Submodel Spec]
|   Subparameter          typ        min        max
|
V_trigger_f               1.4        1.2        1.6  | Falling edge trigger
V_trigger_r               3.6        2.9        4.3  | Rising edge trigger
|
|                         typ        min        max
| [Voltage Range]           5.0        4.5        5.5
| Note, the actual voltage range and reference voltages are inherited from
| the top-level model.
|
[GND Pulse Table]                                     | GND Clamp offset table
|
|    Time          V(typ)       V(min)        V(max)
|
       0             0            0             0
    1e-9             0            0             0
    2e-9           0.9          0.8           1.0
   10e-9           0.9          0.8           1.0
   11e-9             0            0             0
|
[GND Clamp]                                           | Table to be offset
|
|  Voltage        I(typ)       I(min)        I(max)
|
    -5.000     -3.300e+01    -3.000e+01    -3.500e+01
    -4.000     -2.300e+01    -2.200e+01    -2.400e+01
    -3.000     -1.300e+01    -1.200e+01    -1.400e+01
    -2.000     -3.000e+00    -2.300e+00    -3.700e+00
    -1.900     -2.100e+00    -1.500e+00    -2.800e+00
    -1.800     -1.300e+00    -8.600e-01    -1.900e+00
    -1.700     -6.800e-01    -4.000e-01    -1.100e+00
    -1.600     -2.800e-01    -1.800e-01    -5.100e-01
    -1.500     -1.200e-01    -9.800e-02    -1.800e-01
    -1.400     -7.500e-02    -7.100e-02    -8.300e-02
    -1.300     -5.750e-02    -5.700e-02    -5.900e-02
    -1.200     -4.600e-02    -4.650e-02    -4.550e-02
    -1.100     -3.550e-02    -3.700e-02    -3.450e-02
    -1.000     -2.650e-02    -2.850e-02    -2.500e-02
    -0.900     -1.850e-02    -2.100e-02    -1.650e-02
    -0.800     -1.200e-02    -1.400e-02    -9.750e-03
    -0.700     -6.700e-03    -8.800e-03    -4.700e-03
    -0.600     -3.000e-03    -4.650e-03    -1.600e-03
    -0.500     -9.450e-04    -1.950e-03    -3.650e-04
```

```
    -0.400     -5.700e-05    -2.700e-04    -5.550e-06
    -0.300     -1.200e-06    -1.200e-05    -5.500e-08
    -0.200     -3.000e-08    -5.000e-07     0.000e+00
    -0.100      0.000e+00     0.000e+00     0.000e+00
     0.000      0.000e+00     0.000e+00     0.000e+00
     5.000      0.000e+00     0.000e+00     0.000e+00
|
[POWER Pulse Table]                                  | POWER Clamp offset table
|
|    Time          V(typ)       V(min)        V(max)
|
       0             0            0             0
    1e-9             0            0             0
    2e-9          -0.9         -1.0          -0.8
   10e-9          -0.9         -1.0          -0.8
   11e-9             0            0             0
|
[POWER Clamp]                                        | Table to be offset
|
|  Voltage        I(typ)        I(min)        I(max)
|
    -5.000      1.150e+01     1.100e+01     1.150e+01
    -4.000      7.800e+00     7.500e+00     8.150e+00
    -3.000      4.350e+00     4.100e+00     4.700e+00
    -2.000      1.100e+00     8.750e-01     1.300e+00
    -1.900      8.000e-01     6.050e-01     1.000e+00
    -1.800      5.300e-01     3.700e-01     7.250e-01
    -1.700      2.900e-01     1.800e-01     4.500e-01
    -1.600      1.200e-01     6.850e-02     2.200e-01
    -1.500      3.650e-02     2.400e-02     6.900e-02
    -1.400      1.200e-02     1.100e-02     1.600e-02
    -1.300      6.300e-03     6.650e-03     6.100e-03
    -1.200      4.200e-03     4.750e-03     3.650e-03
    -1.100      2.900e-03     3.500e-03     2.350e-03
    -1.000      1.900e-03     2.450e-03     1.400e-03
    -0.900      1.150e-03     1.600e-03     7.100e-04
    -0.800      5.500e-04     9.150e-04     2.600e-04
    -0.700      1.200e-04     4.400e-04     5.600e-05
    -0.600      5.400e-05     1.550e-04     1.200e-05
    -0.500      1.350e-05     5.400e-05     1.300e-06
    -0.400      8.650e-07     7.450e-06     4.950e-08
    -0.300      6.250e-08     7.550e-07     0.000e+00
    -0.200      0.000e+00     8.400e-08     0.000e+00
    -0.100      0.000e+00     0.000e-08     0.000e+00
     0.000      0.000e+00     0.000e+00     0.000e+00
|
```

Bus Hold:

When the Submodel_type subparameter under the [Submodel] keyword is set to Bus_hold, the added model describes the bus hold functionality. However, while described in terms of bus hold functionality, active terminators can also be modeled.

Existing keywords and subparameters are used to describe bus hold models. The [Pullup] and [Pulldown] tables both are used to define an internal buffer that is triggered to switch to its opposite state. This switching transition is specified by a [Ramp] keyword or by the [Rising Waveform] and

[Falling Waveform] keywords. The usage rules for these keywords are the same as under the [Model] keyword. In particular, at least either the [Pullup] or [Pulldown] keyword is required. Also, the [Ramp] keyword is required, even if the [Rising Waveform] and [Falling Waveform] tables exist. However, the voltage ranges and reference voltages are inherited from the top-level model.

For bus hold submodels, the [Submodel Spec] keyword, V_trigger_r, and V_trigger_f are required. The Off_delay subparameter is optional, and can only be used if the submodel consists of a pullup or a pulldown structure only, and not both. Devices which have both pullup and pulldown structures controlled in this fashion can be modeled using two submodels, one for each half of the circuit.

The transition is triggered by action at the buffer pad on the die using the [Submodel Spec] V_trigger_r and V_trigger_f subparameters as described next. In all subsequent discussions, "low" means the pulldown structure is on or active, and the pullup structure is off or inactive if either or both exist. The opposite settings are referred to as "high".

If the starting buffer pad die voltage (Vdie) is below V_trigger_f, then the bus hold model is set to the low state causing additional pulldown current. If the starting voltage is above V_trigger_r, the bus hold model is set to the high state for additional pullup current.

Under some unusual cases, the above conditions can be both met or not met at all. To resolve this, the EDA tool should compute the starting voltage with the bus hold model set to low. If the starting voltage is equal to or less than the average of V_trigger_r and V_trigger_f, keep the bus hold model in the low state. Otherwise, set the bus hold model to the high state.

When the input passes through V_trigger_f during a high-to-low transition at the die, the bus hold output switches to the low state. Similarly, when the input passes though V_trigger_r during a low-to-high transition at the die, the bus hold output switches to the high state.

If the bus hold submodel has a pullup structure only, V_trigger_r provides the time when its pullup is turned on and V_trigger_f or Off_delay provides the time when it is turned off, whichever occurs first. Similarly, if the submodel has a pulldown structure only, V_trigger_f provides the time when its pulldown is turned on and V_trigger_r or Off_delay provides the time when it is turned off, whichever occurs first. The required V_trigger_r and V_trigger_f voltage entries can be set to values outside of the input signal range if the pullup or pulldown structures are to be held on until the Off_delay turns them off.

The starting mode for each of the submodels which include the Off_delay subparameter of the [Submodel Spec] keyword is the off state. Also, while two submodels provide the desired operation, either of the submodels may exist without the other to simulate turning on and off only a pullup or a pulldown current.

Table 5 through Table 8 summarize the bus hold initializations and switching transitions:

Table 5 – Bus Hold without Off_Delay – Initialization

Initial Vdie Value	**Initial Bus Hold Submodel State**
<= V_trigger_r & < V_trigger_f	low
=> V_trigger_f & > V_trigger_r	high

Initial Vdie Value	Initial Bus Hold Submodel State
Recommendations if neither or both conditions above are satisfied	
<= (V_trigger_f + V_trigger_r)/2	low
> (V_trigger_f + V_trigger_r)/2	high

Table 6 – Bus Hold without Off_Delay – Transitions

Prior Bus Hold Submodel State	Vdie transition through V_trigger_r/f	Bus Hold Transition
Low	V_trigger_r	low-to-high
Low	V_trigger_f	no change
High	V_trigger_r	no change
High	V_trigger_f	high-to-low

Table 7 – Bus Hold with Off_Delay – Initialization

[Pullup] or [Pulldown] Table	Initial Bus Hold Submodel State (Off Mode)
[Pullup]	low
[Pulldown]	high
Notes: 1) Requires [Pulldown] or [Pullup] only.	

Table 8 – Bus Hold with Off_Delay – Transitions

Prior Bus Hold Submodel State	Vdie transition through V_trigger_r/f	Bus Hold Transition	Off_delay Transition
Low	V_trigger_r	low-to-high	high-to-low
Low	V_trigger_f	no change	no change
High	V_trigger_r	no change	no change
High	V_trigger_f	high-to-low	low-to-high

Prior Bus Hold Submodel State	Vdie transition through V_trigger_r/f	Bus Hold Transition	Off_delay Transition
Notes: 1) If Vdie passes again through the V_trigger_r/f thresholds before the Off_delay time is reached, the bus hold state follows the change documented in the first table, overriding the Off_delay transition. Requires [Pulldown] or [Pullup] only.			

No additional keywords are needed for this functionality.

Examples:

```
| Complete Bus Hold Model Example:
|
[Submodel]       Bus_hold_1
Submodel_type    Bus_hold
|
[Submodel Spec]
|   Subparameter          typ        min        max
|
V_trigger_f               1.3        1.2        1.4  | Falling edge trigger
V_trigger_r               3.1        2.6        4.6  | Rising edge trigger
|
|                         typ        min        max
| [Voltage Range]         5.0        4.5        5.5
| Note, the actual voltage range and reference voltages are inherited from
| the top-level model.
|
[Pulldown]
|
-5V     -100uA     -80uA     -120uA
-1V     -30uA      -25uA     -40uA
0V       0           0         0
1V       30uA       25uA      40uA
3V       50uA       45uA      50uA
5V       100uA      80uA      120uA
10v      120uA      90uA      150uA
|
[Pullup]
|
-5V      100uA      80uA      120uA
-1V      30uA       25uA      40uA
0V       0           0         0
1V      -30uA      -25uA     -40uA
3V      -50uA      -45uA     -50uA
5V      -100uA     -80uA     -120uA
10v     -120uA     -90uA     -150uA
|
|-------------------------------------------------------------------------------
|
[Ramp]
|                       typ             min             max
dV/dt_r                 2.0/0.50n       2.0/0.75n       2.0/0.35n
```

```
dV/dt_f                 2.0/0.50n       2.0/0.75n       2.0/0.35n
R_load = 500
|
|-------------------------------------------------------------------------
| Complete Pulldown Timed Latch Example:
|
[Submodel]       Timed_pulldown_latch
Submodel_type    Bus_hold
|
[Submodel Spec]
|   Subparameter          typ        min        max
|
V_trigger_r               3.1        2.6        4.6  | Rising edge trigger
                                                     | Values could be set out
                                                     | of range to disable the
                                                     | trigger
V_trigger_f               1.3        1.2        1.4  | Falling edge trigger



Off_delay                 3n         2n         5n   | Delay to turn off the
                                                     | pulldown table
|
| Note that if the input signal goes above the V_trigger_r value, the
| pulldown structure will turn off even if the timer didn't expire yet.
|
|                         typ        min        max
| [Voltage Range]         5.0        4.5        5.5
| Note, the actual voltage range and reference voltages are inherited from
| the top-level model.
|
[Pulldown]
|
-5V     -100uA     -80uA     -120uA
-1V      -30uA     -25uA     -40uA
0V       0           0         0
1V       30uA       25uA      40uA
3V       50uA       45uA      50uA
5V       100uA      80uA      120uA
10v      120uA      90uA     150uA
|
| [Pullup] table is omitted to signal Open_drain functionality.
|
|-------------------------------------------------------------------------
|
[Ramp]
|                       typ             min             max
dV/dt_r                 2.0/0.50n       2.0/0.75n       2.0/0.35n
dV/dt_f                 2.0/0.50n       2.0/0.75n       2.0/0.35n
R_load = 500
|
|=========================================================================
|
```

Fall Back:

When the Submodel_type subparameter under the [Submodel] keyword is set to Fall_back, the added model describes the fall back functionality. This submodel can be used to model drivers that reduce their strengths and increase their output impedances during their transitions. The fall back submodel is specified in a restrictive manner consistent with its intended use with a driver model operating only in Driving mode. In a Non-Driving mode, no action is specified. For example, a fall back submodel added to an Input or Terminator model would be inactive.

Existing keywords and subparameters are used to describe fall back models. However, only one [Pullup] or [Pulldown] table, but not both, is allowed. The switching transition is specified by a [Ramp] keyword or by the [Rising Waveform] and [Falling Waveform] keywords. The [Ramp] keyword is required, even if the [Rising Waveform] and [Falling Waveform] tables exist. However, the voltage ranges and reference voltages are inherited from the top-level model.

For fall back submodels, the [Submodel Spec] subparameters V_trigger_r, and V_trigger_f are required. Unlike the bus hold model, the Off_delay subparameter is not permitted. Devices which have both pullup and pulldown structures can be modeled using two submodels, one for the rising cycle and one for the falling cycle.

In all following discussion, “low” means the pulldown structure is on or active, and the pullup structure is off or inactive. The opposite settings are referred to as “high”.

The transition is triggered by action at the die using the [Submodel Spec] V_trigger_r and V_trigger_f subparameters. The initialization and transitions are shown in Table 9 through Table 11.

Table 9 – Fall Back, Initial State

[Pullup] or [Pulldown] Table	**Initial Fall Back Submodel State (Off Mode)**
[Pullup]	low
[Pulldown]	high

Table 10 – Fall Back, Driver Rising Cycle

Prior State	**Vdie**	**Rising Edge Transition**	**Vdie > V_trigger_r Transition**
Low	<= V_trigger_r	low-to-high	high-to-low
	> V_trigger_r	stays low	stays low
High	<= V_trigger_r	stays high	high-to-low
	> V_trigger_r	stays high	stays high

Table 11 – Fall Back, Driver Falling Cycle

Prior State	Vdie	Falling Edge Transition	Vdie < V_trigger_f Transition
High	=> V_trigger_f	high-to-low	low-to-high
	< V_trigger_f	stays high	stays high
Low	=> V_trigger_f	stays low	low-to-high
	< V_trigger_f	stays low	stays low

One application is to configure the submodel with only a pullup structure. At the beginning of the rising edge cycle, the pullup is turned on to the high state. When the die voltage passes V_trigger_r, the pullup structure is turned off. Because only the pullup structure is used, the off state is low corresponding to a high-Z state. During the falling transition, the pullup remains in the high-Z state if the V_trigger_f is set out of range to avoid setting the submodel to the high state. So, a temporary boost in drive occurs only during the first part of the rising cycle.

A similar submodel consisting of only a pulldown structure could be constructed to provide added drive strength only at the beginning of the falling cycle. The complete IBIS model would have both submodels to give added drive strength for both the start of the rising and the start of the falling cycles.

No additional keywords are needed for this functionality.

Examples:

```
| Complete Dynamic Output Model Example Using Two Submodels:
|
[Submodel]       Dynamic_Output_r
Submodel_type    Fall_back
|
[Submodel Spec]
|   Subparameter          typ        min        max
|
V_trigger_f            -10.0      -10.0      -10.0  | Falling edge trigger
                                                    | set out of range to
                                                    | disable trigger
V_trigger_r              3.1        2.6        4.6  | Rising edge trigger
|
|                          typ        min        max
| [Voltage Range]          5.0        4.5        5.5
| Note, the actual voltage range and reference voltages are inherited from
| the top-level model.
|
[Pullup]
|
-5V      100mA      80mA      120mA
0V       0           0         0
10v     -200mA     -160mA    -240mA
|
| [Pulldown] table is omitted to signify Open_source functionality.
|
```

```
|-------------------------------------------------------------------------
|
[Ramp]
|                       typ             min             max
dV/dt_r                 1.5/0.50n       1.43/0.75n      1.58/0.35n
dV/dt_f                 1.5/0.50n       1.43/0.75n      1.58/0.35n
R_load = 50
|
|-------------------------------------------------------------------------
[Submodel]       Dynamic_Output_f
Submodel_type    Fall_back
|
[Submodel Spec]
|   Subparameter          typ        min        max
|
V_trigger_r               10.0       10.0       10.0 | Rising edge trigger
                                                     | set out of range to
                                                     | disable trigger
V_trigger_f               1.3        1.2        1.4  | Falling edge trigger
|
|                           typ        min        max
| [Voltage Range]           5.0        4.5        5.5
| Note, the actual voltage range and reference voltages are inherited from
| the top-level model.
|
[Pulldown]
|
-5V     -100mA     -80mA     -120mA
0V       0           0         0
10v      200mA      160mA     240mA
|
| [Pullup] table is omitted to signify Open_drain functionality.
|
|-------------------------------------------------------------------------
|
[Ramp]
|                       typ             min             max
dV/dt_r                 1.5/0.50n       1.43/0.75n      1.58/0.35n
dV/dt_f                 1.5/0.50n       1.43/0.75n      1.58/0.35n
R_load = 50
|
```

6.3 MULTI-LINGUAL MODEL EXTENSIONS

6.3.1 INTRODUCTION

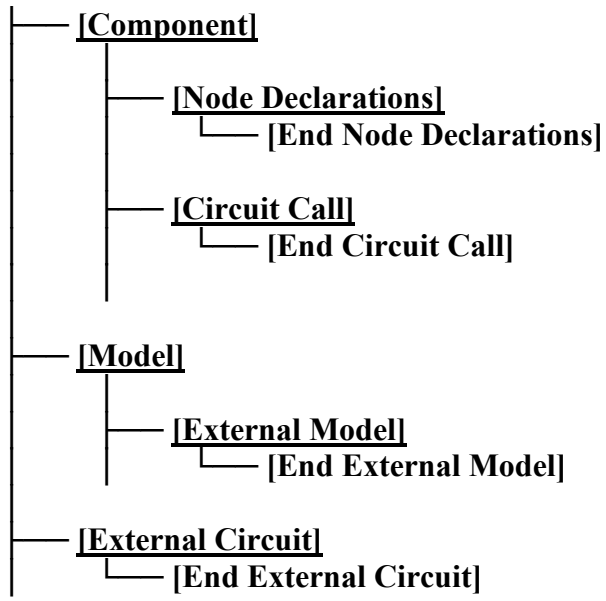
The SPICE, IBIS-ISS, VHDL-AMS and Verilog-AMS languages are supported by IBIS. This section describes how models written in these languages can be referenced and used by .ibs files.

Table 12 shows the keywords used by the language extensions within the IBIS framework.

Table 12 – Language Extension Keywords

Keyword	Description
[External Circuit] [End External Circuit]	References enhanced descriptions of structures on the die, including digital and/or analog, active and/or passive circuits
[External Model] [End External Model]	Same as [External Circuit], except limited to the connection format and usage of the [Model] keyword, with one additional feature added: support for true differential buffers
[Node Declarations] [End Node Declarations]	Lists on-die connection points related to the [Circuit Call] keyword
[Circuit Call] [End Circuit Call]	Instantiates [External Circuit]s and connects them to each other and/or die pads

The placement of these keywords within the hierarchy of IBIS is shown below:



6.3.2 LANGUAGES SUPPORTED

.ibs files can reference other files which are written using the SPICE, IBIS-ISS, VHDL-AMS, or Verilog-AMS languages. In this document, these languages are defined as follows:

"SPICE" refers to SPICE 3, Version 3F5 developed by the University of California at Berkeley, California. Many vendor-specific EDA tools are compatible with most or all of this version.

"IBIS-ISS" refers to the "IBIS Interconnect SPICE Subcircuits Specification (IBIS-ISS)", developed by the members of the IBIS Open Forum.

"VHDL-AMS" refers to "IEEE Standard VHDL Analog and Mixed-Signal Extensions", approved March 18, 1999 by the IEEE-SA Standards Board and designated IEEE Std. 1076.1-1999, or later.

"Verilog-AMS" refers to the Analog and Mixed-Signal Extensions to Verilog-HDL as documented in the Verilog-AMS Language Reference, Version 2.0, or later. This document is maintained by Accellera (formerly Open Verilog International), an independent organization. Verilog-AMS is a superset that includes Verilog-A and the Verilog Hardware Description Language IEEE 1364-2001, or later.

"VHDL-A(MS)" refers to the analog subset of VHDL-AMS described above.
"Verilog-A(MS)" refers to the analog subset of Verilog-AMS described above.

In addition, the "IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)", designated IEEE Std. 1164-1993 or later, is required to promote common digital data types for .ibs files referencing VHDL-AMS. Also, the Accellera Verilog-AMS Language Reference Manual Version 2.2 or later, is required to promote common digital data types for .ibs files referencing Verilog-AMS.

Note that, for the purposes of this section, keywords, subparameters and other data used without reference to the external languages just described are referred to collectively as "native" IBIS.

6.3.3 OVERVIEW

The four keyword pairs discussed in this section can be separated into two groups based on their functionalities. The [External Model], [End External Model], [External Circuit], and [End External Circuit] keywords are used as pointers to the models described by one of the external languages. The [Node Declarations], [End Node Declarations], [Circuit Call], and [End Circuit Call] keywords are used to describe how [External Circuit]s are connected to each other and/or to the die pads.

The [External Model] and [External Circuit] keywords are very similar in that they both support the same external languages, and they can both be used to describe passive and/or active circuitry. The key difference between the two keywords is that [External Model] can only be placed under the [Model] keyword, while [External Circuit] can only be placed outside the [Model] keyword, as illustrated in the portion of the keyword hierarchy, shown above.

The intent behind [External Model] is to provide an upgrade path from native IBIS [Model]s to the external languages (one exception to this is the support for true differential buffers). Thus, the [External Model] keyword can be used to replace the usual I-V and V-T tables, C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, C_comp_gnd_clamp subparameters, [Ramp], [Driver Schedule], [Submodel] keywords, etc. of a [Model] by any modeling technique that the external languages allow. For [External Model]s, the connectivity, test load and

specification parameters (such as Vinh and Vinl) are preserved from the [Model] keyword and the EDA tool is expected to carry out the same types of connections and measurements as are usually done with the [Model] keyword. The only difference is that the model itself is described by an external language.

In the case of the [External Circuit], however, one can model a circuit having any number of ports (see definitions below). For example, the ports may include impedance or buffer strength selection controls in addition to the usual signal and supply connections. The connectivity of an [External Circuit] is defined by the [Node Declarations] and [Circuit Call] keywords. Currently, the test loads and measurement parameters for an [External Circuit] can only be defined inside the model description itself. The results of measurements can be reported to the user or tool via other means.

The [Circuit Call] keyword acts similarly to subcircuit calls in SPICE, instantiating and connecting the various [External Circuit]s and optionally passing parameters to them. Please note that models described by the [External Model] keyword are connected according to the rules and assumptions of the [Model] keyword. [Circuit Call] is not necessary for these cases and must not be used.

6.3.4 DEFINITIONS

For the purposes of this document, several general terms are defined below.

circuit - any arbitrary collection of active or passive electrical elements treated as a unit

node - any electrical connection point; also called die node (may be digital or analog; may be a connection internal to a circuit or between circuits)

pad - a special case of a node. A pad connects a buffer or other circuitry to a package; also called die pad.

port - access point in an [External Model] or [External Circuit] definition for digital or analog signals

pseudo-differential circuits - combination of two single-ended circuits which drive and/or receive complementary signals, but where no internal current relationship exists between them

true differential circuits - circuits where a current relationship exists between two outputs or inputs which drive or receive complementary signals

6.3.5 GENERAL ASSUMPTIONS

PORTS UNDER [MODEL]S

The use of ports under native IBIS must be understood before the multi-lingual extensions can be correctly applied. The [Model] keyword assumes, but does not explicitly require, naming ports on circuits. These ports are automatically connected by IBIS-compliant tools without action by the user.

For multi-lingual modeling, ports must be explicitly named in the [External Model] or [External Circuit]; the ports are no longer assumed by EDA tools. To preserve compatibility with the

assumptions of [Model], a list of pre-defined port names has been created where the ports are reserved with fixed functionality. These reserved ports are defined in Table 13.

Table 13 – Port Names in Multi-Lingual Modeling

Port	Name	Description
1	D_drive	Digital input to a model unit
2	D_enable	Digital enable for a model unit
3	D_receive	Digital receive port of a model unit, based on data on A_signal (and/or A_signal_pos and A_signal_neg)
4	A_puref	Voltage reference port for pullup structure
5	A_pcref	Voltage reference port for power clamp structure
6	A_pdref	Voltage reference port for pulldown structure
7	A_gcref	Voltage reference port for ground clamp structure
8	A_signal	I/O signal port for a model unit
9	A_extref	External reference voltage port
10	D_switch	Digital input for control of a series switch model
11	A_gnd	Simulator global reference node
12	A_pos	Non-inverting port for series or series switch models
13	A_neg	Inverting port for series or series switch models
14	A_signal_pos	Non-inverting port of a differential model
15	A_signal_neg	Inverting port of a differential model

The first letter of the port name designates it as either digital (“D”) or analog (“A”). Reserved ports 1 through 13 are assumed or implied under the native IBIS [Model] keyword. Again, for multi-lingual models, these ports must be explicitly assigned by the user in the model if their functions are to be used. A_gnd is a simulator global reference node, similar to SPICE ideal node “0.” Ports 14 and 15 are only available under [External Model] for support of true differential buffers.

Under the [Model] description, power and ground ports are created and connected by IBIS-compliant tools as defined by the [Power Clamp Reference], [GND Clamp Reference], [Pullup Reference], [Pulldown Reference] and/or [Voltage Range] keywords. The A_signal port is connected to the die pad, to drive or receive an analog signal.

PORTS UNDER [EXTERNAL MODEL]S

The [External Model] keyword may only appear under the [Model] keyword and it may only use the same ports as assumed with the native IBIS [Model] keyword. However, [External Model] requires that reserved ports be explicitly declared in the referenced language(s); tools will continue to assume the connections to these ports.

For [External Model], reserved analog ports are usually assumed to be die pads. These ports would be connected to the component pins through [Package Model]s or [Pin] parasitics. Digital ports under [External Model] would connect to other internal digital circuitry.

Two standard [Model] structures—an I/O buffer and a Series Switch—are shown, with their associated port names, in Figure 22 and Figure 23.

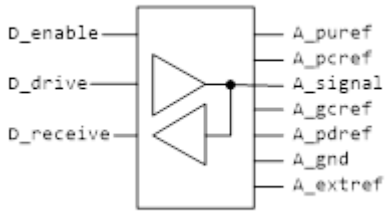


Figure 22 – Port Names for I/O Buffer

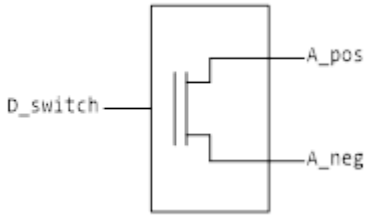


Figure 23 – Port Names for Series Switch

PORTS UNDER [EXTERNAL CIRCUIT]S

The [External Circuit] keyword allows the user to define any number of ports and port functions on a circuit. The [Circuit Call] keyword instantiates [External Circuit]s and connects their ports to specific die nodes (this can include pads). In this way, the ports of an [External Circuit] declaration become specific component die nodes. Note that, if reserved digital port names are used with an [External Circuit], those ports will be connected automatically as defined in the port list above (under [External Circuit], reserved analog port names do not retain particular meanings).

Figure 24 illustrates the use of [External Circuit]. Buffer A is an instance of [External Circuit] "X". Similarly, Buffer B is an instance of [External Circuit] "Z". These instances are created through [Circuit Call]s. [External Circuit] "Y" defines an on-die interconnect circuit. Nodes "a" through "e" and nodes "f" through "j" are specific instances of the ports defined for [External Circuit]s "X" and "Z". These ports become the internal nodes of the die and must be explicitly

declared with the [Node Declarations] keyword. The “On-die Interconnect” [Circuit Call] creates an instance of the [External Circuit] “Y” and connects the instance with the appropriate power, signal, and ground die pads. The “A” and “B” [Circuit Call]s connect the individual ports of each buffer instance to the “On-die Interconnect” [Circuit Call].

Note that the “Analog Buffer Control” signal is connected directly to the pad for pin 3. This connection is also made through an entry under the [Circuit Call] keyword.

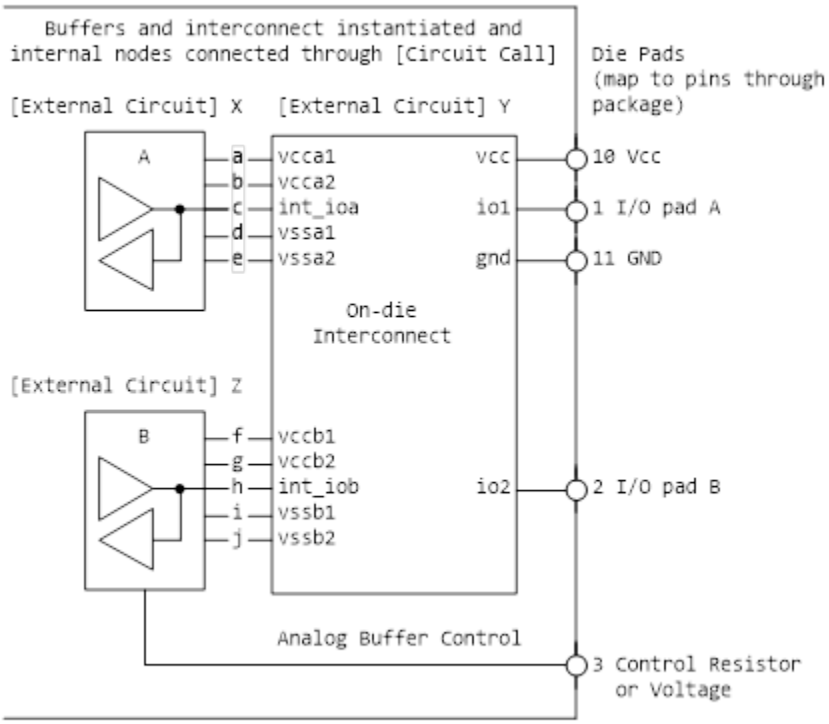


Figure 24 – Example Showing [External Circuit] Ports

The [Model], [External Model] and [External Circuit] keywords (with [Circuit Call]s and [Node Declarations] as appropriate) may be combined together in the same .ibs file or even within the same [Component] description.

PORT TYPES AND STATES

The intent of native IBIS is to model the circuit block between the region where analog signals are of interest, and the digital logic domain internal to the component. For the purposes of this discussion, the IBIS circuit block is called a “model unit” in Figure 25 and Figure 26 and the document text below.

The multi-lingual modeling extensions maintain and expand this approach, assuming that both digital signals and/or analog signals can move to and from the model unit. All VHDL-AMS and Verilog-AMS models, therefore, must have digital ports and analog ports. In certain cases, digital

ports may not be required, as in the case of interconnects; see [External Circuit] below. Routines to convert signals from one format to the other are the responsibility of the model author.

Digital ports under AMS languages must follow certain constraints on type and state. In VHDL-AMS models, analog ports must have type "electrical". Digital ports must have type "std_logic" as defined in IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164), or later. In Verilog-AMS models, analog ports must be of discipline "electrical" or a subdiscipline thereof. Digital ports must be of discipline "logic" as defined in the Accellera Verilog-AMS Language Reference Manual Version 2.2, or later and be constrained to states as defined in IEEE Std. 1164-1993, or later.

The digital ports delivering signals to the AMS model, D_drive, D_enable, and D_switch, must be limited to the '1' or '0' states for VHDL-AMS, or, equivalently, to the 1 or 0 states for Verilog-AMS. The D_receive digital port may only have the '1', '0', or 'X' states in VHDL-AMS, or, equivalently, the 1, 0, or X states in Verilog-AMS. All digital ports other than the foregoing predefined ports may use any of the logic states allowed by IEEE Std. 1164-1993, or later.

SPICE, IBIS-ISS, VHDL-A(MS), Verilog-A(MS) versus VHDL-AMS and VERILOG-AMS:

SPICE, IBIS-ISS, VHDL-A(MS), Verilog-A(MS) cannot process digital signals. All SPICE, IBIS-ISS, VHDL-A(MS), Verilog-A(MS) input and output signals must be in analog format. Consequently, IBIS multi-lingual models using SPICE, IBIS-ISS, VHDL-A(MS) or Verilog-A(MS) require analog-to-digital (A_to_D) and/or digital-to-analog (D_to_A) converters to be provided by the EDA tool. The converter subparameters are declared by the user, as part of the [External Model] or [External Circuit] syntax, with user-defined names for the ports which connect the converters to the analog ports of the SPICE, IBIS-ISS, VHDL-A(MS), or Verilog-A(MS) model. The details behind these declarations are explained in the keyword definitions below.

The electrical output characteristics of D_to_A converters are equivalent to ideal voltage sources having a zero-ohm output impedance, and the electrical input characteristics of A_to_D converters are equivalent to ideal voltage probes, having an infinite input impedance.

To summarize, Verilog-AMS and VHDL-AMS contain all the capability needed to ensure that a model unit consists of only digital ports and/or analog ports. SPICE, IBIS-ISS, VHDL-A(MS) and Verilog-A(MS), however, need extra data conversion, provided by the EDA tool, to ensure that any digital signals can be correctly processed.

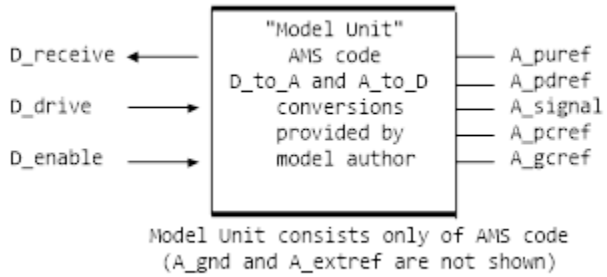


Figure 25 – AMS Model Unit, Using an I/O Buffer as an Example

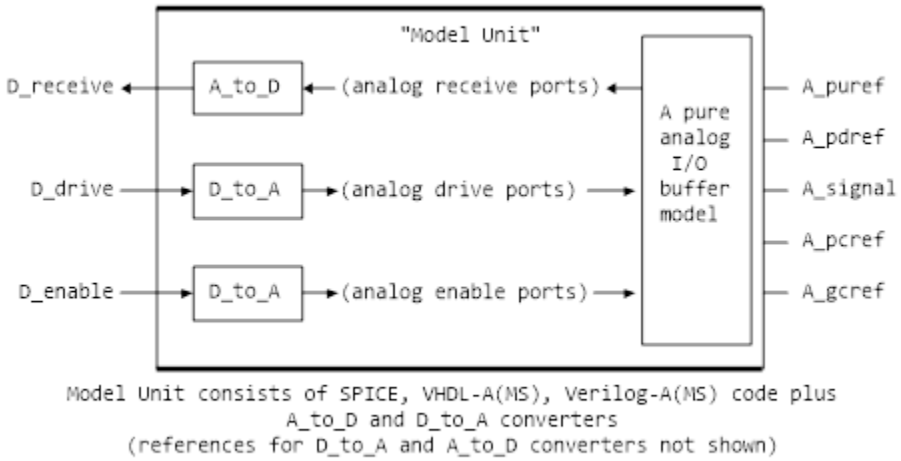


Figure 26 – An Analog-Only Model Unit, Using an I/O Buffer as an Example

6.3.6 KEYWORD DEFINITIONS

Keywords: **[External Model]**, **[End External Model]**

Required: No

Description: Used to reference an external file written in one of the supported languages containing an arbitrary circuit definition, but having ports that are compatible with the [Model] keyword, or having ports that are compatible with the [Model] keyword plus an additional signal port for true differential buffers.

Sub-Params: Language, Corner, Parameters, Converter_Parameters, Ports, D_to_A, A_to_D

Usage Rules: The [External Model] keyword must be positioned within a [Model] section and it may only appear once for each [Model] keyword in a .ibs file. It is not permitted under the [Submodel] keyword.

[Circuit Call] may not be used to connect an [External Model].

A native IBIS [Model] keyword's data may be incomplete if the [Model] correctly references an [External Model]. Any native IBIS keywords that are used in such a case must contain syntactically correct data and subparameters according to native IBIS rules. In all cases, [Model]s which reference [External Model]s must include the following keywords and subparameters:

Model_type
Vinh, Vinl (as appropriate to Model_type)
[Voltage Range] and/or [Pullup Reference], [Pulldown Reference], [POWER Clamp Reference], [GND Clamp Reference], [External Reference]
[Ramp]

In models without the [External Model] keyword, data for [Ramp] should be measured using a load that conforms to the recommendations in Section 9, "NOTES ON DATA DERIVATION METHOD". However, when used within the scope of [External Model], the [Ramp] keyword is

intended strictly to provide EDA tools with a quick first-order estimate of driver switching characteristics. When using [External Model], therefore, data for [Ramp] may be measured using a different load, if it results in data that better represent the driver's behavior in standard operation. Also, in this case, the R_load subparameter is optional, regardless of its value, and will be ignored by EDA tools. For example, the 20% to 80% voltage and time intervals for a differential buffer may be measured using the typical differential operating load appropriate to that buffer's technology. Note that voltage and time intervals must always be recorded explicitly rather than as a reduced fraction, in accordance with [Ramp] usage rules.

The following keywords and subparameters may be omitted, regardless of Model_type, from a [Model] using [External Model]:

C_comp, C_comp_pullup, C_comp_pulldown, C_comp_power_clamp, C_comp_gnd_clamp, [C Comp Model], [Pulldown], [Pullup], [POWER Clamp], [GND Clamp]

Subparameter Definitions:

Language:

Accepts "SPICE", "IBIS-ISS", "VHDL-AMS", "Verilog-AMS", "VHDL-A(MS)" or "Verilog-A(MS)" as arguments. The Language subparameter is required and must appear only once.

Corner:

Three entries follow the Corner subparameter on each line:

corner_name file_reference circuit_name

The corner_name entry is "Typ", "Min", or "Max". The file_reference entry points to a file that resides in the same directory as the .ibs file or in a relative path under that directory.

Up to three Corner lines are permitted. A "Typ" line is required. If "Min" and/or "Max" data are missing, the tool may use "Typ" data in its place. However, the tool should notify the user of this action.

Models instantiated by corner_name "Min" describe slow, weak performance, and models instantiated by corner_name "Max" describe fast, strong performance.

The circuit_name entry provides the name of the circuit to be simulated within the referenced file. For SPICE and IBIS-ISS files, this is normally a ".subckt" name. For VHDL-AMS files, this is normally an "entity(architecture)" name pair. For Verilog-AMS files, this is normally a "module" name.

No character limits, case-sensitivity limits or extension conventions are required or enforced for file_reference and circuit_name entries. However, the total number of characters in each Corner line shall comply with the rules in Section 3.2, "SYNTAX RULES". Furthermore, lower-case file_reference entries are recommended to avoid possible conflicts with file naming conventions under different operating systems. Case differences between otherwise identical file_reference entries or circuit_name entries should be avoided. External languages may not support case-sensitive distinctions.

Parameters:

Lists names of parameters that can be passed into an external model file. Each Parameters entry shall match a name or keyword in the external file or language. The list of Parameters may span several lines by using the word Parameters at the start of each line. The Parameters subparameter

is optional, and the external model shall operate with default settings without any Parameters assignments.

Parameter passing is not supported in SPICE. VHDL-AMS and VHDL-A(MS) parameters are supported using “generic” names, and Verilog-AMS and Verilog-A(MS) parameters are supported using “parameter” names. IBIS-ISS parameters are supported for all IBIS-ISS parameters which are defined on the subcircuit definition line.

Parameters are locally scoped under each [External Model] keyword, i.e., the same parameter under two different [External Model] will have independent values.

The parameter(s) listed under the Parameters subparameter may optionally be followed by an equal sign and a numeric, Boolean or string literal or a reference to a parameter name which is located in a parameter tree. The reference shall begin with a file reference, followed by an open parenthesis and a tree root name, a new open parenthesis for any branch names (including the Reserved_Parameters or Model_Specific branch names if present in the tree) and the parameter name, and a matching set of closing parentheses. Spaces are allowed in the reference following the file reference. The file reference may point to any file which contains one or more parameter trees. The file names of parameter definition files shall follow the rules for file names given in Section 3.2, “SYNTAX RULES”. In addition, file names using only a stem (e.g., xyz) are permitted. IBIS file formats except .ami (e.g., .ibs, .pkg, .ebd, .ims, .ebd, and .ems) do not contain parameter trees and are not permitted as parameter definition files. Parameter definition files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

The following rules apply to parameter trees located in parameter definition files whose file name extension is not “ami”.

a) The parameter tree shall not contain the Reserved_Parameters branch.
b) The parameter tree shall contain the Model_Specific branch.
c) The parameter tree may only contain Usage Info parameters.

The following rules shall be observed when [External Model] parameters or converter parameters reference parameters located in external parameter definition files.

a) Usage Info parameters may be referenced in any external parameter definition file with or without the “ami” extension.
b) Usage In parameters may be referenced in any parameter definition file whose file name extension is “ami”.
c) Usage Dep parameters may also be referenced in an AMI parameter definition file under the following conditions:
   - the [External Model] keyword is located under a [Model] keyword which also contains an [Algorithmic Model] keyword,
   - the [External Model] keyword's parameter and the [Algorithmic Model] keyword point to the same “.ami” file,
   - the AMI parameter definition file contains the parameter AMI_Resolve_Exists with a value of True.

   If all of these conditions are satisfied, the EDA tool shall execute the AMI_Resolve function in the executable model defined by the [Algorithmic Model] keyword to resolve

the value of any Usage Dep parameter before passing its value to the [External Model] (see Section 10.2.3).

Note that in the case when a parameter is located in a .ami file and it is of Usage In, the parameter value will be passed into the AMI executable model but this does not mean that the same parameter couldn't be used by other model(s) which are instantiated through [External Model] or [External Circuit]. Parameters described in parameter trees cannot be of AMI Format Table, Gaussian, Dual-Dirac or DjRj.

Multiple parameters may only be listed on a single line if no value assignments are made. When the Parameters line includes a parameter value assignment, each parameter must be listed on a new line. String literals must be enclosed in double quotes.

The EDA tool may provide additional means to the user to assign values to Parameters. This may include the option to override the values provided in the .ibs file, to allow the user to make selections for multi-valued parameters in the parameter tree, or to provide values for uninitialized Parameters.

Converter_Parameters:

This optional subparameter lists and initializes parameter names to be used as arguments for the A_to_D and/or D_to_A converter(s) of the [External Model] keyword under which it appears. The list of Converter_Parameters may span several lines by using the word Converter_Parameters at the start of each line. Any A_to_D or D_to_A argument which is entered as a parameter must be declared and initialized with the Converter_Parameters subparameter.

Converter_Parameters are locally scoped under each [External Model] keyword, i.e., the same converter parameter under two different [External Model]s will have independent values.

The Converter_Parameters subparameter shall contain one parameter name per line, which shall be followed by an equal sign and a constant numeric literal or a reference to a parameter name which is located in a parameter tree. The reference shall begin with a file reference, followed by an open parenthesis and a tree root name, a new open parenthesis for any branch names (including the Reserved_Parameters or Model_Specific branch names if present in the tree) and the parameter name, and a matching set of closing parentheses. Spaces are allowed in the reference following the file reference. The file reference may point to any file which contains one or more parameter trees. The file names of parameter definition files shall follow the rules for file names given in Section 3.2, "SYNTAX RULES". In addition, file names using only a stem (e.g., xyz) are permitted. IBIS file formats except .ami (e.g., .ibs, .pkg, .ebd, .ims, .emd, and .ems) do not contain parameter trees and are not permitted as parameter definition files. Parameter definition files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

The following rules apply to parameter trees located in parameter definition files whose file name extension is not "ami".

a) The parameter tree shall not contain the Reserved_Parameters branch.
b) The parameter tree shall contain the Model_Specific branch.
c) The parameter tree may only contain Usage Info parameters.

The following rules shall be observed when [External Model] parameters or converter parameters reference parameters located in external parameter definition files.

a) Usage Info parameters may be referenced in any external parameter definition file with or without the “ami” extension.
b) Usage In parameters may be referenced in any parameter definition file whose file name extension is “ami”.
c) Usage Dep parameters may also be referenced in an AMI parameter definition file under the following conditions:
   - the [External Model] keyword is located under a [Model] keyword which also contains an [Algorithmic Model] keyword,
   - the [External Model] keyword's parameter and the [Algorithmic Model] keyword point to the same AMI parameter definition file,
   - the AMI parameter definition file contains the parameter AMI_Resolve_Exists with a value of True.

   If all of these conditions are satisfied, the EDA tool shall execute the AMI_Resolve function in the executable model defined by the [Algorithmic Model] keyword to resolve the value of any Usage Dep parameter before passing its value to the [External Model] (see Section 10.2.3).

Note that in the case when a parameter is located in a .ami file and it is of Usage In, the parameter value will be passed into the AMI executable model but this does not mean that the same parameter couldn’t be used by other model(s) which are instantiated through [External Model] or [External Circuit]. Converter_Parameters described in parameter trees cannot be of AMI Format Table, Gaussian, Dual-Dirac or DjRj.

The EDA tool may provide additional means to the user to make assignments to Converter_Parameters. This may include the option to override the values provided in the .ibs file, or to allow the user to make selections for multi-valued parameters in the parameter tree.

Ports:

Ports are interfaces to the [External Model] which are available to the user and tool at the IBIS level. They are used to connect the [External Model] to die pads. The Ports parameter is used to identify the ports of the [External Model] to the simulation tool. The port assignment is by position and the port names do not have to match exactly the names inside the external file. The list of port names may span several lines if the word Ports is used at the start of each line.

Model units under [External Model] may only use reserved ports. The reserved, pre-defined port names are listed in the General Assumptions heading above. As noted earlier, digital and analog reserved port functions will be assumed by the tool and connections made accordingly. All the ports appropriate to the particular Model_type subparameter entry must be explicitly listed (see below). Note that the user may connect SPICE, IBIS-ISS, Verilog-A(MS) and VHDL-A(MS) models to A_to_D and D_to_A converters using custom names for analog ports within the model unit, as long as the digital ports of the converters use the digital reserved port names.

The rules for pad connections with [External Model] are identical to those for [Model]. The [Pin Mapping] keyword may be used with [External Model]s but is not required. If used, the [External Model] specific voltage supply ports—A_puref, A_pdref, A_gcref, A_pcref, and A_extref—are connected as defined under the [Pin Mapping] keyword. In all cases, the voltage levels connected

on the reserved supply ports are defined by the [Power Clamp Reference], [GND Clamp Reference], [Pullup Reference], [Pulldown Reference], and/or [Voltage Range] keywords, as in the case of [Model].

Digital-to-Analog/Analog-to-Digital Conversions:

These subparameters define all digital-to-analog and analog-to-digital converters needed to properly connect digital signals with the analog ports of referenced external SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) models. These subparameters must be used when [External Model] references a file written in the SPICE, IBIS-ISS, Verilog-A(MS), or VHDL-A(MS) languages. They are not permitted with Verilog-AMS or VHDL-AMS external files.

D_to_A:

As assumed in [Model], some interface ports of [External Model] circuits expect digital input signals. As SPICE, IBIS-ISS, Verilog-A(MS), or VHDL-A(MS) models understand only analog signals, some conversion from digital to analog format is required. For example, input logical states such as “0” or “1”, implied in [Model], must be converted to actual input voltage stimuli, such as a voltage ramp, for SPICE simulation.

The D_to_A subparameter provides information for converting a digital stimulus, such as “0” or “1”, into an analog voltage ramp (a digital “X” input is ignored by D_to_A converters). Each digital port which carries data for conversion to analog format must have its own D_to_A line.

The D_to_A subparameter is followed by eight or optionally nine arguments:

d_port port1 port2 vlow vhigh trise tfall corner_name polarity

The d_port entry holds the name of the digital port. This entry is used for the reserved port names D_drive, D_enable, and D_switch. The port1 and port2 entries hold the SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) analog input port names across which voltages are specified. These entries are used for the user-defined port names, together with another port name, used as a reference.

Normally port1 accepts an input signal and port2 is the reference for port1. However, for an opposite polarity stimulus, port1 could be connected to a reference port and port2 could serve as the input. In some situations, such as in the case of a true differential buffer model, it might be desirable to provide two D_to_A converters, one to drive the Non-Inverting input and the other one to drive the Inverting input. In this case the D_to_A converters may be defined with the polarity argument, one with the value Non-Inverting and the other with the value Inverting.

The vlow and vhigh entries accept analog voltage values which must correspond to the digital off and on states, where the vhigh value must be greater than the vlow value. When polarity is Non-Inverting, vlow corresponds to the digital off state '0', vhigh corresponds to the digital on state '1', trise corresponds to the analog edge rate going from the digital off to on state, and tfall corresponds to the analog edge rate going from the digital on to off state. When polarity is Inverting, the analog behavior corresponds to the opposite digital states. For example, a 3.3 V ground-referenced buffer would list vlow as 0 V and vhigh as 3.3 V. For a Non-Inverting D_to_A converter, a rising edge in D_drive would result in a transition from 0 V to 3.3 V, and for an Inverting D_to_A converter, a rising edge in D_drive would result in a transition from 3.3 V to 0 V. The trise and tfall entries are times, must be positive, and define input ramp rise and fall times between 0 and 100 percent.

The vlow, vhigh, trise and tfall arguments may be defined by parameter names, which must be declared and initialized by one or more Converter_Parameters subparameter.

The corner_name entry holds the name of the external model corner being referenced, as listed under the Corner subparameter.

The last argument, polarity, is optional. If present, its value must be "Inverting" or "Non-Inverting". If the argument is not present, "Non-Inverting" is in effect. The polarity argument may only be used with D_to_A converters which are connected to the d_port name D_drive. If the polarity argument is used, two D_to_A converter lines are required, one defined as Non-Inverting and another defined as Inverting.

At least one D_to_A line must be present, corresponding to the "Typ" corner model, for each digital line to be converted. Additional D_to_A lines for other corners may be omitted. In this case, the typical corner D_to_A entries will apply to all model corners and the "Typ" corner_name entry may be omitted if the polarity argument is not present. When the polarity argument is present, the corner_name argument must also be present.

A_to_D:

The A_to_D subparameter is used to generate a digital state ("0", "1", or "X") based on analog voltages generated by the SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) model or analog voltages present at the pad/pin. This allows an analog signal from the external SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) circuit or pad/pin to be read as a digital signal by the simulation tool.

The A_to_D subparameter is followed by six arguments:

d_port port1 port2 vlow vhigh corner_name

The d_port entry lists the reserved port name D_receive. As with D_to_A, the port1 entry would normally contain the reserved name A_signal (see below) or a user-defined port name, while port2 may list any other analog reserved port name, used as a reference. The voltage measurements are taken in this example from the port1 entry with respect to the port2 entry. These ports must also be named by the Ports subparameter.

The vlow and vhigh entries list the low and high analog threshold voltage values. The reported digital state on D_receive will be "0" if the measured voltage is lower than the vlow value, "1" if above the vhigh value, and "X" otherwise.

The vlow and vhigh arguments may be defined by parameter names, which must be declared and initialized by one or more Converter_Parameters subparameter.

The corner_name entry holds the name of the external model corner being referenced, as listed under the Corner subparameter.

At least one A_to_D line must be supplied corresponding to the "Typ" corner model. Other A_to_D lines for other corners may be omitted. In this case, the typical corner A_to_D entries will apply to all model corners.

IMPORTANT: measurements for receivers in IBIS are normally assumed to be conducted at the die pads/pins. In such cases, the electrical input model data comprises a "load" which affects the waveform seen at the pads. However, for [External Model]s, the user may choose whether to measure the analog input response at the die pads or inside the circuit (this does not preclude tools from reporting digital D_receive and/or analog port responses in addition to at-buffer terminal A_signal response). If at-buffer terminal measurements are desired, the A_signal port would be named in the A_to_D line under port1. The A_to_D converter then effectively acts "in parallel" with the load of the circuit. If internal measurements are desired (e.g., if the user wishes to view

the signal after processing by the receiver), the user-defined signal port would be named in the A_to_D line under port1. The A_to_D converter is effectively "in series" with the receiver model. The vhigh and vlow parameters should be adjusted as appropriate to the measurement point of interest. In this case, both A_signal port and user-defined signal ports shall be listed in the Ports subparameter.

Note that, while the port assignments and SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) model must be provided by the user, the D_to_A and A_to_D converters will be provided automatically by the tool (the converter parameters must still be declared by the user). There is no need for the user to develop external SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) code specifically for these functions.

A conceptual diagram of the port connections of a SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) [External Model] is shown in Figure 27. The example illustrates an I/O buffer. Note that the drawing implies that the D_receive state changes in response to the analog signal my_receive, not A_signal:

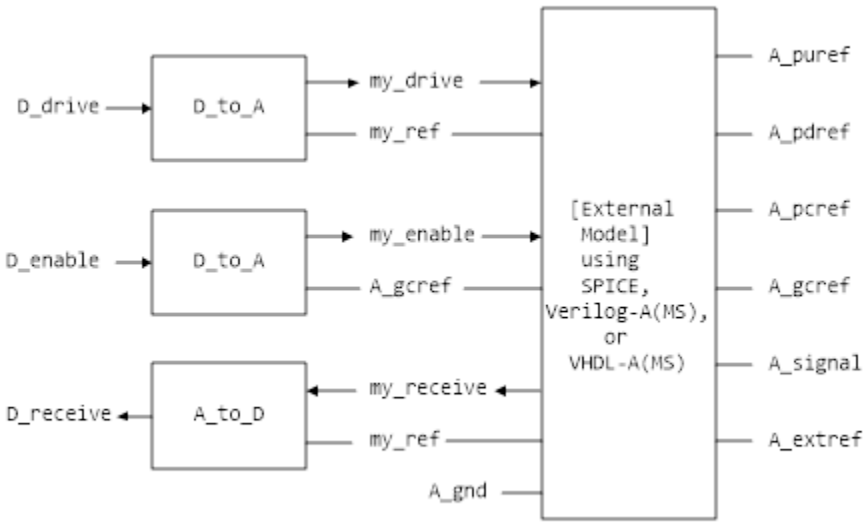


Figure 27 – Multi-lingual [External Model] I/O Buffer

Pseudo-Differential Buffers:

Pseudo-differential buffers may be described using a pair of [External Model]s which may or may not be identical. Each of the analog I/O signal ports (usually A_signal) is connected to a specific pad through the [Pin] list in the usual fashion, and the two ports are linked together as a differential pair through the [Diff Pin] keyword.

The reserved signal name A_signal is required for the I/O signal ports of [External Model]s connected to pads used in a pseudo-differential configuration.

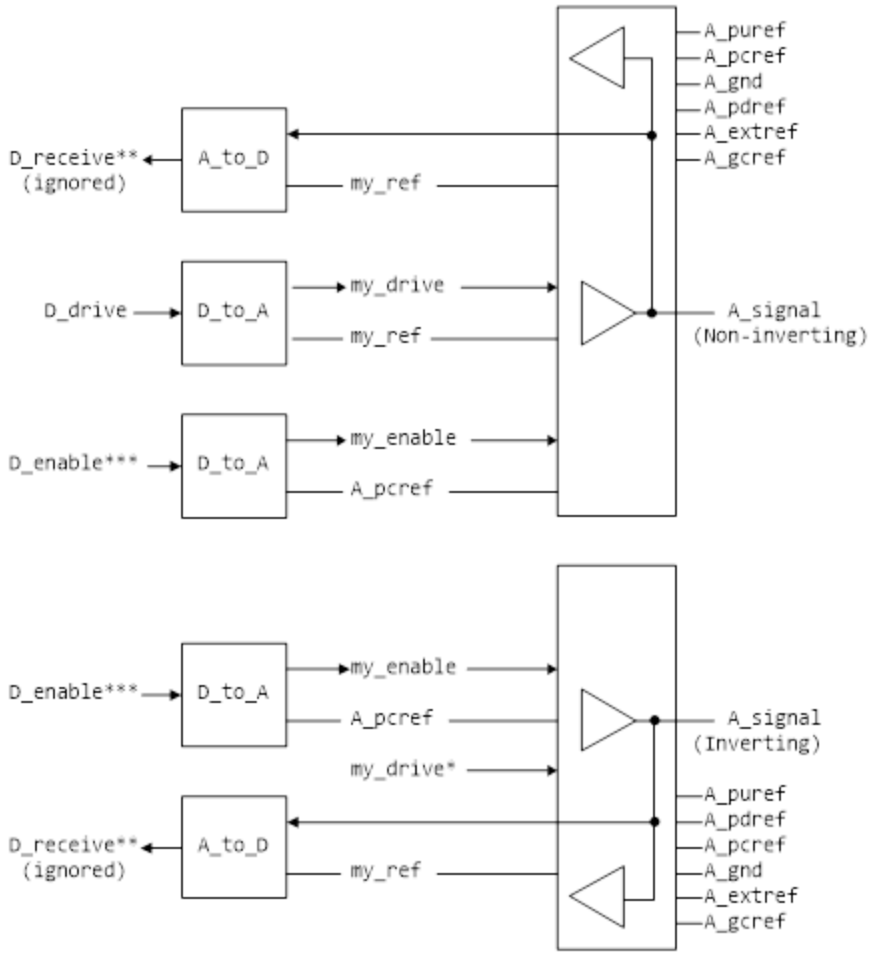
Users should note that, in pseudo-differential buffers, only one formal signal port is used to stimulate the two [External Model] digital inputs (D_drive). One of these inputs will reflect the

timing and polarity of the formal signal port named by the user, while the other input is inverted and (potentially) delayed with respect to the formal port as defined under the [Diff Pin] keyword. This second port is automatically created by the simulation tool. Users do not have to create special structures to invert or delay the driven digital signal. Simulation tools will correctly implement the two input ports once the [Diff Pin] keyword has been detected in the .ibs file. This approach is identical to that used in native IBIS.

The D_to_A adapters used for SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) files can be set up to control ports on pseudo-differential buffers. If SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) is used as an external language, the [Diff Pin] vdiff subparameter overrides the contents of vlow and vhigh under A_to_D.

IMPORTANT: For pseudo-differential buffers under [External Model], the analog input response may only be measured at the die pads. The [Diff Pin] parameter is required, and controls both the polarity and the differential thresholds used to determine the D_receive port response (the D_receive port will follow the state of the non-inverting pin/pad as referenced to the inverting pin/pad). For SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) models, the A_to_D line must name the A_signal port under either port1 or port2, as with a single-ended buffer. The A_to_D converter then effectively acts “in parallel” with the load of the buffer circuit. The vhigh and vlow parameters will be overridden by the [Diff Pin] vdiff declarations.

The port relationships are shown in Figure 28.



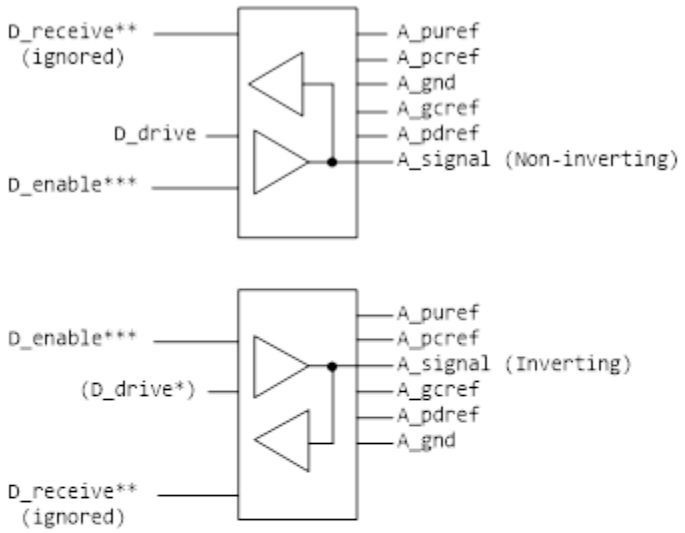
* This signal is automatically created, by inverting and delaying D_drive based on the information in [Diff Pin].

** Pseudo-differential buffers must have A_to_D entries, but D_receive is determined by the state of A_signal (Inverting) and A_signal (Non-inverting) according to the [Diff Pin] declaration.

*** D_enable is shared between the separate buffers. This sharing is handled by the EDA tool.

Figure 28 – Multi-lingual Pseudo-differential I/O Buffer

Figure 29 illustrates the same concepts with a *-AMS model. Note that the state of D_receive is determined by the tool automatically by observing the A_signal ports. The outputs of the actual receiver circuits in the *-AMS models are not used for determining D_receive.



* This signal is automatically created, by inverting and delaying D_drive based on the information in [Diff Pin] (digital output will be based on evaluation of signals %% and && also using [Diff Pin]).

** D_receive for pseudo-differential buffers is determined by the state of A_signal (Inverting) and A_signal (Non-inverting) according to the [Diff Pin] declaration.

*** D_enable is shared between the separate buffers. This sharing is handled by the EDA tool.

Figure 29 – Multi-lingual *-AMS I/O Buffers

Two additional differential timing test loads are available:

Rref_diff, Cref_diff

These subparameters are also available under the [Model Spec] keyword for typical, minimum, and maximum corners.

These timing test loads require both sides of the differential model to be operated. They can be used with the existing timing test loads Rref, Cref, and Vref. The existing timing test loads and Vmeas are used if Rref_diff and Cref_diff are *not* given.

True Differential Models:

True differential buffers may be described using [External Model]. In a true differential [External Model], the differential I/O ports which connect to die pads use the reserved names A_signal_pos and A_signal_neg, as shown in Figure 30.

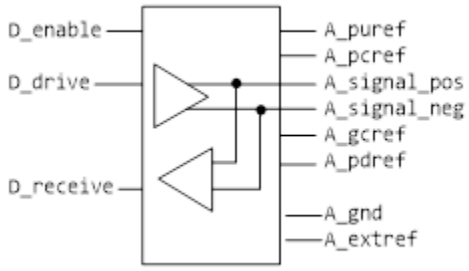


Figure 30 – Port Names for True Differential I/O Buffer

IMPORTANT: All true differential models under [External Model] assume single-ended digital port connections (D_drive, D_enable, D_receive).

The [Diff Pin] keyword is still required within the same [Component] definition when [External Model] describes a true differential buffer. The [Model] names or [Model Selector] names referenced by the pair of pins listed in an entry of the [Diff Pin] *must* be the same.

The D_to_A or A_to_D adapters used for SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) files may be set up to control or respond to true differential ports. An example is shown in Figure 31.

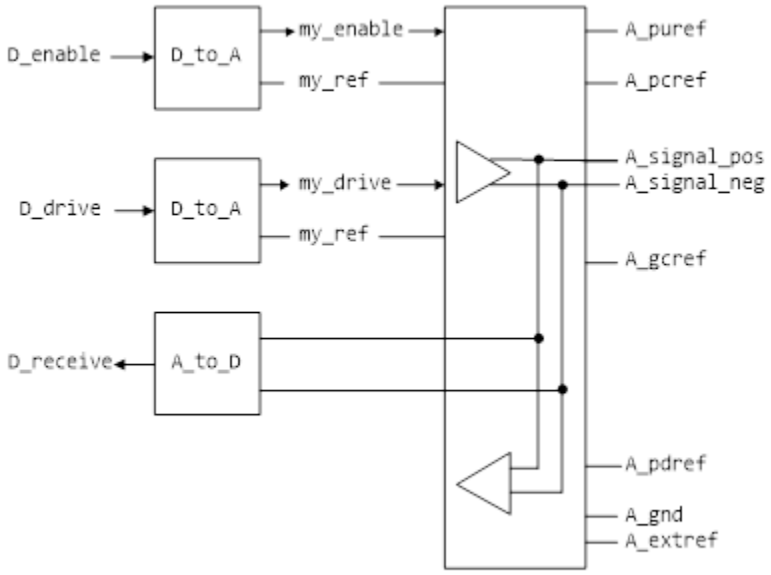


Figure 31 – Multi-lingual True Differential Buffer

If at-buffer terminal or at-pin measurement using a SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) [External Model] is desired, the vlow and vhigh entries under the A_to_D subparameter must be consistent with the values of the [Diff Pin] vdiff subparameter entry (the vlow value must match -vdiff, and the vhigh value must match +vdiff). The logic states produced by the A_to_D conversion follow the same rules as for single-ended buffers, listed above. An example is shown at the end of this section.

IMPORTANT: For true-differential buffers under [External Model], the user can choose whether to measure the analog input response at the die pads or internal to the circuit (this does not preclude tools from reporting digital D_receive and/or analog responses in addition to at-buffer terminal A_signal response). If at-buffer terminal measurements for a SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) model are desired, the A_signal_pos port would be named in the A_to_D line under port1 and A_signal_neg under port2. The A_to_D converter then effectively acts “in parallel” with the load of the buffer circuit. If internal measurements are desired (e.g., if the user wishes to view the signal after processing by the input buffer), the ports in the A_to_D line would name either two user-defined analog output signal port names (if the input buffer’s output is differential), or one user-defined analog output signal port name and a reserved or user-defined reference port name (if the input buffer’s output is single-ended). The A_to_D converter is “in series” with the receiver buffer model. The vhigh and vlow parameters should be adjusted appropriate to the measurement point of interest, so long as they as they are consistent with the [Diff Pin] vdiff declarations. In this case, A_signal_pos and A_signal_neg ports and user-defined signal ports shall be listed in the Ports subparameter.

Note that the thresholds refer to the state of the non-inverting signal, using the inverting signal as a reference. Therefore, the output signal is considered high when, for example, the non-inverting

input is +200 mV above the inverting input. Similarly, the output signal is considered low when the same non-inverting input is -200 mV "above" the inverting input.

EDA tools will report the state of the D_receive port for true differential *-AMS [External Model]s according to the AMS code written by the model author; the use of [Diff Pin] does not affect the reporting of D_receive in this case. EDA tools are free to additionally report the state of the I/O pads according to the [Diff Pin] vdiff subparameter.

For SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) and *-AMS true differential [External Model]s, the EDA tool must not override or change the model author's connection of the D_receive port.

Four additional Model_type arguments are available under the [Model] keyword. One of these must be used when an [External Model] describes a true differential model:

I/O_diff, Output_diff, 3-state_diff, Input_diff

Two additional differential timing test loads are available:

Rref_diff, Cref_diff

These subparameters are also available under the [Model Spec] keyword for the typical, minimum, and maximum corner cases.

These timing test loads require that both the inverting and non-inverting ports of the differential model refer to valid buffer model data (not terminations, supply rails, etc.). The differential test loads may also be combined with the single-ended timing test loads Rref, Cref, and Vref. Note that the single-ended timing test loads plus Vmeas are used if Rref_diff and Cref_diff are NOT supplied.

Series and Series Switch Models:

Native IBIS did not define the transition characteristics of digital switch controls. Switches were assumed to either be on or off during a simulation and I-V characteristics could be defined for either or both states. The [External Model] format allows users to control the state of a switch through the D_switch port. As with other digital ports, the use of SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) in an [External Model] requires the user to declare D_to_A ports, to convert the D_switch signal to an analog input to the SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) model (whether the port's state may actually change during a simulation is determined by the EDA tool used).

Series and Series_switch devices both are described under the [External Model] keyword using the reserved port names A_pos and A_neg. Note that the [Series Pin Mapping] keyword must be present and correctly used elsewhere in the file, in order to properly set the logic state of the switch. The A_pos port is defined in the first entry of the [Series Pin Mapping] keyword, and the A_neg port is defined in the pin2 entry. For series switches, the [Series Switch Groups] keyword is required.

Ports required for various Model_types:

As [External Model] makes use of the [Model] keyword's Model_type subparameter, not all digital and analog reserved ports may be needed for all Model_types. Table 14 and Table 15 below define which reserved port names are required for various Model_types.

Table 14 – Required Port Names for Single-ended Model_type Assignments

Model_type	**D_drive**	**D_enable**	**D_receive**	**A_signal**	**D_switch**	**A_pos**	**A_neg**
I/O*	X	X	X	X			
3-state*	X	X		X			
Output*, Open*	X			X			
Input			X	X			
Terminator				X			
Series						X	X
Series_switch					X	X	X

Table 15 – Required Port Names for Differential Model_type Assignments

Model_type	**D_drive**	**D_enable**	**D_receive**	**A_signal_pos**	**A_signal_neg**
I/O_diff	X	X	X	X	X
3-state_diff	X	X		X	X
Output_diff	X			X	X
Input_diff			X	X	X

Examples:

Example [External Model] using SPICE:

```
[Model] ExBufferSPICE
Model_type I/O
Vinh = 2.0
Vinl = 0.8
|
| Other model subparameters are optional
|
|                  typ      min     max
[Voltage Range]    3.3      3.0     3.6
|
[Ramp]
dV/dt_r         1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f         1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language SPICE
|
| Corner corner_name  file_name       circuit_name (.subckt name)
```

```
Corner    Typ         buffer_typ.spi  buffer_io_typ
Corner    Min         buffer_min.spi  buffer_io_min
Corner    Max         buffer_max.spi  buffer_io_max
|
| Parameters - Not supported in SPICE
|
| Ports List of port names (in same order as in SPICE)
Ports A_signal my_drive my_enable my_receive my_ref
Ports A_puref A_pdref A_pcref A_gcref A_extref
|
| D_to_A d_port   port1     port2    vlow vhigh trise tfall corner_name
D_to_A   D_drive  my_drive   my_ref   0.0  3.3  0.5n  0.3n   Typ
D_to_A   D_enable my_enable  A_gcref  0.0  3.3  0.5n  0.3n   Typ
|
| A_to_D d_port    port1       port2       vlow  vhigh  corner_name
A_to_D   D_receive  my_receive  my_ref  0.8   2.0    Typ
|
| Note: A_signal might also be used instead of a user-defined interface port
| for measurements taken at the die pads
|
[End External Model]
```

Example [External Model] using IBIS-ISS:

```
[Model] ExBufferISS
Model_type I/O
Vinh = 2.0
Vinl = 0.8
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language IBIS-ISS
|
| Corner corner_name file_name       circuit_name (.subckt name)
Corner    Typ         buffer_typ.spi  buffer_io_typ
Corner    Min         buffer_min.spi  buffer_io_min
Corner    Max         buffer_max.spi  buffer_io_max
|
| List of parameters
Parameters  sp_file_name = paramfile.par(RootName(Model_Specific(My_File)))
Parameters  C1_value
Parameters  R1_value = paramfile.par(RootName(Model_Specific(R1)))
|
| List of converter parameters
Converter_Parameters  MyVlow  = 0.0
Converter_Parameters  MyVhigh = 3.3
Converter_Parameters  MyVinl  = paramfile.par(RootName(Model_Specific(Vinl)))
Converter_Parameters  MyVinh  = paramfile.par(RootName(Model_Specific(Vinh)))
Converter_Parameters  MyTrise = paramfile.par(RootName(Model_Specific(Trf)))
```

```
Converter_Parameters  MyTfall = paramfile.par(RootName(Model_Specific(Trf)))
|
| Ports List of port names (in same order as in ISS)
Ports A_signal my_drive my_enable my_receive my_ref
Ports A_puref A_pdref A_pcref A_gcref A_extref
|
| D_to_A d_port   port1     port2   vlow   vhigh   trise   tfall   corner_name
D_to_A   D_drive  my_drive  my_ref  MyVlow MyVhigh MyTfall MyTrise Typ
D_to_A   D_enable my_enable A_gcref 0.0    3.3     0.5n    0.3n    Typ
|
| A_to_D d_port    port1      port2  vlow   vhigh  corner_name
A_to_D   D_receive my_receive my_ref MyVinl MyVinh Typ
|
| Note: A_signal might also be used instead of a user-defined interface port
| for measurements taken at the die pads
|
[End External Model]
```

Example [External Model] using VHDL-AMS:

```
[Model] ExBufferVHDL
Model_type I/O
Vinh = 2.0
Vinl = 0.8
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language VHDL-AMS
|
| Corner corner_name  file_name       entity(architecture)
Corner    Typ         buffer_typ.vhd  buffer(buffer_io_typ)
Corner    Min         buffer_min.vhd  buffer(buffer_io_min)
Corner    Max         buffer_max.vhd  buffer(buffer_io_max)
|
| Parameters List of parameters
Parameters delay rate
Parameters preemphasis
| Ports List of port names (in same order as in VHDL-AMS)
Ports A_signal A_puref A_pdref A_pcref A_gcref
Ports D_drive D_enable D_receive
|
[End External Model]
```

Example [External Model] using Verilog-AMS:

```
[Model] ExBufferVerilog
Model_type I/O
Vinh = 2.0
```

```
Vinl = 0.8
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language Verilog-AMS
|
| Corner corner_name  file_name     circuit_name (module)
Corner    Typ         buffer_typ.v  buffer_io_typ
Corner    Min         buffer_min.v  buffer_io_min
Corner    Max         buffer_max.v  buffer_io_max
|
| Parameters List of parameters
Parameters delay rate
Parameters preemphasis
|
| Ports List of port names (in same order as in Verilog-AMS)
Ports A_signal A_puref A_pdref A_pcref A_gcref
Ports D_drive D_enable D_receive
|
[End External Model]
```

Example [External Model] using VHDL-A(MS):

```
[Model] ExBufferVHDL_analog
Model_type I/O
Vinh = 2.0
Vinl = 0.8
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language VHDL-A(MS)
|
| Corner corner_name  file_name       circuit_name entity(architecture)
Corner    Typ         buffer_typ.vhd  buffer(buffer_io_typ)
Corner    Min         buffer_min.vhd  buffer(buffer_io_min)
Corner    Max         buffer_max.vhd  buffer(buffer_io_max)
|
| Parameters List of parameters

Parameters delay rate
Parameters preemphasis
```

```
|
| Ports List of port names (in same order as in VHDL-A(MS))
Ports A_signal my_drive my_enable my_receive my_ref
Ports A_puref A_pdref A_pcref A_gcref A_extref
|
| D_to_A d_port   port1     port2    vlow vhigh trise tfall corner_name
D_to_A   D_drive  my_drive   my_ref   0.0  3.3  0.5n  0.3n   Typ
D_to_A   D_enable my_enable  A_gcref  0.0  3.3  0.5n  0.3n   Typ
|
| A_to_D d_port    port1       port2       vlow  vhigh  corner_name
A_to_D   D_receive  my_receive  my_ref  0.8   2.0    Typ
|
| Note: A_signal might also be used instead of a user-defined interface port
| for measurements taken at the die pads
```

Example [External Model] using Verilog-A(MS):

```
[Model] ExBufferVerilog_analog
Model_type I/O
Vinh = 2.0
Vinl = 0.8
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language Verilog-A(MS)
|
| Corner corner_name  file_name      circuit_name (module)
Corner    Typ         buffer_typ.va  buffer_io_typ
Corner    Min         buffer_min.va  buffer_io_min
Corner    Max         buffer_max.va  buffer_io_max
| Parameters List of parameters
Parameters delay rate
Parameters preemphasis
|
| Ports List of port names (in same order as in Verilog-A(MS))
Ports A_signal my_drive my_enable my_receive my_ref
Ports A_puref A_pdref A_pcref A_gcref A_extref
|
| D_to_A d_port   port1     port2    vlow vhigh trise tfall corner_name
D_to_A   D_drive  my_drive   my_ref   0.0  3.3  0.5n  0.3n   Typ
D_to_A   D_enable my_enable  A_gcref  0.0  3.3  0.5n  0.3n   Typ
|
| A_to_D d_port      port1       port2   vlow  vhigh  corner_name
A_to_D   D_receive  my_receive  my_ref  0.8   2.0    Typ
|
| Note: A_signal might also be used instead of a user-defined interface port
| for measurements taken at the die pads
|
[End External Model]
```

Example of True Differential [External Model] using SPICE:

```
[Model] Ext_SPICE_Diff_Buff
Model_type I/O_diff
Rref_diff = 100
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language SPICE
|
| Corner corner_name  file_name   circuit_name (.subckt name)
Corner    Typ         diffio.spi  diff_io_typ
Corner    Min         diffio.spi  diff_io_min
Corner    Max         diffio.spi  diff_io_max
|
| Ports List of port names (in same order as in SPICE)
Ports A_signal_pos A_signal_neg my_receive my_drive my_enable
Ports A_puref A_pdref A_pcref A_gcref A_extref my_ref A_gnd
|
| D_to_A  d_port   port1      port2    vlow vhigh trise tfall corner_name
D_to_A    D_drive  my_drive   my_ref   0.0  3.3   0.5n  0.3n  Typ
D_to_A    D_drive  my_drive   my_ref   0.0  3.0   0.6n  0.3n  Min
D_to_A    D_drive  my_drive   my_ref   0.0  3.6   0.4n  0.3n  Max
D_to_A    D_enable my_enable  my_ref   0.0  3.3   0.5n  0.3n  Typ
D_to_A    D_enable my_enable  my_ref   0.0  3.0   0.6n  0.3n  Min
D_to_A    D_enable my_enable  my_ref   0.0  3.6   0.4n  0.3n  Max
|
| A_to_D  d_port     port1         port2         vlow   vhigh corner_name
A_to_D    D_receive  A_signal_pos  A_signal_neg  -200m  200m  Typ
A_to_D    D_receive  A_signal_pos  A_signal_neg  -200m  200m  Min
A_to_D    D_receive  A_signal_pos  A_signal_neg  -200m  200m  Max
|
[End External Model]
```

Example of True Differential [External Model] using IBIS-ISS:

```
[Model] Ext_ISS_Diff_Buff
Model_type I/O_diff
Rref_diff = 100
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
```

```
[External Model]
Language IBIS-ISS
|
| Corner corner_name file_name   circuit_name (.subckt name)
Corner    Typ         diffio.spi  diff_io_typ
Corner    Min         diffio.spi  diff_io_min
Corner    Max         diffio.spi  diff_io_max
|
| List of parameters
Parameters sp_file_name
Parameters c_diff  r_diff
|
|
| List of converter parameters
Converter_Parameters  MyVlow  = 0.0
Converter_Parameters  MyVhigh = 3.3
|
| Ports List of port names (in same order as in IBIS-ISS)
Ports A_signal_pos A_signal_neg my_receive my_driveP my_driveN my_enable
Ports A_puref A_pdref A_pcref A_gcref A_extref my_ref A_gnd
|
| D_to_A d_port  port1     port2 vlow vhigh trise tfall corner_name polarity
D_to_A  D_drive  my_driveP my_ref MyVlow MyVhigh 0.5n  0.3n  Typ Non-Inverting
D_to_A  D_drive  my_driveN my_ref MyVlow MyVhigh 0.5n  0.3n  Typ Inverting
D_to_A  D_enable my_enable  my_ref   0.0  3.3   0.5n  0.3n  Typ
D_to_A  D_enable my_enable  my_ref   0.0  3.0   0.6n  0.3n  Min
D_to_A  D_enable my_enable  my_ref   0.0  3.6   0.4n  0.3n  Max
|
| A_to_D d_port     port1         port2         vlow   vhigh corner_name
A_to_D  D_receive  A_signal_pos  A_signal_neg  -200m  200m  Typ
A_to_D  D_receive  A_signal_pos  A_signal_neg  -200m  200m  Min
A_to_D  D_receive  A_signal_pos  A_signal_neg  -200m  200m  Max
|
[End External Model]
```

Example of True Differential [External Model] using VHDL-AMS:

```
[Model] Ext_VHDL_Diff_Buff
Model_type I/O_diff
Rref_diff = 100
|
|                 typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
| Other model subparameters are optional
|
[External Model]
Language VHDL-AMS
|
| Corner corner_name   file_name       entity(architecture)
Corner    Typ          diffio_typ.vhd  buffer(diff_io_typ)
Corner    Min          diffio_min.vhd  buffer(diff_io_min)
Corner    Max          diffio_max.vhd  buffer(diff_io_max)
```

```
|
| Parameters List of parameters
Parameters delay rate
Parameters preemphasis
|
| Ports List of port names (in same order as in VHDL-AMS)
Ports A_signal_pos A_signal_neg D_receive D_drive D_enable
Ports A_puref A_pdref A_pcref A_gcref
|
[End External Model]
```

Example of Pseudo-Differential [External Model] using SPICE:

```
| Note that [Pin] and [Diff Pin] declarations are shown for clarity
|
|
[Pin] signal_name model_name R_pin L_pin C_pin
1 Example_pos Ext_SPICE_PDiff_Buff
2 Example_neg Ext_SPICE_PDiff_Buff
|
| ...
|
[Diff Pin] inv_pin vdiff tdelay_typ tdelay_min tdelay_max
1            2     200mV    0ns        0ns        0ns
|
| ...
|
[Model] Ext_SPICE_PDiff_Buff
Model_type I/O
|
| Other model subparameters are optional
|
|                  typ     min    max
[Voltage Range]   3.3     3.0    3.6
|
[Ramp]
dV/dt_r        1.57/0.36n   1.44/0.57n   1.73/0.28n
dV/dt_f        1.57/0.35n   1.46/0.44n   1.68/0.28n
|
[External Model]
Language SPICE
|
| Corner  corner_name  file_name   circuit_name (.subckt name)
Corner     Typ          diffio.spi  diff_io_typ
Corner     Min          diffio.spi  diff_io_min
Corner     Max          diffio.spi  diff_io_max
|
| Ports List of port names (in same order as in SPICE)
Ports A_signal my_drive my_enable my_ref
Ports A_puref A_pdref A_pcref A_gcref A_gnd A_extref
|
| D_to_A d_port   port1      port2    vlow vhigh trise tfall corner_name
D_to_A    D_drive  my_drive   my_ref   0.0  3.3   0.5n  0.3n  Typ
D_to_A    D_drive  my_drive   my_ref   0.0  3.0   0.6n  0.3n  Min
D_to_A    D_drive  my_drive   my_ref   0.0  3.6   0.4n  0.3n  Max
D_to_A    D_enable my_enable  A_pcref  0.0  3.3   0.5n  0.3n  Typ
D_to_A    D_enable my_enable  A_pcref  0.0  3.0   0.6n  0.3n  Min
```

```
D_to_A    D_enable my_enable  A_pcref  0.0  3.6   0.4n  0.3n  Max
|
| A_to_D d_port     port1     port2     vlow  vhigh corner_name
A_to_D    D_receive  A_signal  my_ref    0.8    2.0   Typ
A_to_D    D_receive  A_signal  my_ref    0.8    2.0   Min
A_to_D    D_receive  A_signal  my_ref    0.8    2.0   Max
|
| This example shows the evaluation of the received signals at the die
| pads.  [Diff Pin] defines the interpretation of the A_to_D output
| polarity and levels and overrides the A_to_D settings shown above.
|
[End External Model]
```

Keywords: **[External Circuit]**, **[End External Circuit]**

Required: No

Description: Used to reference an external file containing an arbitrary circuit description using one of the supported languages.

Sub-Params: Language, Corner, Parameters, Converter_Parameters, Ports, D_to_A, A_to_D

Usage Rules: Each [External Circuit] keyword must be followed by a unique name that differs from any name used for any [Model] or [Submodel] keyword.

The [External Circuit] keyword may appear multiple times. It is not scoped by any other keyword.

Each instance of an [External Circuit] is referenced by one or more [Circuit Call] keywords discussed later (the [Circuit Call] keyword cannot be used to reference a [Model] keyword).

The [External Circuit] keyword and contents may be placed anywhere in the file, outside of any [Component] keyword group or [Model] keyword group, in a manner similar to that of the [Model] keyword.

Subparameter Definitions:

Language:

Accepts “SPICE”, “IBIS-ISS”, “VHDL-AMS”, “Verilog-AMS”, “VHDL-A(MS)” or “Verilog-A(MS)” as arguments. The Language subparameter is required and must appear only once.

Corner:

Three entries follow the Corner subparameter on each line:

corner_name file_reference circuit_name

The corner_name entry is “Typ”, “Min”, or “Max”. The file_reference entry points to a file in that resides in the same directory as the .ibs file or in a relative path under that directory.

Up to three Corner lines are permitted. A “Typ” line is required. If “Min” and/or “Max” data are missing, the tool may use “Typ” data in its place. However, the tool should notify the user of this action.

The circuit_name entry provides the name of the circuit to be simulated within the referenced file. For SPICE and IBIS-ISS files, this is normally a “.subckt” name. For VHDL-AMS files, this is normally an “entity(architecture)” name pair. For Verilog-AMS files, this is normally a “module” name.

No character limits, case-sensitivity limits or extension conventions are required or enforced for file_reference and circuit_name entries. However, the total number of characters in each Corner line shall comply with Section 3. Furthermore, lower-case file_reference entries are recommended to avoid possible conflicts with file naming conventions under different operating systems. Case differences between otherwise identical file_reference entries or circuit_name entries should be avoided. External languages may not support case-sensitive distinctions.

Parameters:

Lists names of parameters that may be passed into an external circuit file. Each Parameters entry shall match a name or keyword in the external file or language. The list of Parameters can span several lines by using the word Parameters at the start of each line. The Parameters subparameter is optional, and the external circuit shall operate with default settings without any Parameters assignments.

Parameter passing is not supported in SPICE. VHDL-AMS and VHDL-A(MS) parameters are supported using "generic" names, and Verilog-AMS and Verilog-A(MS) parameters are supported using "parameter" names. IBIS-ISS parameters are supported for all IBIS-ISS parameters which are defined on the subcircuit definition line.

Parameters are locally scoped under each [External Circuit] keyword, i.e., the same parameter under two different [External Circuit] will have independent values.

The parameter(s) listed under the Parameters subparameter may optionally be followed by an equal sign and a numeric, Boolean or string literal or a reference to a parameter name which is located in a parameter tree. The reference shall begin with a file reference, followed by an open parenthesis and a tree root name, a new open parenthesis for any branch names (including the Reserved_Parameters or Model_Specific branch names if present in the tree) and the parameter name, and a matching set of closing parentheses. The file reference may point to any file which contains one or more parameter trees. The file names of parameter definition files shall follow the rules for file names given in Section 3.2, "SYNTAX RULES". In addition, file names using only a stem (e.g., xyz) are permitted. IBIS file formats except .ami (e.g., .ibs, .pkg, .ebd, .ims, .emd, and .ems) do not contain parameter trees and are not permitted as parameter definition files. Parameter definition files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

The following rules apply to parameter trees located in parameter definition files whose file name extension is not "ami".

a) The parameter tree shall not contain the Reserved_Parameters branch.
b) The parameter tree shall contain the Model_Specific branch.
c) The parameter tree may only contain Usage Info parameters.

The following rules shall be observed when [External Circuit] parameters or converter parameters reference parameters located in external parameter definition files.

a) Usage Info parameters may be referenced in any external parameter definition file with or without the "ami" extension.
b) Usage In parameters may be referenced in any parameter definition file whose file name extension is "ami".

Note that in the case when a parameter is located in a .ami file and it is of Usage In, the parameter value will be passed into the AMI executable model but this does not mean that the same parameter couldn't be used by other model(s) which are instantiated through [External Model] or [External Circuit].

Multiple parameters may only be listed on a single line if no value assignments are made. When the Parameters line includes a parameter value assignment, each parameter must be listed on a new line. String literals must be enclosed in double quotes.

The EDA tool may provide additional means to the user to assign values to Parameters. This may include the option to override the values provided in the .ibs file, to allow the user to make selections for multi-valued parameters in the parameter tree, or to provide values for uninitialized Parameters.

Converter_Parameters:

This optional subparameter lists and initializes parameter names to be used as arguments in the A_to_D and/or D_to_A converter(s) of the [External Circuit] keyword under which it appears. The list of Converter_Parameters may span several lines by using the word Converter_Parameters at the start of each line. Any A_to_D or D_to_A argument which is entered as a parameter must be declared and initialized with the Converter_Parameters subparameter.

Converter_Parameters are locally scoped under each [External Circuit] keyword, i.e., the same converter parameter under two different [External Circuit]s will have independent values.

The Converter_Parameters subparameter shall contain one parameter name per line, which shall be followed by an equal sign and a constant numeric literal or a reference to a parameter name which is located in a parameter tree. The reference shall begin with a file reference, followed by an open parenthesis and a tree root name, a new open parenthesis for any branch names (including the Reserved_Parameters or Model_Specific branch names if present in the tree) and the parameter name, and a matching set of closing parentheses. Spaces are allowed in the reference following the file reference. The file reference may point to any file which contains one or more parameter trees. The file names of parameter definition files shall follow the rules for file names given in Section 3.2, "SYNTAX RULES". In addition, file names using only a stem (e.g., xyz) are permitted. IBIS file formats except .ami (e.g., .ibs, .pkg, .ebd, .ims, .emd, and .ems) do not contain parameter trees and are not permitted as parameter definition files. Parameter definition files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

The following rules apply to parameter trees located in parameter definition files whose file name extension is not "ami".

a) The parameter tree shall not contain the Reserved_Parameters branch.
b) The parameter tree shall contain the Model_Specific branch.
c) The parameter tree may only contain Usage Info parameters.

The following rules shall be observed when [External Circuit] parameters or converter parameters reference parameters located in external parameter definition files.

a) Usage Info parameters may be referenced in any external parameter definition file with or without the "ami" extension.
b) Usage In parameters may be referenced in any parameter definition file whose file name extension is "ami".

Note that in the case when a parameter is located in a .ami file and it is of Usage In, the parameter value will be passed into the AMI executable model but this does not mean that the same parameter couldn't be used by other model(s) which are instantiated through [External Model] or [External Circuit]. Converter_Parameters described in parameter trees cannot be of AMI Format Table, Gaussian, Dual-Dirac or DjRj.

The EDA tool may provide additional means to the user to make assignments to Converter_Parameters. This may include the option to override the values provided in the .ibs file, or to allow the user to make selections for multi-valued parameters in the parameter tree.

Ports:

Ports are interfaces to the [External Circuit] which are available to the user and tool at the IBIS level. They are used to connect the [External Circuit] to die pads. The Ports parameter is used to identify the ports of the [External Circuit] to the simulation tool. The port assignment is by position and the port names do not have to match exactly the names inside the external file. The list of port names may span several lines if the word Ports is used at the start of each line.

[External Circuit] allows any number of ports to be defined, with any names which comply with Section 3 format requirements. Reserved port names may be used, but ONLY DIGITAL PORTS will have the pre-defined functions listed in the General Assumptions heading above. User-defined and reserved port names may be combined within the same [External Circuit].

The [Pin Mapping] keyword cannot be used with [External Circuit] in the same [Component] description.

Digital-to-Analog/Analog-to-Digital Conversions:

These subparameters define all digital-to-analog and analog-to-digital converters needed to properly connect digital signals with the analog ports of referenced external SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) models. These subparameters must be used when [External Circuit] references a file written in the SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) language. They are not permitted with Verilog-AMS or VHDL-AMS external files.

D_to_A:

As assumed in [Model] and [External Model], some interface ports of [External Circuit]s expect digital input signals. As SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) models understand only analog signals, some conversion from digital to analog format is required. For example, input logical states such as "0" or "1" must be converted to actual input voltage stimuli, such as a voltage ramp, for SPICE simulation.

The D_to_A subparameter provides information for converting a digital stimulus, such as "0" or "1", into an analog voltage ramp (a digital "X" input is ignored by D_to_A converters). Each digital port which carries data for conversion to analog format must have its own D_to_A declaration.

The D_to_A subparameter is followed by eight or optionally nine arguments:

d_port port1 port2 vlow vhigh trise tfall corner_name polarity

The d_port entry holds the name of the digital port. This entry may contain user-defined port names or the reserved port names D_drive, D_enable, and D_switch. The port1 and port2 entries hold the SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) analog input port names across which voltages are specified. These entries contain user-defined port names. One of these port entries must name a reference for the other port (for example, A_gnd).

Normally, port1 accepts an input signal and port2 is the reference for port1. However, for an opposite polarity stimulus, port1 could be connected to a voltage reference and port2 could serve as the input. In some situations, such as in the case of a true differential buffer model, it might be desirable to provide two D_to_A converters, one to drive the Non-Inverting input and the other one to drive the Inverting input. In this case the D_to_A converters may be defined with the polarity argument, one with the value Non-Inverting and the other with the value Inverting.

The vlow and vhigh entries accept voltage values which correspond to fully-off and fully-on states, where the vhigh value must be greater than the vlow value. When polarity is Non-Inverting, vlow corresponds to the digital off state '0', vhigh corresponds to the digital on state '1', trise corresponds to the analog edge rate going from the digital off to on state, and tfall corresponds to the analog edge rate going from the digital on to off state. When polarity is Inverting, the analog behavior corresponds to the opposite digital states. For example, a 3.3 V ground-referenced buffer would list vlow as 0 V and vhigh as 3.3 V. For a Non-Inverting D_to_A converter, a rising edge in D_drive would result in a transition from 0 V to 3.3 V, and for an Inverting D_to_A converter, a rising edge in D_drive would result in a transition from 3.3 V to 0 V. The trise and tfall entries are times, must be positive and define input ramp rise and fall times between 0 and 100 percent.

The vlow, vhigh, trise and tfall arguments may be defined by parameter names, which must be declared and initialized by one or more Converter_Parameters subparameter.

The corner_name entry holds the name of the external circuit corner being referenced, as listed under the Corner subparameter.

The last argument, polarity, is optional. If present, its value must be “Inverting” or “Non-Inverting”. If the argument is not present, “Non-Inverting” is in effect. The polarity argument may only be used with D_to_A converters which are connected to the d_port name D_drive. If the polarity argument is used, two D_to_A converter lines are required, one defined as Non-Inverting and another defined as Inverting. Any number of D_to_A subparameter lines is allowed, so long as each contains a unique port name entry and at least one unique port1 or port2 entry (i.e., several D_to_A declarations may use the same reference node under port1 or port2). At least one D_to_A line must be present, corresponding to the “Typ” corner model, for each digital line to be converted. Additional D_to_A lines for other corners may be omitted. In this case, the typical corner D_to_A entries will apply to all model corners and the “Typ” corner_name entry may be omitted if the polarity argument is not present. When the polarity argument is present, the corner_name argument must also be present.

A_to_D:

The A_to_D subparameter is used to generate a digital state (“0”, “1”, or “X”) based on analog voltages from the SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) model or from the pad/pin. This allows an analog signal from the external SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) circuit to be read as a digital signal by the simulation tool.

The A_to_D subparameter is followed by six arguments:

d_port port1 port2 vlow vhigh corner_name

The d_port entry lists port names to be used for digital signals going. As with D_to_A, the port1 entry would contain a user-defined analog signal. Port2 would list another port name to be used as a reference. The voltage measurements are taken from the port1 entry with respect to the port2 entry. These ports must also be named by the Ports subparameter.

The vlow and vhigh entries list the low and high analog threshold voltage values. The reported digital state on D_receive will be "0" if the measured voltage is lower than the vlow value, "1" if above the vhigh value, and "X" otherwise.

The vlow and vhigh arguments may be defined by parameter names, which must be declared and initialized by one or more Converter_Parameters subparameter.

The corner_name entry holds the name of the external model corner being referenced, as listed under the Corner subparameter.

Any number of A_to_D subparameter lines is allowed, so long as each line contains at least one column entry which is distinct from the column entries of all other lines. In practice, this means that A_to_D subparameter lines describing different corners will have identical port names. Other kinds of variations described through A_to_D subparameter lines should use unique port names. For example, a user may wish to create additional A_to_D converters for individual analog signals to monitor common mode behaviors on differential buffers.

At least one A_to_D line must be supplied corresponding to the "Typ" corner model. Other A_to_D lines for other corners may be omitted. In this case, the typical corner D_to_A entries will apply to all model corners.

IMPORTANT: measurements for receivers in IBIS may be conducted at the die pads or the pins. In such cases, the electrical input model data comprises a "load" which affects the waveform seen. However, for [External Circuit]s, the user may choose whether to measure the analog input response in the usual fashion or internal to the circuit (this does not preclude tools from reporting digital D_receive and/or analog responses in addition to normal at-pad response). If native IBIS measurements are desired, the ports in the A_to_D line would name either two user-defined analog input signal port names (if the input buffer is differential), or one user-defined analog input signal port name and a user-defined reference port name (if the input buffer is single-ended). The A_to_D converter then effectively acts "in parallel" with the load of the circuit. If internal measurements are desired (e.g., if the user wishes to view the signal after processing by the receiver), the ports in the A_to_D line would name either two user-defined analog output signal port names (if the input buffer's output is differential), or one user-defined analog output signal port name and a user-defined reference port name (if the input buffer's output is single-ended). The A_to_D converter is effectively "in series" with the receiver model. The vhigh and vlow parameters should be adjusted appropriate to the measurement point of interest.

Note that, while the port assignments and SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) model data must be provided by the user, the D_to_A and A_to_D converters will be provided automatically by the tool. There is no need for the user to develop external SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) code specifically for these functions.

The [Diff Pin] keyword is NOT required for true differential [External Circuit] descriptions.

Pseudo-differential buffers are not supported under [External Circuit]. Use the existing [Model] and [External Model] keywords to describe these structures.

Note that the EDA tool is responsible for determining the specific measurement points for reporting timing and signal quality for [External Circuit]s.

In all other respects, [External Circuit] behaves exactly as [External Model].

Examples:

Example of Model B as an [External Circuit] using SPICE:

```
[External Circuit] BUFF-SPICE
Language SPICE
|
| Corner corner_name file_reference       circuit_name (.subckt name)
Corner    Typ         buffer_typ.spi  bufferb_io_typ
Corner    Min         buffer_min.spi  bufferb_io_min
Corner    Max         buffer_max.spi  bufferb_io_max
|
| Parameters - Not supported in SPICE
|
| Ports List of port names (in same order as in SPICE)
Ports A_signal int_in int_en int_out A_control
Ports A_puref A_pdref A_pcref A_gcref
|
| D_to_A d_port   port1   port2   vlow vhigh trise tfall corner_name
D_to_A   D_drive  int_in  my_gcref 0.0  3.3   0.5n  0.3n  Typ
D_to_A   D_drive  int_in  my_gcref 0.0  3.0   0.6n  0.3n  Min
D_to_A   D_drive  int_in  my_gcref 0.0  3.6   0.4n  0.3n  Max
D_to_A   D_enable int_en  my_gnd   0.0  3.3   0.5n  0.3n  Typ
D_to_A   D_enable int_en  my_gnd   0.0  3.0   0.6n  0.3n  Min
D_to_A   D_enable int_en  my_gnd   0.0  3.6   0.4n  0.3n  Max
|
| A_to_D d_port     port1    port2     vlow vhigh corner_name
A_to_D    D_receive  int_out  my_gcref  0.8  2.0   Typ
A_to_D    D_receive  int_out  my_gcref  0.8  2.0   Min
A_to_D    D_receive  int_out  my_gcref  0.8  2.0   Max
|
| Note, the A_signal port might also be used and int_out not defined in
| a modified .subckt.
|
[End External Circuit]
```

Example [External Circuit] using IBIS-ISS:

```
[External Circuit] BUFF-ISS
Language IBIS-ISS
|
| Corner corner_name file_reference       circuit_name (.subckt name)
Corner    Typ         buffer_typ.spi  bufferb_io_typ
Corner    Min         buffer_min.spi  bufferb_io_min
Corner    Max         buffer_max.spi  bufferb_io_max
|
| List of parameters
Parameters  sp_file_name = paramfile.par(RootName(Model_Specific(My_File)))
Parameters  C1_value
Parameters  R1_value = paramfile.par(RootName(Model_Specific(R1)))
|
Converter_Parameters  MyVlow  = 0.0
Converter_Parameters  MyVhigh = 3.3
Converter_Parameters  MyVinl  = paramfile.par(RootName(Model_Specific(Vinl)))
Converter_Parameters  MyVinh  = paramfile.par(RootName(Model_Specific(Vinh)))
Converter_Parameters  MyTrise = paramfile.par(RootName(Model_Specific(Trf)))
Converter_Parameters  MyTfall = paramfile.par(RootName(Model_Specific(Trf)))
|
| Ports List of port names (in same order as in ISS)
Ports A_signal int_in int_en int_out A_control
```

```
Ports A_puref A_pdref A_pcref A_gcref
|
| D_to_A d_port   port1  port2    vlow   vhigh   trise   tfall   corner_name
D_to_A   D_drive  int_in my_gcref MyVlow MyVhigh MyTfall MyTrise Typ
D_to_A   D_enable int_en my_gnd   0.0    3.3     0.5n    0.3n    Typ
D_to_A   D_enable int_en my_gnd   0.0    3.0     0.6n    0.3n    Min
D_to_A   D_enable int_en my_gnd   0.0    3.6     0.4n    0.3n    Max
|
| A_to_D d_port    port1   port2    vlow   vhigh  corner_name
A_to_D   D_receive int_out my_gcref MyVinl MyVinh Typ
|
| Note, the A_signal port might also be used and int_out not defined in
| a modified .subckt.
|
[End External Circuit]
```

Example [External Circuit] using VHDL-AMS:

```
[External Circuit] BUFF-VHDL
Language VHDL-AMS
|
| Corner corner_name file_reference       entity(architecture)
Corner    Typ         buffer_typ.vhd  bufferb(buffer_io_typ)
Corner    Min         buffer_min.vhd  bufferb(buffer_io_min)
Corner    Max         buffer_max.vhd  bufferb(buffer_io_max)
|
| Parameters List of parameters
Parameters delay rate
Parameters preemphasis
|
| Ports List of port names (in same order as in VHDL-AMS)
Ports A_signal A_puref A_pdref A_pcref A_gcref A_control
Ports D_drive D_enable D_receive
|
[End External Circuit]
```

Example [External Circuit] using Verilog-AMS:

```
[External Circuit] BUFF-VERILOG
Language Verilog-AMS
|
| Corner corner_name file_reference     circuit_name (module)
Corner    Typ         buffer_typ.v  bufferb_io_typ
Corner    Min         buffer_min.v  bufferb_io_min
Corner    Max         buffer_max.v  bufferb_io_max
|
| Parameters List of parameters
Parameters delay rate
Parameters preemphasis
|
| Ports List of port names (in same order as in Verilog-AMS)
Ports A_signal A_puref A_pdref A_pcref A_gcref A_control
Ports D_drive D_enable D_receive

|
[End External Circuit]
```

Example [External Circuit] using SPICE:

```
| Interconnect Structure as an [External Circuit]
|
|
[External Circuit] BUS_SPI
Language SPICE
|
| Corner corner_name file_reference   circuit_name (.subckt name)
Corner    Typ        bus_typ.spi  Bus_typ
Corner    Min        bus_min.spi  Bus_min
Corner    Max        bus_max.spi  Bus_max
|
| Parameters - Not supported in SPICE
|
| Ports are in same order as defined in SPICE
Ports vcc gnd io1 io2
Ports int_ioa vcca1 vcca2 vssa1 vssa2
Ports int_iob vccb1 vccb2 vssb1 vssb2
|
| No A_to_D or D_to_A required, as no digital ports are used
|
[End External Circuit]
```

Example [External Circuit] using IBIS-ISS:

```
| Interconnect Structure as an [External Circuit]
|
|
[External Circuit] BUS_SPI
Language IBIS-ISS
|
| Corner corner_name file_reference   circuit_name (.subckt name)
Corner    Typ        bus_typ.spi  Bus_typ
Corner    Min        bus_min.spi  Bus_min
Corner    Max        bus_max.spi  Bus_max
|
| List of parameters
Parameters sp_file_name
Parameters C1_value  R1_value
|
| Ports are in same order as defined in IBIS-ISS
Ports vcc gnd io1 io2
Ports int_ioa vcca1 vcca2 vssa1 vssa2
Ports int_iob vccb1 vccb2 vssb1 vssb2
|
| No A_to_D or D_to_A required, as no digital ports are used
|
[End External Circuit]
```

Example [External Circuit] using VHDL-AMS:

```
[External Circuit] BUS_VHD
Language VHDL-AMS
|
| Corner corner_name file_reference  entity(architecture)
Corner    Typ        bus.vhd     Bus(Bus_typ)
Corner    Min        bus.vhd     Bus(Bus_min)
```

```
Corner    Max        bus.vhd     Bus(Bus_max)
|
| Parameters List of parameters
Parameters r1 l1
Parameters r2 l2 temp
|
| Ports are in the same order as defined in VHDL-AMS
Ports vcc gnd io1 io2
Ports int_ioa vcca1 vcca2 vssa1 vssa2
Ports int_iob vccb1 vccb2 vssb1 vssb2
```

Example [External Circuit] using Verilog-AMS:

```
[External Circuit] BUS_V
Language Verilog-AMS
|
| Corner corner_name file_reference circuit_name (module)
Corner    Typ         bus.v     Bus_typ
Corner    Min         bus.v     Bus_min
Corner    Max         bus.v     Bus_max
|
| Parameters List of parameters
Parameters r1 l1
Parameters r2 l2 temp
|
| Ports are in the same order as defined in Verilog-AMS
Ports vcc gnd io1 io2
Ports int_ioa vcca1 vcca2 vssa1 vssa2
Ports int_iob vccb1 vccb2 vssb1 vssb2
|
[End External Circuit]
```

The scope of the following keywords is limited to the [Component] keyword. They apply to the specific set of pin numbers and internal nodes only within that [Component].

Keywords: **[Node Declarations]**, **[End Node Declarations]**

Required: Yes, if any internal nodes exist on the die as listed in [Circuit Call], and/or if any die pads need to be explicitly defined.

Description: Provides a list of internal die nodes and/or die pads for a [Component] to make unambiguous interconnection descriptions possible.

Usage Rules: All die node and die pad names that appear under any [Circuit Call] keyword within the same [Component] must be listed under the [Node Declarations] keyword.

If used, the [Node Declarations] keyword must appear before any [Circuit Call] keyword(s) under the [Component] keyword. Only one [Node Declarations] keyword is permitted for each [Component] keyword. Since the [Node Declarations] keyword is part of the [Component] keyword, all internal node or pad references apply only to that [Component] (i.e., they are local).

The internal die node and/or die pad names within [Node Declarations] must be unique and therefore different from the pin names used in the [Pin] keyword. Each node and/or pad name must

be separated by at least one whitespace character. The list may span several lines and is terminated by the [End Node Declarations] keyword.

The names of die nodes and die pads can be composed of any combination of the legal characters outlined in Section 3.

Example:

```
[Node Declarations]          | Must appear before any [Circuit Call] keyword
|
| Die nodes:
a b c d e                    | List of die nodes
f g h nd1
|
[End Node Declarations]
```

Keywords: **[Circuit Call]**, **[End Circuit Call]**

Required: Yes, if any [External Circuit]s are present in a [Component].

Description: This keyword is used to instantiate [External Circuit]s and to connect their ports to the die nodes or die pads.

Sub-Params: Signal_pin, Diff_signal_pins, Series_pins, Parameters, Converter_Parameters, Port_map

Usage Rules: The [Circuit Call] keyword must be followed by the name of an [External Circuit] that exists in the same [Component].

When a [Circuit Call] keyword defines any connections that involve one or more die pads (and consequently pins), the corresponding pins on the [Pin] list must use the reserved word "CIRCUITCALL" in the third column instead of a model name.

Each [External Circuit] must have at least one corresponding [Circuit Call] keyword. Multiple [Circuit Call] keywords may appear under a [Component] using the same [External Circuit] name, if multiple instantiations of an [External Circuit] are needed.

Signal_pin, Diff_signal_pins, or Series_pins:

The purpose of these subparameters is to identify which [External Circuit] needs to be stimulated in order to obtain a signal on a certain pin. These subparameters must be used only when the [External Circuit] that is referenced by the [Circuit Call] keyword makes use of the stimulus signal of the EDA tool. Any given [Circuit Call] keyword must contain no more than one instance of only one of these three subparameters. The subparameter is followed by one or two pin names which must be defined by the [Pin] keyword.

Signal_pin is used when the referenced [External Circuit] has a single analog signal port (I/O) connection to one pin. The subparameter is followed by a pin name that must match one of the pin names under the [Pin] keyword.

Diff_signal_pins is used when the referenced [External Circuit] describes a true differential model which has two analog signal port (I/O) connections, each to a separate pin. The subparameter is followed by two pin names, each of which must match one of the pin names under the [Pin] keyword. The first and second pin names correspond to the non-inverting and inverting signals of the differential model, respectively. The two pin names must not be identical.

Series_pins is used when the referenced [External Circuit] describes a Series or Series_switch model which has two analog signal port (I/O) connections to two pins. The subparameter is followed by two pin names, each of which must match one of the pin names under the [Pin] keyword. The first and second pin names correspond to the positive and negative ports of the Series or Series_switch model, respectively. However, the polarity order matters only when the model is polarity sensitive (as with the [Series Current] keyword). The two pin names must not be identical.

Parameters:

Lists names of parameters that may be passed into a specific instance of an external circuit file. The rules for this subparameter are the same as the rules for the corresponding Parameters subparameter under the [External Circuit] keyword, except that if a parameter assignment exists for the same parameter name under both the [Circuit Call] and the [External Circuit] keywords, the parameter assignment under the [Circuit Call] keyword will be in effect.

Converter_Parameters:

This optional subparameter lists and initializes parameter names to be used as arguments in the A_to_D and/or D_to_A converter(s) of a specific instance of an [External Circuit]. The rules for this subparameter are the same as the rules for the corresponding Converter_Parameters subparameter under the [External Circuit] keyword, except that if a parameter assignment exists for the same parameter name under both the [Circuit Call] and the [External Circuit] keywords, the parameter assignment under the [Circuit Call] keyword will be in effect.

Port_map:

The Port_map subparameter is used to connect the ports of an [External Circuit] to die nodes or die pads.

Every occurrence of the Port_map subparameter must begin on a new line and must be followed by two arguments, the first being a port name, and the second being a die node or a pin name.

The first argument of Port_map must contain a port name that matches one of the port names in the corresponding [External Circuit] definition. No port name may be listed more than once within a [Circuit Call] statement. Only those port names need to be listed with the Port_map subparameter which are connected to a die node or a die pad. This includes reserved and/or user-defined port names.

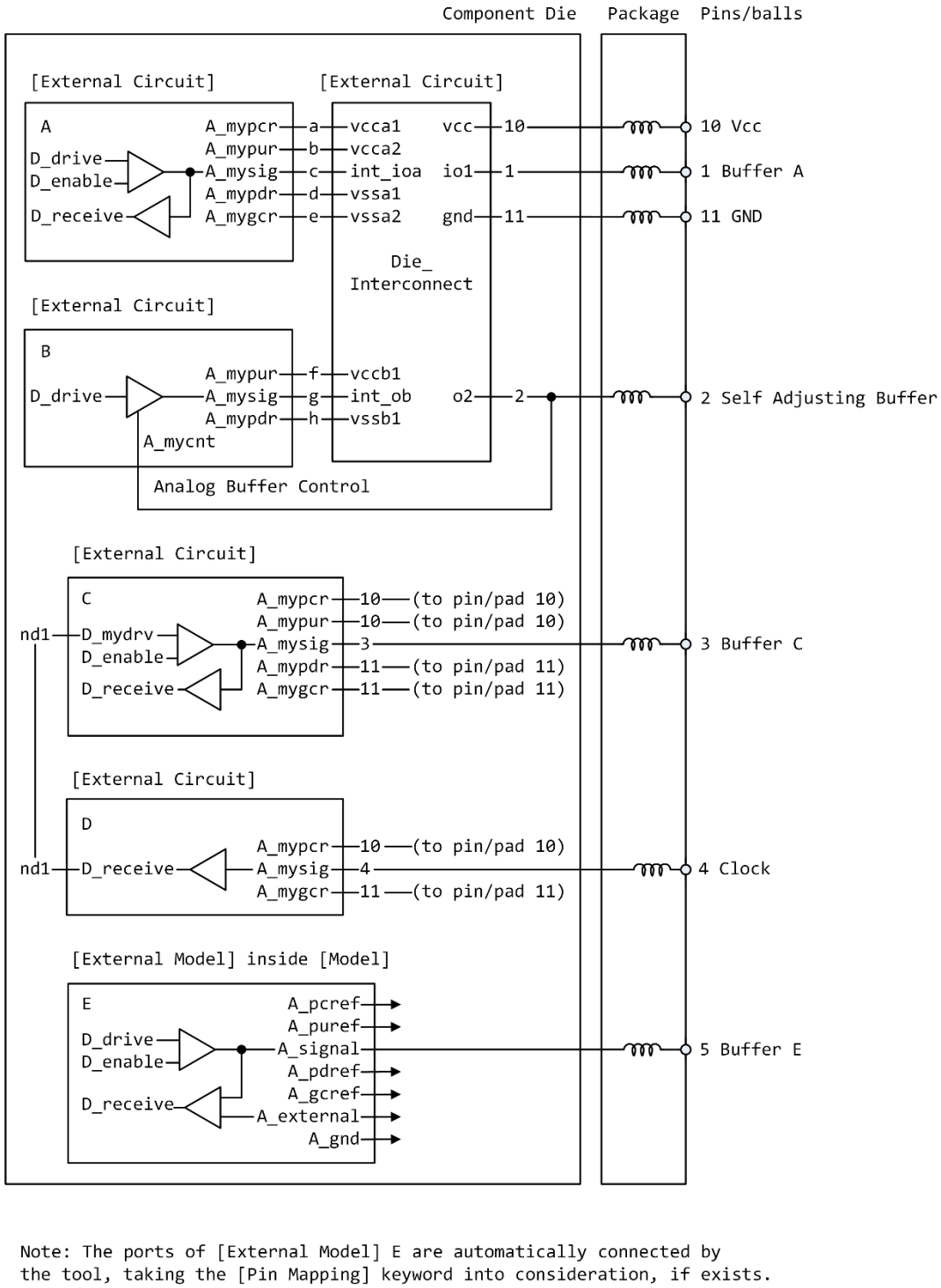
The second argument of the Port_map subparameter contains the name of a die node, die pad, or a pin. The names of die nodes, die pads, and pins may appear multiple times as Port_map subparameter arguments within the same [Circuit Call] statement to signify a common connection between multiple ports, such as common voltage supply.

Please note that a pin name in the second argument does not mean that the connection is made directly to the pin. Since IBIS assumes a one-to-one path between the die pads and pins (i.e., each pin has one and only one corresponding die pad, and each die pad has one and only one corresponding pin), it does not have a mechanism to declare die pads. Consequently, when the second argument of Port_map contains a pin name, the connection is made to the die pad that is associated with that pin name.

Examples:

NOTE REGARDING THIS EXAMPLE:

For the examples below please refer to Figure 32 and the example provided for the [Node Declarations] keyword.



Note: The ports of [External Model] E are automatically connected by the tool, taking the [Pin Mapping] keyword into consideration, if exists.

Figure 32 – Reference Example for [Node Declarations] Keyword

```
[Circuit Call] A                | Instantiates [External Circuit] named "A"
|
Signal_pin 1
|
| mapping  port          pad/node
|
Port_map   A_mypcr       a        | Port to internal node connection
Port_map   A_mypur       b        | Port to internal node connection
Port_map   A_mysig       c        | Port to internal node connection
Port_map   A_mypdr       d        | Port to internal node connection
Port_map   A_mygcr       e        | Port to internal node connection
|
[End Circuit Call]
|
|
[Circuit Call] B                | Instantiates [External Circuit] named "B"
|
Signal_pin 2
|
| mapping  port          pad/node
|
Port_map   A_mypur       f        | Port to internal node connection
Port_map   A_mysig       g        | Port to internal node connection
Port_map   A_mypdr       h        | Port to internal node connection
Port_map   A_mycnt       2        | Port to implicit pad connection
|
[End Circuit Call]
|
|
[Circuit Call] C                | Instantiates [External Circuit] named "C"
|
Signal_pin 3
|
| mapping  port          pad/node
|
Port_map   A_mypcr       10       | Port to implicit pad connection
Port_map   A_mypur       10       | Port to implicit pad connection
Port_map   A_mysig       3        | Port to implicit pad connection
Port_map   A_mypdr       11       | Port to implicit pad connection
Port_map   A_mygcr       11       | Port to implicit pad connection
Port_map   D_mydrv       nd1      | Port to internal node connection
|
[End Circuit Call]
|
|
[Circuit Call] D                | Instantiates [External Circuit] named "D"
|
Signal_pin 4
|
| mapping  port          pad/node
|
Port_map   A_my_pcref    10       | Port to implicit pad connection
Port_map   A_my_signal   4        | Port to implicit pad connection
Port_map   A_my_gcref    11       | Port to implicit pad connection
Port_map   D_receive     nd1      | Port to internal node connection
```

```
|
[End Circuit Call]
|
|
[Circuit Call] Die_Interconnect   | Instantiates [External Circuit] named
|                                   "Die_Interconnect"
|
| List of parameters
Parameters  sp_file_name = paramfile.par(RootName(Model_Specific(My_File)))
Parameters  C1_value
Parameters  R1_value = paramfile.par(RootName(Model_Specific(R1)))
Parameters  R2_value = 45
|
| mapping  port          pad/node
|
Port_map   vcc           10       | Port to implicit pad connection
Port_map   gnd           11       | Port to implicit pad connection
Port_map   io1           1        | Port to implicit pad connection
Port_map   o2            2        | Port to implicit pad connection
Port_map   vcca1         a        | Port to internal node connection
Port_map   vcca2         b        | Port to internal node connection
Port_map   int_ioa       c        | Port to internal node connection
Port_map   vssa1         d        | Port to internal node connection
Port_map   vssa2         e        | Port to internal node connection
Port_map   vccb1         f        | Port to internal node connection
Port_map   int_ob        g        | Port to internal node connection
Port_map   vssb1         h        | Port to internal node connection
|
[End Circuit Call]
```

6.4 TEST LOAD AND DATA DESCRIPTION

6.4.1 INTRODUCTION

The [Test Load] and [Test Data] keywords are top-level keywords to provide reference waveforms against which IBIS model simulation results can be compared to determine the accuracy of the IBIS data and EDA tool implementation.

6.4.2 KEYWORD DEFINITIONS

Keyword: **[Test Data]**

Required: No

Description: Indicates the beginning of a set of Golden Waveforms and references the conditions under which they were derived. A .ibs file may contain any number of [Test Data] sections representing different driver and load combinations. Golden Waveforms are a set of waveforms simulated using known ideal test loads. They are useful in verifying the accuracy of behavioral simulation results against the transistor level circuit model from which the IBIS model parameters originated.

Sub-Params: Test_data_type, Driver_model, Driver_model_inv, Test_load

Usage Rules: The name following the [Test Data] keyword is required. It allows a tool to select which data to analyze.

The Test_data_type subparameter is required, and its value must be either “Single_ended” or “Differential.” The value of Test_data_type must match the value of Test_load_type found in the load called by Test_load.

The Driver_model subparameter is required. Its value specifies the “device-under-test” and must be a valid [Model] name. Driver_model_inv is only legal if Test_data_type is Differential. Driver_model_inv is not required but may be used in the case in which a differential driver uses two different models for the inverting and non-inverting pins.

The Test_load subparameter is required and indicates which [Test Load] was used to derive the Golden Waveforms. It must reference a valid [Test Load] name.

Example:

```
[Test Data] Data1
Test_data_type Single_ended
Driver_model Buffer1
Test_load Load1
```

Keywords: **[Rising Waveform Near], [Falling Waveform Near], [Rising Waveform Far], [Falling Waveform Far], [Diff Rising Waveform Near], [Diff Falling Waveform Near], [Diff Rising Waveform Far], [Diff Falling Waveform Far]**

Required: At least one Rising/Falling waveform is required under the scope of the [Test Data] keyword.

Description: Describes the shape of the rising and falling Golden Waveforms of a given driver and a given [Test Load] measured at the driver I/O pad (near) or receiver I/O pad (far). A model developer may use the [Rising Waveform Near/Far] and [Falling Waveform Near/Far] keywords to document Golden Waveforms whose purpose is to facilitate the correlation of reference waveforms and behavioral simulations.

Usage Rules: The process, temperature, and voltage conditions under which the Golden Waveforms are generated must be identical to those used to generate the I-V and V-T tables. The Golden Waveforms must be generated using unpackaged driver and receiver models. The EDA tool must NOT use the Golden Waveform tables in the construction of its internal stimulus function.

The tables must conform to the format described under the [Rising Waveform] and [Falling Waveform] keywords.

Both differential and single-ended waveforms are allowed regardless of the value of Test_data_type. If Test_data_type is Single_ended then differential waveforms will be ignored. If Test_data_type is Differential, a single-ended waveform refers to the model specified by Driver_model and the non-inverting driver output.

Examples:

```
[Rising Waveform Far]
| Time            V(typ)             V(min)             V(max)
   0.0000s       25.2100mV          15.2200mV          43.5700mV
   0.2000ns       2.3325mV          -8.5090mV          23.4150mV
   0.4000ns       0.1484V           15.9375mV           0.3944V
   0.6000ns       0.7799V            0.2673V            1.3400V
   0.8000ns       1.2960V            0.6042V            1.9490V
   1.0000ns       1.6603V            0.9256V            2.4233V
   1.2000ns       1.9460V            1.2050V            2.8130V
   1.4000ns       2.1285V            1.3725V            3.0095V
   1.6000ns       2.3415V            1.5560V            3.1265V
   1.8000ns       2.5135V            1.7015V            3.1600V
   2.0000ns       2.6460V            1.8085V            3.1695V
| ...
  10.0000ns       2.7780V            2.3600V            3.1670V
|
[Falling Waveform Far]
| Time            V(typ)             V(min)             V(max)
   0.0000s        5.0000V            4.5000V            5.5000V
   0.2000ns       4.7470V            4.4695V            4.8815V
   0.4000ns       3.9030V            4.0955V            3.5355V
   0.6000ns       2.7313V            3.4533V            1.7770V
   0.8000ns       1.8150V            2.8570V            0.8629V
   1.0000ns       1.1697V            2.3270V            0.5364V
   1.2000ns       0.7539V            1.8470V            0.4524V
   1.4000ns       0.5905V            1.5430V            0.4368V
   1.6000ns       0.4923V            1.2290V            0.4266V
   1.8000ns       0.4639V            0.9906V            0.4207V
   2.0000ns       0.4489V            0.8349V            0.4169V
| ...
  10.0000ns       0.3950V            0.4935V            0.3841V
```

Keyword: **[Test Load]**

Required: No

Description: Defines a generic test load network and its associated electrical parameters for reference by Golden Waveforms under the [Test Data] keyword. The Golden Waveform tables correspond to a given [Test Load] which is specified by the Test_load subparameter under the [Test Data] keyword.

Sub-Params: Test_load_type, C1_near, Rs_near, Ls_near, C2_near, Rp1_near, Rp2_near, Td, Zo, Rp1_far, Rp2_far, C2_far, Ls_far, Rs_far, C1_far, V_term1, V_term2, Receiver_model, Receiver_model_inv, R_diff_near, R_diff_far.

Usage Rules: The Test_load_type subparameter is required, and its value must be either "Single_ended" or "Differential."

The subparameters specify the electrical parameters associated with a fixed generic test load. Figure 33 describes the Single_ended test load.

All subparameters except Test_load_type are optional. If omitted, series elements are shorted and shunt elements are opened by default.

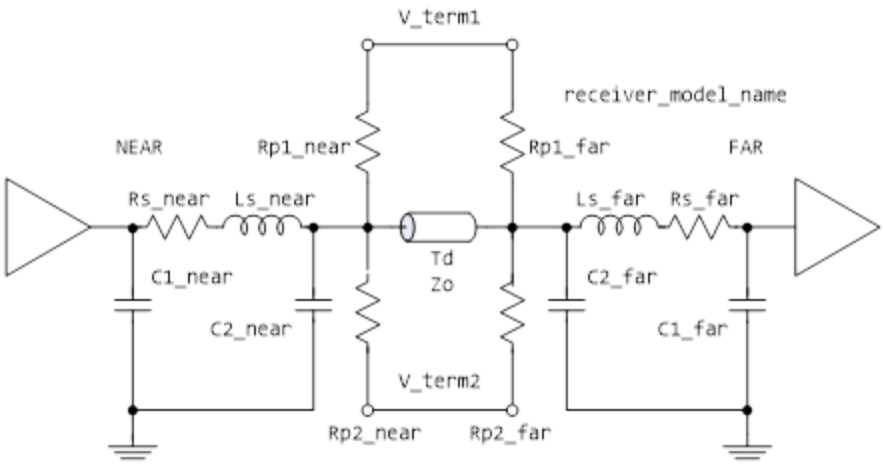


Figure 33 – [Test Load] Elements and Placement

If the Td subparameter is present, then the Zo subparameter must also be present. If the Td subparameter is not present, then the EDA tool must remove the transmission line from the network and short the two nodes to which it was connected.

V_term1 defines the termination voltage for parallel termination resistors Rp1_near and Rp1_far. This voltage is not related to the [Voltage Range] keyword. If either Rp1_near or Rp1_far is used, then V_term1 must also be used.

V_term2 defines the termination voltage for parallel termination resistors Rp2_near and Rp2_far. If either Rp2_near or Rp2_far is used, then V_term2 must also be used.

Receiver_model is optional and indicates which, if any, receiver is connected to the far end node. If not used, the network defaults to no receiver.

Receiver_model_inv is not required but may be used in the case in which a differential receiver uses two different models for the inverting and non-inverting pins. Receiver_model_inv is ignored if Test_load_type is Single_ended.

If Test_load_type is Differential, then the test load is a pair of the above circuits. If the R_diff_near or R_diff_far subparameter is used, a resistor is connected between the near or far nodes of the two circuits. If Test_load_type is Single_ended, R_diff_near and R_diff_far are ignored.

Examples:

```
[Test Load] Load1
Test_load_type Single_ended
C1_near     = 1p
Rs_near     = 10
Ls_near     = 1n
C2_near     = 1p
Rp1_near    = 100
Rp2_near    = 100
Td          = 1ns
Zo          = 50
Rp1_far     = 100
Rp2_far     = 100
C2_far      = 1p
Ls_far      = 1n
Rs_far      = 10
C1_far      = 1p
R_diff_far  = 100
Receiver_model Input1
| variable      typ             min             max
|
V_term1          1.5             1.4             1.6
V_term2          0.0             0.0             0.0
|
| Example of a transmission line and receiver test load
|
[Test Load] Tline_rcv
Td          = 1n
Zo          = 50
Receiver_model Input1
```

7 PACKAGE MODELING

7.1 INTRODUCTION

Several package modeling formats are available in IBIS. These include:

1. Lumped [Component]-level models for the entire [Component], using the [Package] keyword.
2. Lumped [Component]-level modeling per-pin, using the [Pin] keyword.
3. [Package Model] (including [Alternate Package Models] and [Define Package Model]).
4. [Interconnect Model Group] and the keywords associated with it.

The lumped formats are described in the [Package] and [Pin] keyword definitions in Section 5. Keywords for use with the [Package Model] format are described in this section, while keywords for use with [Interconnect Model Group] are described in Section 12.

7.2 RULES OF PRECEDENCE

The order of precedence for package model data to be used by EDA tools in simulation is defined below, in ascending order. If a package data format at a numerically higher position on the list is available in an IBIS or related file, that data shall be considered by the EDA tool to be more detailed and is therefore preferred.

1. [Component]/[Package]
2. [Component]/[Pin]
3. [Package Model] (including [Alternate Package Models] and [Define Package Model])
4. [Interconnect Model Group]

Note that [External Circuit] and [Node Declarations] are mutually exclusive with [Interconnect Model Group] within the same [Component]. [Package Model] and [Interconnect Model Group] may both be present for the same [Component] but should not both be used simultaneously in simulation for the same interconnect.

7.3 KEYWORD DEFINITIONS

The [Package Model] keyword is optional. If more than the default RLC package model is desired, use the [Define Package Model] keyword.

Use the [Package Model] keyword within a [Component] to indicate the package model for that component. The specification permits .ibs files to contain the following additional list of package model keywords. Note that the actual package models can be in a separate <stem>.pkg file or can exist in the .ibs files between the [Define Package Model] and [End Package Model] keywords for each package model that is defined. For reference, these keywords are listed in Table 16. EDA tools that do not support these keywords will ignore all entries between the [Define Package Model] and [End Package Model] keywords.

Table 16 – Package Modeling Keywords

Keyword	Notes
[Define Package Model]	Required if the [Package Model] keyword is used
[Manufacturer]	(note 1)
[OEM]	(note 1)
[Description]	(note 1)
[Number Of Sections]	(note 2)
[Number Of Pins]	(note 1)
[Pin Numbers]	(note 1)
[Model Data]	(note 2)
[Merged Pins]	Optional when [Model Data] is used
[Resistance Matrix]	Optional when [Model Data] is used
[Inductance Matrix]	(note 3)
[Capacitance Matrix]	(note 3)
[Bandwidth]	Required (for Banded_matrix matrices only)
[Row]	(note 3)
[End Model Data]	(note 2)
[End Package Model]	(note 1)

Notes:

1) Required when the [Define Package Model] keyword is used.
2) Either the [Number Of Sections] or the [Model Data]/[End Model Data] keywords are required. Note that [Number Of Sections] and the [Model Data]/[End Model Data] keywords are mutually exclusive.
3) Required when the [Define Package Model] keyword is used and the [Number Of Sections] keyword is not used.

When package model definitions occur within a .ibs file, their scope is “local”—they are known only within that .ibs file and no other. In addition, within that .ibs file, they override any externally defined package models that have the same name.

Usage Rules for the .Pkg File:

Package models are stored in a file whose name looks like:

<stem>.pkg

The <stem> provided shall adhere to the rules given in the [File Name] keyword. Use the "pkg" extension to identify files containing package models. The .pkg file shall contain all of the required elements of a normal .ibs file, including [IBIS Ver], [File Name], [File Rev], and the [End] keywords. Optional elements include the [Date], [Source], [Notes], [Disclaimer], [Copyright], and [Comment Char] keywords. All of the elements follow the same rules as those for a normal .ibs file.

Note that the [Component] and [Model] keywords are not allowed in the .pkg file. The .pkg file is for package models only.

Keyword: **[Define Package Model]**

Required: Yes

Description: Marks the beginning of a package model description.

Usage Rules: If the .pkg file contains data for more than one package, each section must begin with a new [Define Package Model] keyword. The length of the package model name must not exceed 40 characters. Blank characters are allowed. For every package model name defined under the [Package Model] keyword, there must be a matching [Define Package Model] keyword.

Example:

```
[Define Package Model]     QS-SMT-cer-8-pin-pkgs
```

Keyword: **[Manufacturer]**

Required: Yes

Description: Declares the manufacturer of the component(s) that use this package model.

Usage Rules: The length of the manufacturer's name must not exceed 40 characters (blank characters are allowed, e.g., Texas Instruments). In addition, each manufacturer must use a consistent name in all .ibs and .pkg files.

Example:

```
[Manufacturer]  NoName Corp.
```

Keyword: **[OEM]**

Required: Yes

Description: Declares the manufacturer of the package.

Usage Rules: The length of the manufacturer's name must not exceed 40 characters (blank characters are allowed). In addition, each manufacturer must use a consistent name in all .ibs and .pkg files.

Other Notes: This keyword is useful if the semiconductor vendor sells a single IC in packages from different manufacturers.

Example:

```
[OEM]           Acme Packaging Co.
```

Keyword: **[Description]**

Required: Yes

Description: Provides a concise yet easily human-readable description of what kind of package the [Package Model] is representing.

Usage Rules: The description shall fit on a single line, and may contain spaces.

Example:

```
[Description]   220-Pin Quad Ceramic Flat Pack
```

Keyword: **[Number Of Sections]**

Required: No

Description: Defines the maximum number of sections that make up a "package stub". A package stub is defined as the connection between the die pad and the corresponding package pin; it can include (but is not limited to) the bond wire, the connection between the bond wire and pin, and the pin itself. This keyword must be used if a modeler wishes to describe any package stub as other than a single, lumped L/R/C. The sections of a package stub are assumed to connect to each other in a series fashion.

Usage Rules: The argument is a positive integer greater than zero. This keyword, if used, must appear in the specification before the [Pin Numbers] keyword. The maximum number of sections includes sections between the Fork and Endfork subparameters.

Example:

```
[Number Of Sections] 3
```

Keyword: **[Number Of Pins]**

Required: Yes

Description: Defines the number of pins, which shall match the number of pins found in the [Pin Numbers] keyword.

Usage Rules: The field must be a positive decimal integer. The [Number Of Pins] keyword must be positioned before the [Pin Numbers] keyword.

Example:

```
[Number Of Pins]   128
```

Keyword: **[Pin Numbers]**

Required: Yes

Description: Defines the set of names that are used for the package pins, and defines pin ordering. If the [Number Of Sections] keyword is present it also lists the elements for each section of a pin's die to pin connection.

Sub-Params: Len, L, R, C, Fork, Endfork

Usage Rules: Following the [Pin Numbers] keyword, the names of the pins are listed. There must be as many names listed as the number of pins given by the preceding [Number Of Pins] keyword, but it is not required to include all of the pins listed under the [Pin] keyword. Pin names shall not exceed twenty (20) characters in length. The first pin name given is the "lowest" pin, and the last pin given is the "highest." If the [Number Of Sections] keyword is used then each pin name must be followed by one or more of the legal subparameter combinations listed below. If the [Number Of Sections] keyword is not present then subparameter usage is NOT allowed.

If a [Component] references a [Define Package Model] with the [Package Model] or an [Alternate Package Models] keyword, the EDA tool is expected to simulate the package parasitics of the pins in the component's [Pin] keyword that are not listed in the [Pin Numbers] keyword according to the hierarchy rules stated under the [Package] keyword.

However, if power/ground buses are defined by the [Pin Mapping] keyword of a component, the following rules apply to the power/ground pins. If the name of one or more power/ground pin from the power/ground bus defined by the [Pin Mapping] keyword appears under the [Pin Numbers] keyword, then the model data of those pins which are members of the same bus but are not listed under the [Pin Numbers] keyword are assumed to be merged into the model data of the pin(s) which are listed. Consequently, pins which are not listed under the [Pin Numbers] keyword shall *not* be simulated with their corresponding RLC values from the [Pin] or the [Package] keywords. Instead, all unlisted pins whose corresponding pads are members of the bus defined by the [Pin Mapping] keyword shall be shorted to the first pin of the same bus that is listed under the [Pin Numbers] keyword. This mechanism supports merged power/ground pin package modeling without additional keywords. Since the [Merged Pins] keyword (defined below) provides a mechanism to explicitly and unambiguously define the connectivity of merged pins with greater detail and freedom, the use of the [Merged Pins] keyword is strongly recommended for models in which merged pin modeling exists.

If none of the power/ground pins of a bus defined in the [Pin Mapping] keyword appears under the [Pin Numbers] keyword, the EDA tool is expected to use the RLC values from the [Pin] or [Package] keywords according to the hierarchy rules stated under the [Package] keyword.

Subparameters:

The Len, L, R, and C subparameters specify the length, inductance, capacitance and resistance of each section of each stub on a package.

The Fork and Endfork subparameters are used to denote branches from the main package stub.

Len	The length of a package stub section. Lengths are given in terms of arbitrary "units".
L	The inductance of a package stub section, in terms of henries/unit length. For example, if the total inductance of a section is 3.0nH and the length of the section is 2 "units", the inductance would be listed as L = 1.5nH (i.e., 3.0 / 2).

C	The capacitance of a package stub section, in terms of farads/unit length.
R	The DC (ohmic) resistance of a package stub section, in terms of ohms/unit length.
Fork	This subparameter indicates that the sections following (up to the Endfork subparameter) are part of a branch off of the main package stub. This subparameter has no arguments.
Endfork	This subparameter indicates the end point of a branch. For every Fork subparameter there must be a corresponding Endfork subparameter. As with the Fork subparameter, the Endfork subparameter has no arguments.

Specifying a Len or L/R/C value of zero is allowed. If Len = 0 is specified, then the L/R/C values are the total for that section. If a non-zero length is specified, then the total L/R/C for a section is calculated by multiplying the value of the Len subparameter by the value of the L, R, or C subparameter. However, if a non-zero length section is specified, the L and C for that section should be treated as distributed elements.

Using The Subparameters to Describe Package Stub Sections:

A section description begins with the Len subparameter and ends with the slash (/) character. The value of the Len, L, R, and C subparameters and the subparameter itself are separated by an equals sign (=); whitespace around the equals sign is optional. The Fork and Endfork subparameters are placed between section descriptions (i.e., between the concluding slash of one section and the "Len" parameter that starts another). A particular section description can contain no data (i.e., the description is given as "Len = 0 /").

Legal Subparameter Combinations for Section Descriptions:

A) A single Len = 0 subparameter, followed by a slash. This is used to describe a section with no data.

B) Len, and one or more of the L, R, and C subparameters. If the Len subparameter is given as zero, then the L/R/C subparameters represent lumped elements. If the Len subparameter is non-zero, then the L/R/C subparameters represent distributed elements.

C) Single Fork or Endfork subparameter. Normally, a package stub is described as several sections, with the Fork and Endfork subparameters surrounding a group of sections in the middle of the complete package stub description. However, it is legal for the Fork/Endfork subparameters to appear at the end of a section description. The package pin is connected to the last section of a package stub description not surrounded by the Fork/Endfork statements. See the examples below.

Package Stub Boundaries:

A package stub description starts at the connection to the die and ends at the point at which the package pin interfaces with the board or substrate the IC package is mounted on. Note that in the case of a component with through-hole pins, the package stub description should include only the portion of the pin not physically inserted into the board or socket. However, it is legal for a package stub description to include both the component and socket together if this is how the component is intended to be used.

Examples:

```
| A three-section package stub description that includes a bond wire (lumped
| inductance), a trace (treated as a transmission line with DC resistance),
```

```
| and a pin modeled as a lumped L/C element.
|
[Pin Numbers]
A1 Len=0 L=1.2n/ Len=1.2 L=2.0n C=0.5p R=0.05/ Len=0 L=2.0n C=1.0p/
|
| Pin A2 below has a section with no data
|
A2 Len=0 L=1.2n/ Len=0/ Len=1.2 L=2.0n C=0.5p R=0.05/ Len=0 L=2.0n C=1.0p/
|
| A section description using the Fork and Endfork subparameters.  Note that
| the indentations of the Fork and Endfork subparameters are for readability
| are not required.
|
A1 Len=0 L=2.3n /         | bond wire
Len=1.2 L=1.0n C=2.5p /   | first section
 Fork                     | indicates the starting of a branch
 Len=1.0 L=2.0n C=1.5p /  | section
 Endfork                  | ending of the branch
Len=0.5 L=1.0 C=2.5p/     | second section
Len=0.0 L=1.5n /          | pin
|
| Here is an example where the Fork/Endfork subparameters are at the end of a
| package stub description.
|
B13 Len=0 L=2.3n /        | bond wire
Len=1.2 L=1.0n C=2.5p /   | first section
Len=0.5 L=1.0 C=2.5/      | second section, pin connects here
Fork                      | indicates the starting of a branch
Len=1.0 L=2.0n C=1.5p /   | section
Endfork                   | ending of the branch
```

Keyword: **[Merged Pins]**

Required: Optional when [Model Data] is used, otherwise illegal

Description: When the [Pin Mapping] keyword defines power/ground buses that span over multiple power/ground pins (i.e., pads), the package parasitics of one or more groups of power/ground pins may be merged into one or more single pin representations. The [Merged Pins] keyword declares the package model for the pin whose name follows the [Merged Pins] keyword as a merged package model and lists the names of the pins whose package parasitics have been merged into this merged package model.

Usage Rules: This keyword may optionally be used when the [Model Data] keyword is present in the [Define Package Model] section. When used, it must be placed after the end of the pin list defined by the [Pin Numbers] keyword and before the [Model Data] keyword. The keyword must be followed by one pin name (the merging pin) on the same line on which the keyword appears, separated by at least one whitespace character. This pin name must be listed under the [Pin Numbers] keyword, it must be listed as a POWER or GND pin under the [Pin] keyword and it must also be a member of a power or ground bus defined by the [Pin Mapping] keyword. This is the pin whose package model contains the merged package model data for a group of power or ground pins.

The line on which the [Merged Pins] keyword appears must be followed by a new line providing a list of one or more pin names (the merged pins), which are separated by at least one whitespace character. The list may be on a single line, or span multiple lines, and is terminated by either another [Merged Pins] keyword or the [Model Data] keyword.

Each pin name in the list of merged pins must match the name of a POWER or GND pin in the [Pin] keyword and must also be a member of the same power or ground bus as the merging pin (pin name that follows the [Merged Pins] keyword). Pin names in this list must not be present in the pin list under the [Pin Numbers] keyword. The list must include the names of all those pins which are to be connected to the merging pin that follows the [Merged Pins] keyword due to merged modeling. No pin name may appear more than once under all [Merged Pins] keywords.

The EDA tool shall connect all of the pins (not die pads) named in the [Merged Pins] keyword together with an ideal short. It will connect other pins according to the usage rules of the [Pin Numbers] keyword.

Other Notes: Note that power integrity (PI) analysis including the package parasitics on power and ground nets is not possible with Components which do not contain power/ground bus definitions using the [Pin Mapping] keyword together with the [Define Package Model] keyword, because key pieces of information on how power is distributed between the power and ground pins and the power terminals of buffer [Model]s are not available for the EDA tool. For PI analysis, at least one power/ground pin should be included in [Pin Numbers] from each power/ground bus defined in [Pin Mapping] for a given signal pin's buffer. If no power/ground pins are defined, ideal power/ground connections based on the [Voltage Range] and/or the [* Reference] keywords can be assumed. However, there is insufficient information for PI analysis.

Example:

```
[Manufacturer]  NoName Corp.
[OEM]  ACME, Inc.
[Description]  FBGA Package Model for x4 Data Pins and POWER/GND
[Number Of Pins]  13
[Pin Numbers]
A1 |VDD
A2 |VSSQ
A8 |VSSQ
A9 |VSS
B2 |VDDQ
B3 |DQS_c
B7 |DQ1
C2 |DQ0
C3 |DQS_t
C7 |VDD
D3 |DQ2
D7 |DQ3
D9 |VSSQ

[Merged Pins] A1
H1 M1 | Merged VDD

[Merged Pins] C7
F9 J9 N9 | Merged VDD (electrically in parallel with A1, shorted at the die)

[Merged Pins] A9
```

```
C8 E9 G1 H9 K1 K9 N1 | Merged VSS

[Merged Pins] A2
D1 | Merged VSSQ (electrically in parallel with A8 and D9, shorted at the die)

[Merged Pins] B2
B8 C1 C9 E2 E8 | Merged VDDQ
```

Keyword: **[Model Data]**

Required: Yes

Description: Indicates the beginning of the formatted package model data, that can include the [Resistance Matrix], [Inductance Matrix], [Capacitance Matrix], [Bandwidth], and [Row] keywords.

Example:

```
[Model Data]
```

Keyword: **[End Model Data]**

Required: Yes

Description: Indicates the end of the formatted model data.

Other Notes: In between the [Model Data] and [End Model Data] keywords is the package model data itself. The data are a set of three matrices: the resistance (R), inductance (L), and capacitance (C) matrices. Each matrix can be formatted differently (see below). Use one of the matrix keywords below to mark the beginning of each new matrix.

Example:

```
[End Model Data]
```

Keywords: **[Resistance Matrix]**, **[Inductance Matrix]**, **[Capacitance Matrix]**

Required: [Resistance Matrix] is optional. If it is not present, its entries are assumed to be zero. [Inductance Matrix] and [Capacitance Matrix] are required.

Description: The keywords mark the beginning of a matrix, and one of three format arguments (Full_matrix, Banded_matrix, or Sparse_matrix described below) on the same line specify how the matrix data are formatted. See Figure 34.

Usage Rules: For each matrix keyword, use only one of the enumerated formats. After each of these subparameters, insert the matrix data in the appropriate format (these formats are described in detail below).

Other Notes: The resistance, inductance, and capacitance matrices are also referred to as "RLC matrices" within this specification.

When measuring the entries of the RLC matrices, either with laboratory equipment or field-solver software, currents are defined as ENTERING the pins of the package from the board (item 11 in

Section 3.2, “SYNTAX RULES”). The corresponding voltage drops are to be measured with the current pointing “in” to the “+” sign and “out” of the “-” sign.

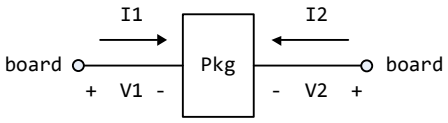


Figure 34 – Package Matrix Voltage Polarities and Current Directions

It is important to observe this convention in order to get the correct signs for the mutual inductances and resistances.

Examples:

```
[Resistance Matrix]     Banded_matrix
[Inductance Matrix]     Sparse_matrix
[Capacitance Matrix]    Full_matrix
```

RLC Matrix Notes:

For each [Resistance Matrix], [Inductance Matrix], or [Capacitance Matrix], a different format can be used for the data. The choice of formats is provided to satisfy different simulation accuracy and speed requirements.

Also, there are many packages in which the resistance matrix can have no coupling terms at all. In this case, the most concise format (Banded_matrix) can be used.

Package RLC matrices are assumed to be reciprocal and passive, thereby necessitating the R, L and C matrices be symmetric and positive semi-definite (all eigenvalues are real and non-negative, including zero). The C matrix belongs to the class of M-matrices, while the L and R matrices belong to the class of inverse M-matrices. For further information see Roger Horn and Charles Johnson (1990), *Matrix Analysis*, Cambridge University Press. Passivity enforcement and the M-matrix rules lead to the following requirements for the matrix elements.

There are two different ways to extract the coefficients that are reported in the capacitance and inductance matrices. For the purposes of this specification, the coefficients reported in the capacitance matrices shall be the “electrostatic induction coefficients” or “Maxwell’s capacitances”. The Maxwell capacitance Kij is defined as the charge induced on conductor “j” when conductor “i” is held at 1 volt and all other conductors are held at zero volts. Note that Kij (when i /= j) will be a negative number and should be entered as such. Kii shall be a non-negative number (positive or zero). Additionally, Kii coefficients should satisfy the condition of diagonal dominance, whereby each Kii coefficient shall be greater than or equal to the sum of all absolute values of the Kij coefficients (when i /= j). Likewise, for the inductance matrix the coefficients for Lij are defined as the voltage induced on conductor “j” when conductor “i”’s current is changed by 1 amp/sec and all other conductors have no current change. Lii shall be a non-negative number. Additionally, each Lii coefficient shall be larger than the absolute value of any Lij coefficient. The inverse L matrix shall also satisfy the condition of diagonal dominance. This ensures that all eigenvalues of the matrix are non-negative. For the resistance matrix, Rii coefficients shall be non-

negative numbers. If mutual resistances are included in the resistance matrix, then each Rii coefficient shall be larger than the absolute value of any Rij coefficient. The inverse R matrix shall also satisfy the condition of diagonal dominance.

One common aspect of all the different formats is that they exploit the symmetry of the matrices they describe. This means that the entries below the main diagonal of the matrix are identical to the corresponding entries above the main diagonal. Therefore, only roughly one-half of the matrix needs to be described. By convention, the main diagonal and the *upper* half of the matrix are provided.

In the following text, we use the notation [I, J] to refer to the entry in row I and column J of the matrix. Note that I and J are allowed to be alphanumeric strings as well as integers. An ordering of these strings is defined in the [Pin Numbers] section. In the following text, "Row 1" means the row corresponding to the first pin.

Also note that the numeric entries of the RLC matrices are standard IBIS floating-point numbers. As such, it is permissible to use multiplier "suffix" notation. Thus, an entry of the C matrix could be given as 1.23e-12 or as 1.23p or 1.23pF.

Full_matrix:

When the Full_matrix format is used, the couplings between every pair of elements are specified explicitly. Assume that the matrix has N rows and N columns. The Full_matrix is specified one row at a time, starting with Row 1 and continuing down to Row N.

Each new row is identified with the Row keyword.

Keyword: **[Row]**

Required: Yes

Description: Indicates the beginning of a new row of the matrix.

Usage Rules: The argument must be one of the pin names listed under the [Pin Numbers] keyword.

Example:

```
[Row]            3
```

Following a [Row] keyword is a block of numbers that represent the entries for that row. Suppose that the current row is number M. Then the first number listed is the diagonal entry, [M,M]. Following this number are the entries of the upper half of the matrix that belong to row M: [M, M+1], [M, M+2], ... up to [M,N].

For even a modest-sized package, this data will not all fit on one line. You can break the data up with new-line characters so that the 1024 character line length limit is observed.

An example: suppose the package has 40 pins and that we are currently working on Row 19. There is 1 diagonal entry, plus 40 - 19 = 21 entries in the upper half of the matrix to be specified, for 22 entries total. The data might be formatted as follows:

```
[Row]   19
5.67e-9  1.1e-9  0.8e-9  0.6e-9  0.4e-9  0.2e-9   0.1e-9   0.09e-9
8e-10    7e-10   6e-10   5e-10   4e-10   3e-10    2e-10    1e-10
```

```
9e-11    8e-11   7e-11   6e-11   5e-11   4e-11
```

In the above example, the entry 5.67e-9 is on the diagonal of row 19.

Observe that Row 1 always has the most entries, and that each successive row has one fewer entry than the last; the last row always has just a single entry.

Banded_matrix:

A Banded_matrix is one whose entries are guaranteed to be zero if they are farther away from the main diagonal than a certain distance, known as the "bandwidth." Let the matrix size be N x M, and let the bandwidth be B. An entry [I,J] of the matrix is zero if:

$$| I - J | > B$$

where |.| denotes the absolute value.

The Banded_matrix is used to specify the coupling effects up to B pins on either side. Two variations are supported. One allows for the coupling to circle back on itself. This is technically a simple form of a bordered block diagonal matrix. However, its data can be completely specified in terms of a Banded_matrix for an N x M matrix consisting of N rows and M = N + B columns. The second variation is just in terms of an N x N matrix where no circle back coupling needs to be specified.

The bandwidth for a Banded_matrix must be specified using the [Bandwidth] keyword.

Keyword: **[Bandwidth]**

Required: Yes (for Banded_matrix matrices only)

Description: Indicates the bandwidth of the matrix.

Usage Rules: The bandwidth field must be a non-negative integer. This keyword must occur after the [Resistance Matrix], etc., keywords, and before the matrix data are given.

Example:

```
[Bandwidth]     10
```

Specify the banded matrix one row at a time, starting with row 1 and working up to higher rows. Mark each row with the [Row] keyword, as above. As before, symmetry is exploited: do not provide entries below the main diagonal.

For the case where coupling can circle back on itself, consider a matrix of N pins organized into N rows, 1 ... N, and M columns, 1 ... N, 1 ... B. The first row only needs to specify the entries [1,1] through [1,1+B] since all other entries are guaranteed to be zero. The second row will need to specify the entries [2,2] through [2,2+B], and so on. For row K, the entries [K,K] through [K,K+B] are given when K + B is less than or equal to the size of the matrix N. When K + B exceeds N, the entries in the last columns, 1 ... B, specify the coupling to the first rows. For row K, the entries [K,K] ... [K,N] [K,1] ... [K,R] are given where R = mod(K + B - 1, N) + 1. All rows will contain B + 1 entries. To avoid redundant entries, the bandwidth is limited to B <= int((N - 1) / 2).

For the case where coupling does not circle back on itself, the process is modified. Only N columns need to be considered. When K + B finally exceeds the size of the matrix N, the number

of entries in each row starts to decrease; the last row (row N) has only 1 entry. This construction constrains the bandwidth to B < N.

As in the Full_matrix, if all the entries for a particular row do not fit into a single 1024-character line, the entries can be broken across several lines.

It is possible to use a bandwidth of 0 to specify a diagonal matrix (a matrix with no coupling terms). This is sometimes useful for resistance matrices.

Sparse_matrix:

A Sparse_matrix is expected to consist mostly of zero-valued entries, except for a few nonzeros. Unlike the Banded_matrix, there is no restriction on where the nonzero entries can occur. This feature is useful in certain situations, such as for Pin Grid Arrays (PGAs).

As usual, symmetry can be exploited to reduce the amount of data by eliminating from the matrix any entries below the main diagonal.

An N x N Sparse_matrix is specified one row at a time, starting with row 1 and continuing down to row N. Each new row is marked with the [Row] keyword, as in the other matrix formats.

Data for the entries of a row is given in a slightly different format, however. For the entry [I, J] of a row, it is necessary to explicitly list the name of pin J before the value of the entry is given. This specification serves to indicate to the parser where the entry is put into the matrix.

The proper location is not otherwise obvious because of the lack of restrictions on where nonzeros can occur. Each (Index, Value) pair is listed upon a separate line. An example follows. Suppose that row 10 has nonzero entries [10,10], [10,11], [10,15], and [10,25]. The following row data would be provided:

```
[Row]   10
| Index         Value
10              5.7e-9
11              1.1e-9
15              1.1e-9
25              1.1e-9
```

Note that each of the column indices listed for any row must be greater than or equal to the row index, because they always come from the upper half of the matrix. When alphanumeric pin names are used, special care must be taken to ensure that the ordering defined in the [Pin Numbers] section is observed.

With this convention, please note that the Nth row of an N x N matrix has just a single entry (the diagonal entry).

Keyword: **[End Package Model]**

Required: Yes

Description: Marks the end of a package model description.

Usage Rules: This keyword must come at the end of each complete package model description.

Optionally, add a comment after the [End Package Model] keyword to clarify which package model has just ended. For example,

[Define Package Model] My_Model
|
| ... content of model ...
|
[End Package Model] | end of My_Model

Example:

```
[End Package Model]
```

Package Model Example

The following is an example of a package model file following the package modeling specifications. For the sake of brevity, an 8-pin package has been described. For purposes of illustration, each of the matrices is specified using a different format.

Example:

```
[IBIS Ver]      8.0
[File Name]     example.pkg
[File Rev]      0.1
[Date]          December 5, 2025
[Source]        Quality Semiconductors.  Data derived from Helmholtz Inc.'s
                field solver using 3-D model from Acme Packaging.
[Notes]         Example of couplings in packaging
[Disclaimer]    The models given below may not represent any physically
                realizable 8-pin package.  They are provided solely for the
                purpose of illustrating the .pkg file format.
|
|=============================================================================
|
[Define Package Model]  QS-SMT-cer-8-pin-pkgs
[Manufacturer]          NoName Corp.
[OEM]                   Acme Package Co.
[Description]           8-Pin ceramic SMT package
[Number Of Pins]        8
|
[Pin Numbers]
1
2
3
4
5
6
7
8
|
[Model Data]
|
| The resistance matrix for this package has no coupling
|
[Resistance Matrix]     Banded_matrix
[Bandwidth]             0
[Row]   1
10.0
```

```
[Row]   2
15.0
[Row]   3
15.0
[Row]   4
10.0
[Row]   5
10.0
[Row]   6
15.0
[Row]   7
15.0
[Row]   8
10.0
|
| The inductance matrix has loads of coupling
|

[Inductance Matrix]     Full_matrix
[Row]   1
3.04859e-07      4.73185e-08      1.3428e-08     6.12191e-09
1.74022e-07      7.35469e-08     2.73201e-08     1.33807e-08
[Row]   2
3.04859e-07      4.73185e-08      1.3428e-08     7.35469e-08
1.74022e-07      7.35469e-08     2.73201e-08
[Row]   3
3.04859e-07      4.73185e-08     2.73201e-08     7.35469e-08
1.74022e-07      7.35469e-08
[Row]   4
3.04859e-07      1.33807e-08     2.73201e-08     7.35469e-08
1.74022e-07
[Row]   5
4.70049e-07      1.43791e-07     5.75805e-08     2.95088e-08
[Row]   6
4.70049e-07      1.43791e-07     5.75805e-08
[Row]   7
4.70049e-07      1.43791e-07
[Row]   8
4.70049e-07
|
| The capacitance matrix has sparse coupling
|
[Capacitance Matrix]    Sparse_matrix
[Row]   1
1       2.48227e-10
2       -1.56651e-11
5       -9.54158e-11
6       -7.15684e-12
[Row]   2
2       2.51798e-10
3       -1.56552e-11
5       -6.85199e-12
6        -9.0486e-11
7       -6.82003e-12
[Row]   3
3       2.51798e-10
4       -1.56651e-11
```

```
6       -6.82003e-12
7        -9.0486e-11
8       -6.85199e-12
[Row]   4
4       2.48227e-10
7       -7.15684e-12
8       -9.54158e-11
[Row]   5
5       1.73542e-10
6       -3.38247e-11
[Row]   6
6       1.86833e-10
7       -3.27226e-11
[Row]   7
7       1.86833e-10
8       -3.38247e-11
[Row]   8
8       1.73542e-10
|
[End Model Data]
[End Package Model]
|
```

8 ELECTRICAL BOARD DESCRIPTION

8.1 INTRODUCTION

A “board level component” is the generic term to be used to describe a printed circuit board (PCB) or substrate which can contain components or even other boards, and which can connect to another board through a set of user-visible pins. The electrical connectivity of such a board level component is referred to as an “Electrical Board Description”. For example, a SIMM module is a board level component that is used to attach several DRAM components on the PCB to another board through edge connector pins. An electrical board description file (a .ebd file) is defined to describe the connections of a board level component between the board pins and its components on the board.

A fundamental assumption regarding the electrical board description is that the inductance and capacitance parameters listed in the file are derived with respect to well-defined reference plane(s) within the board. Also, this current description does not allow one to describe electrical (inductive or capacitive) coupling between paths. It is recommended that if coupling is an issue, then an electrical description should be extracted from the physical parameters of the board.

What is, and is not, included in an Electrical Board Description is defined by its boundaries. For the definition of the boundaries, see the Description section under the [Path Description] Keyword.

Usage Rules:

A .ebd file is intended to be a stand-alone file, not referenced by nor included in any .ibs or .pkg file. Electrical Board Descriptions are stored in a file whose name looks like <stem>.ebd, where <stem> shall conform to the naming rules given in the [File Name] keyword. The “ebd” extension is mandatory.

Contents:

A .ebd file is structured similar to a standard .ibs file. It must contain the following keywords, as defined in IBIS: [IBIS Ver], [File Name], [File Rev], and [End]. It may also contain the following optional keywords: [Comment Char], [Date], [Source], [Notes], [Disclaimer], and [Copyright]. The actual board description is contained between the keywords [Begin Board Description] and [End Board Description], and includes the keywords listed below:

[Begin Board Description]
[Manufacturer]
[Number Of Pins]
[Pin List]
[Path Description]
[Reference Designator Map]
[End Board Description]

More than one [Begin Board Description]/[End Board Description] keyword pair is allowed in a .ebd file.

8.2 KEYWORD DEFINITIONS

Keyword: **[Begin Board Description]**

Required: Yes

Description: Marks the beginning of an Electrical Board Description.

Usage Rules: The keyword is followed by the name of the board level component. If the .ebd file contains more than one [Begin Board Description] keyword, then each name must be unique. The length of the component name must not exceed 40 characters, and blank characters are allowed. For every [Begin Board Description] keyword there must be a matching [End Board Description] keyword.

Example:

```
[Begin Board Description]  16Meg X 8 SIMM Module
```

Keyword: **[Manufacturer]**

Required: Yes

Description: Declares the manufacturer of the components(s) that use this .ebd file.

Usage Rules: Following the keyword is the manufacturer's name. It must not exceed 40 characters, and can include blank characters. Each manufacturer must use a consistent name in all .ebd files.

Example:

```
[Manufacturer] NoName Corp.
```

Keyword: **[Number Of Pins]**

Required: Yes

Description: Defines the number of pins, which shall match the number of pins found in the [Pin List] keyword. Pins are any externally accessible electrical connection to the component.

Usage Rules: The field must be a positive decimal integer. Note: The EDA tool must not limit the Number Of Pins to any value less than 1,000. The [Number Of Pins] keyword must be positioned before the [Pin List] keyword.

Example:

```
[Number Of Pins] 128
```

Keyword: **[Pin List]**

Required: Yes

Description: Defines the pin names of the user accessible pins, and also defines which pins are connected to power and ground.

Sub-Params: signal_name

Usage Rules: Following the [Pin List] keyword are two columns. The first column lists the pin name while the second lists the data book name of the signal connected to that pin. There shall be as many pin_name/signal_name rows as there are pins listed under the preceding [Number Of Pins] keyword. Pin names shall be the alphanumeric external pin names of the part. The pin names shall not exceed twenty (20) characters in length. Any pin associated with a signal name that begins with "GND" or "POWER" must be interpreted by simulation tools as connecting to the board's ground or power plane. In addition, "NC" is a legal signal name and indicates that the Pin is a "no connect". As noted in Section 3.2, "SYNTAX RULES", "GND," "POWER," and "NC" are case insensitive.

Example:

```
|  A SIMM Board Example:
|
[Pin List]  signal_name
 A1          GND
 A2          data1
 A3          data2
 A4          POWER5    | This pin connects to 5 V
 A5          NC        | a no connect pin
| .
| .
 A22         POWER3.3  | This pin connects to 3.3 V
 B1          casa
| .
| .
|etc.
```

Keyword: **[Path Description]**

Required: Yes

Description: This keyword allows the user to describe the connection between the user accessible pins of a board level component and other pins or pins of the ICs mounted on that board. Each pin to node connection is divided into one or more cascaded "sections," where each section is described in terms of its L/R/C per unit length. The Fork and Endfork subparameters allow the path to branch to multiple nodes, or another pin. A path description is required for each pin whose signal name is not "GND," "POWER," or "NC."

Board Description and IC Boundaries:

In any system, each board level component interfaces with another board level component at some boundary. Every electrical board description must contain the components necessary to represent the behavior of the board level component being described within its boundaries. The boundary definition depends upon the board level component being described.

For card edge connections such as a SIMM or a PC daughter card plugged into a SIMM socket or edge connector, the boundary should be at the end of the board card edge pads as they emerge from the connector.

For any through-hole mounted component, the boundary will be at the surface of the board on which the component is mounted.

Surface mounted component models end at the outboard end of their recommended surface mount pads.

If the board level component contains an unmated connector, the unmated connector will be described in a separate file, with its boundaries being as described above for the through-hole or surface mounted component.

Sub-Params: Len, L, R, C, Fork, Endfork, Pin, Node

Usage Rules: Each individual connection path (user pin to node(s)) description begins with the [Path Description] keyword and a path name, followed by the subparameters used to describe the path topology and the electrical characteristics of each section of the path. The path name must not exceed 40 characters, blanks are not allowed, and each occurrence of the [Path Description] keyword must be followed by a unique path name. Every signal pin (pins other than POWER, GND or NC) must appear in one and only one path description per [Begin Board Description]/[End Board Description] pair. Pin names do not have to appear in the same order as listed in the [Pin List] table. The individual subparameters are broken up into those that describe the electrical properties of a section, and those that describe the topology of a path.

Section Description Subparameters:

The Len, L, R, and C subparameters specify the length, the series inductance, resistance, and the capacitance to ground of each section in a path description.

- Len The physical length of a section. Lengths are given in terms of arbitrary "units". Any non-zero length requires that the parameters that follow must be interpreted as distributed elements by the EDA tool.
- L The series inductance of a section, in terms of henries/unit length. For example, if the total inductance of a section is 3.0 nH and the length of the section is 2 "units", the inductance would be listed as L = 1.5nH (i.e., 3.0 / 2).
- C The capacitance to ground of a section, in terms of farads/unit length.
- R The series DC (ohmic) resistance of a section, in terms of ohms/unit length.

Topology Description Subparameters:

The Fork and Endfork subparameters denote branches from the main pin-to-node or pin-to-pin connection path. The Node subparameter is used to reference the pin of a component or board as defined in a .ibs or .ebd file. The Pin subparameter is used to indicate the point at which a path connects to a user visible pin.

- Fork This subparameter indicates that the sections following (up to the Endfork subparameter) are part of a branch off of the main connection path. This subparameter has no arguments.
- Endfork This subparameter indicates the end point of a branch. For every Fork subparameter there must be a corresponding Endfork subparameter. As with the Fork subparameter, the Endfork subparameter has no arguments. The Fork and Endfork parameters must appear on separate lines.
- Node This subparameter is used when the connection path connects to a pin of another, externally defined component. The arguments of the Node subparameter indicate the pin and reference designator of the external component. The pin and reference designator portions of the argument are separated by a period ("."), as in

"reference_designator.pin". The reference designator is mapped to an external component description (another .ebd file or .ibs file) by the [Reference Designator Map] keyword. Note that a Node shall reference a model of a passive or active component. A Node is not an arbitrary connection point between two elements or paths.

Pin — This subparameter is used to mark the point at which a path description connects to a user accessible pin. Every path description must contain at least one occurrence of the Pin subparameter. It may also contain the reserved word NC. The value of the Pin subparameter must be one of the pin names listed in the [Pin List] section.

Note: The reserved word NC can also be used in path descriptions in a similar manner as the subparameters in order to terminate paths. This usage is optional.

Using the Subparameters to Describe Paths:

A section description begins with the Len subparameter and ends with the slash (/) character. The value of the Len, L, R, and C subparameters and the subparameter itself are separated by an equals sign (=); whitespace around the equals sign is optional. The Fork, Endfork, Node, and Pin subparameters are placed between section descriptions (i.e., between the concluding slash of one section and the "Len" parameters that starts another). The arguments of the Pin and Node subparameter are separated by whitespace.

Specifying a Len or L/R/C value of zero is allowed. If Len = 0 is specified, then the L/R/C values are the total for that section. If a non-zero length is specified, then the total L/R/C for a section is calculated by multiplying the value of the Len subparameter by the value of the L, R, or C subparameter. However, as noted below, if a non-zero length is specified, that section MUST be interpreted as distributed elements.

Legal Subparameter Combinations for Section Descriptions:

A) Len, and one or more of the L, R and C subparameters. If the Len subparameter is given as zero, then the L/R/C subparameters represent lumped elements. If the Len subparameter is non-zero, then the L/R/C subparameters represent distributed elements and both L and C must be specified, R is optional. The segment Len ..../ must not be split; the whole segment must be on one line.

B) The first subparameter following the [Path Description] keyword must be "Pin", followed by one or more section descriptions. The path description can terminate in a Node, another pin or the reserved word, NC. However, NC may be optionally omitted.

Dealing With Series Elements:

A discrete series R or L component can be included in a path description by defining a section with Len=0 and the proper R or L value. A discrete series component can also be included in a path description by writing node statements that reference the same component. This can be done as two back-to-back node statements for a series component within a single [Path Description]. It is also allowed to insert a series component between two branches of a single [Path Description], or even between two separate [Path Description]s (see the examples below).

When a series component is modeled with node statements and reference designator.pin arguments, the references pin models can be Series or Series_switch. The following models are supported: [R

Series], [L Series], [C Series], [Rl Series], [Lc Series], [Rc Series], [Series Current], and [Series MOSFET].

Examples:

An Example Path for a SIMM Module (see Figure 35):

```
|
[Path Description] CAS_2
Pin J25
Len = 0.5 L=8.35n C=3.34p R=0.01 /
Node u21.15
Len = 0.5 L=8.35n C=3.34p R=0.01 /
Node u22.15
Len = 0.5 L=8.35n C=3.34p R=0.01 /
Node u23.15
```

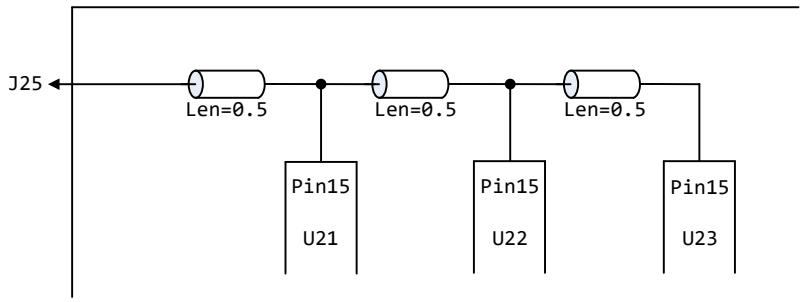


Figure 35 – SIMM Package Path Example

A Description Using the Fork and Endfork Subparameters (see Figure 36):

```
|
[Path Description] PassThru1
Pin B5
Len = 0   L=2.0n /
Len = 2.1 L=6.0n C=2.0p /
 Fork
 Len = 1.0 L = 1.0n C= 2.0p /
 Node u23.16
 Endfork
Len = 1.0 L = 6.0n C=2.0p /
Pin A5
|
```

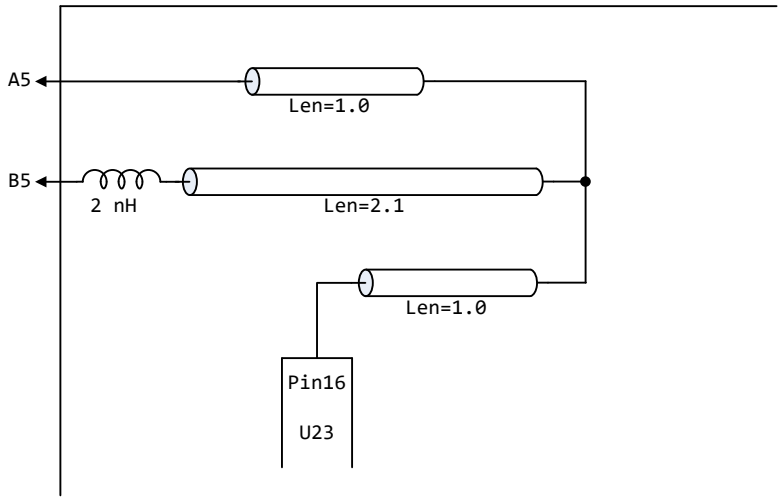


Figure 36 – Fork and Endfork in [Path Description]

A Description Including a Discrete Series Element (see Figure 37):

```
|
[Path Description] sig1
Pin B27
Len = 0  L=1.6n /
Len = 1.5 L=6.0n C=2.0p /
Node R2.1
Node R2.2
Len = 0.25 L=6.0n C=2.0p /
Node U25.6
```

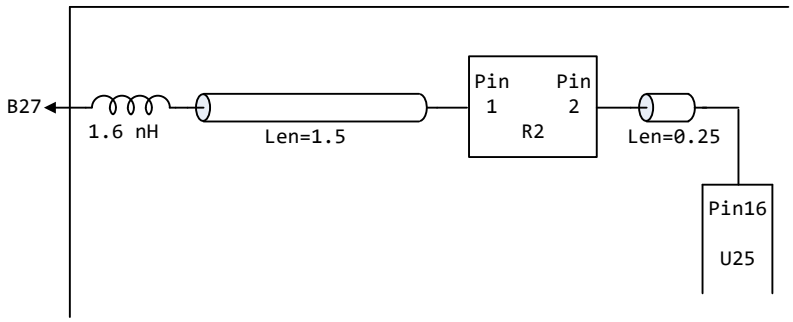


Figure 37 – Discrete Series Element in [Path Description]

A path including series passive components (C17, R21) between branches forming a differential termination (see Figure 38):

```
[Path Description]   CLK
Pin 137
Len=1.1 L=1n C=0.4p /
Node C17.1                  | Pin 1 of Series C17
Len=1.2 L=1n C=0.4p /
Node R21.1                  | Series R21 Pin 1 and 2 connections
Node R21.2
Len=1.3 L=1n C=0.4p /
Node C17.2                  | Pin 2 of Series C17
Len=1.4 L=1n C=0.4p
Pin 138
```

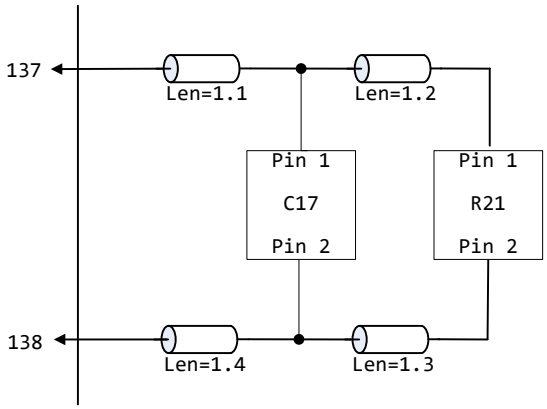


Figure 38 – Series Passive Components as Differential Termination

Two paths connected by series resistors (R8, R9) used as differential termination between components (see Figure 39):

```
[Path Description] DP+
Pin 20
Len=1 L=1n C=0.4p   /
Fork
  Len=1.1 L=1n C=0.4p /
  Fork
    Node P8.D7
  Endfork
  Len=1.2 L=1n C=0.4p /
  Node R8.1                 | Pin 1 of Series R8
Endfork
Len=1.3 L=1n C=0.4p   /
Fork
  Len=1.4 L=1n C=0.4p /
  Node P8.D5
Endfork
Len=1.5 L=1n C=0.4p   /
Node R9.1                   | Pin 1 of Series R9
```

Other path(s):

```
[Path Description] DP-
Pin 22
Len=1 L=1n C=0.4p   /
Fork
  Len=1.1 L=1n C=0.4p /
  Fork
    Node Q8.D7
  Endfork
  Len=1.2 L=1n C=0.4p /
  Node R8.2                  | Pin 2 of Series R8
Endfork
Len=1.3 L=1n C=0.4p   /
Fork
  Len=1.4 L=1n C=0.4p /
  Node Q8.D5
Endfork
Len=1.5 L=1n C=0.4p   /
Node R9.2                    | Pin 2 of Series R9
```

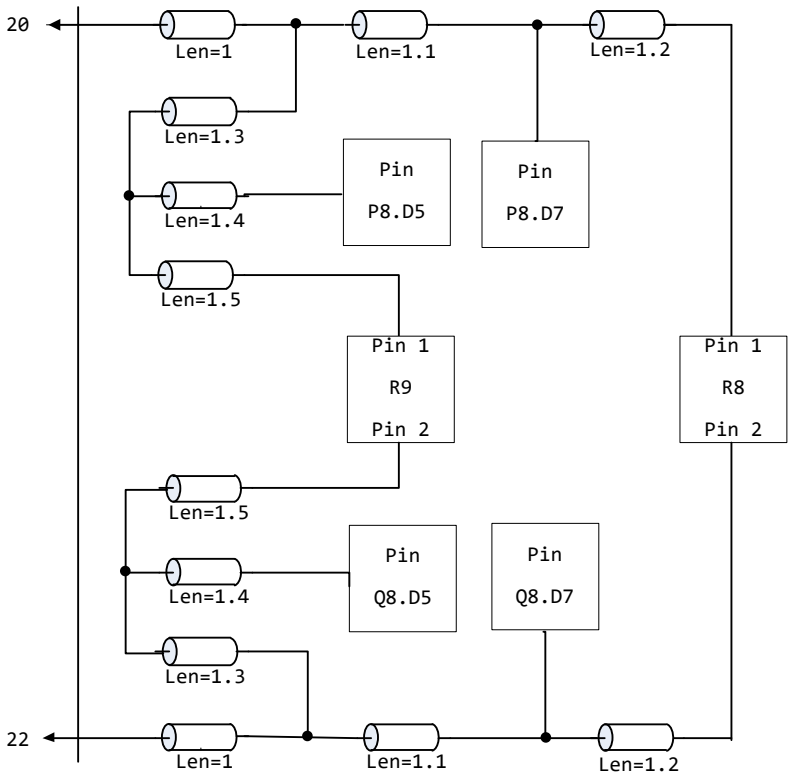


Figure 39 – Paths Connected by Series Resistors as Differential Terminators

Keyword: **[Reference Designator Map]**

Required: Yes, if any of the path descriptions use the Node subparameter

Description: Maps a reference designator to a component or electrical board description contained in a .ibs or .ebd file.

Usage Rules: The [Reference Designator Map] keyword shall be followed by a list of all of the reference designators called out by the Node subparameters used in the various path descriptions. Each reference designator is followed by the file reference of the .ibs or .ebd file containing the electrical description of the component or board, then the name of the component itself as given by the .ibs or .ebd file's [Component] or [Begin Board Description] keyword respectively. The reference designator, file name and component name terms are separated by whitespace. The referenced .ibs or .ebd files shall exist in the same directory as the calling .ebd file or shall exist in a relative path under this directory. It is legal for a reference designator to point to a component that is contained in the calling .ebd file.

The reference designator is limited to ten characters.

Example:

```
[Reference Designator Map]
|
|  External Part References:
|
| ref_des  file_reference   component_name
u23        pp100.ibs        Processor
u24        board/simm.ebd   16Meg X 36 SIMM Module
u25        ls244.ibs        NoName 74LS244a
u26        r10K.ibs         My_10K_Pullup
```

Keyword: **[End Board Description]**

Required: Yes

Description: Marks the end of an Electrical Interconnect Description.

Usage Rules: This keyword must come at the end of each complete electrical There are a number of rules that apply to this combined list description.

Optionally, a comment may be added after the [End Board Description] keyword to clarify which board model has ended.

Example:

```
[End Board Description]        | End: 16Meg X 8 SIMM Module
```

9 NOTES ON DATA DERIVATION METHOD

This section explains how data values are derived. It describes certain assumed parameter and table extraction conditions if they are not explicitly specified. It also describes the allocation of data into the "typ," "min," and "max" columns under variations of voltage, temperature, and process.

The required "typ" column for all data represents typical operating conditions. For most [Model] keyword data, the "min" column describes slow, weak performance, and the "max" column describes the fast, strong performance. It is permissible to use slow, weak components or models to derive the data for the "min" column, and to use fast, strong components or models to derive the data in the "max" columns under the corresponding voltage and temperature derating conditions for these columns. It is also permissible to use typical components or models derated by voltage and temperature and optionally apply proprietary "X%" and "Y%" factors described later for further derating. This methodology has the nice feature that the data can be derived either from semiconductor vendor proprietary models, or typical component measurement over temperature/voltage.

The voltage and temperature keywords, and optionally the process models, control the conditions that define the "typ," "min," and "max" column entries for all I-V table keywords [Pulldown], [Pullup], [GND Clamp], and [POWER Clamp]; all [Ramp] subparameters dV/dt_r and dV/dt_f; and all waveform table keywords and subparameters [Rising Waveform], [Falling Waveform], V_fixture, V_fixture_min, and V_fixture_max.

The voltage keywords that control the voltage conditions are [Voltage Range], [Pulldown Reference], [Pullup Reference], [GND Clamp Reference], and [POWER Clamp Reference]. The entries in the "min" columns contain the smallest magnitude voltages, and the entries in the "max" columns contain the largest magnitude voltages.

The optional [Temperature Range] keyword will contain the temperature which causes or amplifies the slow, weak conditions in the "min" column and the temperature which causes or amplifies the fast, strong conditions in the "max" column. Therefore, the "min" column for [Temperature Range] will contain the lowest value for bipolar models (TTL and ECL) and the highest value for CMOS models. Default values described later are assumed if temperature is not specified.

The "min" and "max" columns for all remaining keywords and subparameters will contain the smallest and largest magnitude values. This applies to the [Model] subparameter C_comp as well, even if the correlation to the voltage, temperature, and process variations are known, because information about such correlation is not available in all cases.

C_comp is considered an independent variable. This is because C_comp includes bonding pad capacitance, which does not necessarily track fabrication process variations. The conservative approach to using IBIS data will associate large C_comp values with slow, weak models, and the small C_comp values with fast, strong models.

The default temperatures under which all I-V tables are extracted are provided below. The same defaults also are stated for the [Ramp] subparameters, but they also apply for the waveform keywords.

The stated voltage ranges for I-V tables cover the most common, single supply cases. When multiple supplies are specified, the voltages shall extend similarly to values that handle practical extremes in reflected wave simulations.

For the [Ramp] subparameters, the default test load and voltages are provided. However, the test load can be entered directly by the R_load subparameter. The allowable test loads and voltages for the waveform keywords are stated by required and optional subparameters; no defaults are needed. Even with waveform keywords, the [Ramp] keyword continues to be required so that the IBIS model remains functional in situations which do not support waveform processing.

The following discussion lists test details and default conditions.

1) I-V Tables:

I-V tables for CMOS models:

typ = typical voltage, typical temp deg C, typical process
min = minimum voltage, max temp deg C, typical process, minus “X%”
max = maximum voltage, min temp deg C, typical process, plus “X%”

I-V tables for bipolar models:

typ = typical voltage, typical temp deg C, typical process
min = minimum voltage, min temp deg C, typical process, minus “X%”
max = maximum voltage, max temp deg C, typical process, plus “X%”

Nominal, min, and max temperature are specified by the semiconductor vendor. The default range is 50 deg C nom, 0 deg C min, and 100 deg C max temperatures.

X% should be statistically determined by the semiconductor vendor based on numerous fabrication lots, test chips, process controls, etc. The value of X need not be published in the .ibs file, and may decrease over time as data on the I/O buffers and silicon process increases.

Temperatures are junction temperatures.

2) Voltage Ranges:

Points for each table must span the voltages listed in Table 17.

Table 17 – Voltage Ranges

Table	Low Voltage	High Voltage
[Pulldown]	GND – POWER	POWER + POWER
[Pullup]	GND – POWER	POWER + POWER
[GND Clamp]	GND – POWER	GND + POWER
[POWER Clamp]	POWER	POWER + POWER
[Series Current]	GND – POWER	GND + POWER
[Series MOSFET]	GND	GND + POWER

As described in the [Pulldown Reference] keyword section, the I-V tables of the [Pullup] and the [POWER Clamp] structures are “Vcc relative”, using the equation:

Vtable = Vcc - Voutput

For example, a model with a 5 V power supply voltage should be characterized between (0 - 5) = -5 V and (5 + 5) = 10 V; and a model with a 3.3 V power supply should be characterized between (0 - 3.3) = -3.3 V and (3.3 + 3.3) = 6.6 V for the [Pulldown] table.

When tabulating output data for ECL type models, the voltage points must span the range of Vcc to Vcc - 2.2 V. This range applies to both the [Pullup] and [Pulldown] tables. Note that this range applies ONLY when characterizing an ECL output.

These voltage ranges must be spanned by the IBIS data. Data derived from lab measurements may not be able to span these ranges as such and so may need to be extrapolated to cover the full range. This data must not be left for the EDA tool to provide.

3) Ramp Rates:

The following steps assume that the default load resistance of 50 ohms is used. There may be models that will not drive a load of only 50 ohms into any useful level of dynamics. In these cases, use the semiconductor vendor’s suggested (nonreactive) load and add the R_load subparameter to the [Ramp] keyword.

The ramp rate does not include packaging but does include the effects of the C_comp parameter; it is the intrinsic output stage rise and fall time only.

The ramp rates (listed in AC characteristics below) should be derived as follows:

a. If starting with the silicon model, remove all packaging. If starting with a packaged model, perform the measurements as outlined below. Then use whatever techniques are appropriate to derive the actual, unloaded rise and fall times.

b. If: The Model_type is one of the following: Output, I/O, or 3-state (not open or ECL types);

   Then: Attach a 50-ohm resistor to GND to derive the rising edge ramp. Attach a 50-ohm resistor to POWER to derive the falling edge ramp.

   If: The Model_type is Output_ECL, I/O_ECL, 3-state_ECL;

   Then: Attach a 50-ohm resistor to the termination voltage (Vterm = VCC - 2 V). Use this load to derive both the rising and falling edges.

   If: The Model_type is either an Open_sink type or Open_drain type;

   Then: Attach either a 50-ohm resistor or the semiconductor vendor suggested termination resistance to either POWER or the suggested termination voltage. Use this load to derive both the rising and falling edges.

   If: The Model_type is an Open_source type;

   Then: Attach either a 50-ohm resistor or the semiconductor vendor suggested termination resistance to either GND or the suggested termination voltage. Use this load to derive both the rising and falling edges.

c. Due to the resistor, output swings will not make a full transition as expected. However, the pertinent data can still be collected as follows:

1. Determine the 20% to 80% voltages of the 50-ohm swing.
2. Measure this voltage change as “dV”.
3. Measure the amount of time required to make this swing “dt”.

d. Post the value as a ratio “dV/dt”. The EDA tool extrapolates this value to span the required voltage swing range in the final model.

e. Typ, Min, and Max must all be posted, and are derived at the same extremes as the I-V tables, which are:

Ramp rates for CMOS models:

typ = typical voltage, typical temp deg C, typical process
min = minimum voltage, max temp deg C, typical process, minus “Y%”
max = maximum voltage, min temp deg C, typical process, plus “Y%”

Ramp rates for bipolar models:

typ = typical voltage, typical temp deg C, typical process
min = minimum voltage, min temp deg C, typical process, minus “Y%”
max = maximum voltage, max temp deg C, typical process, plus “Y%”

where nominal, min, and max temp are specified by the semiconductor vendor. The preferred range is 50 deg C nom, 0 deg C min, and 100 deg C max temperatures.

Note that the derating factor, “Y%”, may be different than that used for the I-V table data. This factor is similar to the X% factor described above. As in the case of I-V tables, temperatures are junction temperatures.

f. During the I-V measurements, the driving waveform should have a rise/fall time fast enough to avoid thermal feedback. The specific choice of sweep time is left to the modeling engineer.

4) Transit Time Extractions:

The transit time parameter is indirectly derived to be the value that produces the same effect as that extracted by the reference measurement or reference simulation. See Figure 40.

The test circuit consists of the following:

a. A pulse source (10 ohms, 1 ns at full duration ramp) or equivalent and transitioning between Vcc and 0 V,

b. A 50-ohm, 1-ns-long trace or transmission line,

c. A 500-ohm termination to the ground clamp reference voltage for TTgnd extraction and to the power clamp reference voltage for TTpower extraction (to provide a convenient, minimum loading 450-ohm-to-50-ohm divider for high-speed sampling equipment observation of the component denoted as the device under test), and

d. The device under test (DUT).

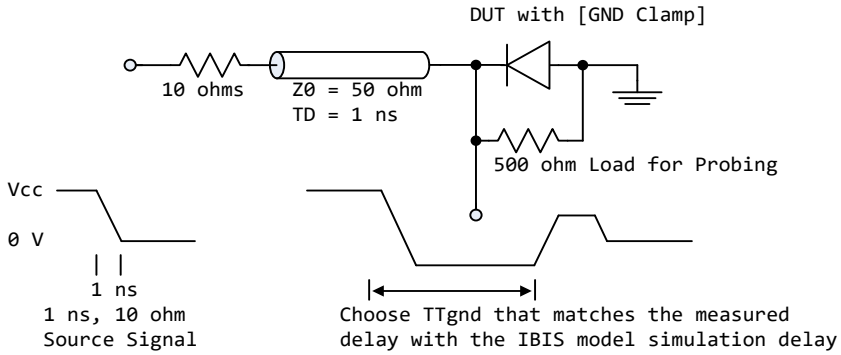


Figure 40 – Example of TTgnd Extraction Setup

The TTgnd extraction will be done only if a [GND Clamp] table exists. A high to low transition that produces a positive "glitch," perhaps several nanoseconds later, indicates a stored charge in the ground clamp circuit. The test circuit is simulated using the complete IBIS model with C_comp and the Ct model defined under the [TTgnd] and [TTpower] keywords. An effective TTgnd value that produces a "glitch" with the same delay is extracted.

Similarly, the TTpower extraction will be done only if a [POWER Clamp] table exists. A low to high transition that produces a negative "glitch," perhaps several nanoseconds later, indicates a stored charge in the power clamp circuit. An effective TTpower value that produces a glitch with the same delay is extracted.

It is preferred to do the extractions with the package parameters removed. However, if the extraction is done from measurements, then the package model should be included in the IBIS based simulation.

5) Series MOSFET Table Extractions:

An extraction circuit is set up according to Figure 41. The switch is configured into the "On" state. This assumes that the Vcc voltage will be applied to the gate by internal logic. Designate one pin of the switch as the source node, and the other pin as the drain node. The table currents designated as Ids are derived directly as a function of the Vs voltage at the source node as Vs is varied from 0 to Vcc. This voltage is entered as a Vgs value as a consequence of the relationship Vtable = Vgs = Vcc - Vs. Vds is held constant by having a fixed voltage Vds between the drain and source nodes. Note, Vds > 0 V. The current flowing into the drain is tabulated in the table for the corresponding Vs points.

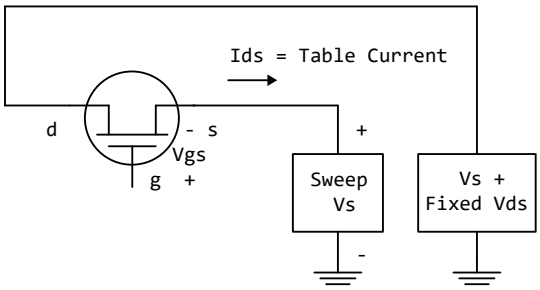


Figure 41 – Example of Series MOSFET Table Extraction

It is expected that this data will be created from semiconductor vendor proprietary silicon models, and later correlated with actual component measurement.

10 ALGORITHMIC MODELING

10.1 ALGORITHMIC MODELING INTERFACE (AMI)

10.1.1 INTRODUCTION

Algorithmic modeling of advanced Serializer-Deserializer (SERDES) devices is supported by IBIS, through the Algorithmic Modeling Interface (AMI; also called IBIS-AMI in this document). The AMI approach breaks SERDES device modeling into two parts – electrical and algorithmic. The combination of the transmitter's analog back-end, the serial channel and the receiver's analog front-end is assumed to be linear and time invariant. There is no limitation that the equalization has to be linear and time invariant. The "analog" portion of the channel is characterized by means of an impulse response leveraging the IBIS constructs for device models defined in Sections 6.1, 6.2 and 6.3.

The transmitter equalization, receiver equalization and clock recovery circuits are assumed to have a high-impedance (electrically isolated) connection to the analog portion of the channel. This makes it possible to model these circuits based on a characterization of the analog channel. The behavior of these circuits is modeled algorithmically through two files:

- an executable model file, which processes the waveforms that characterize the channel
- an AMI parameter definition file, which defines key parameters and parameter ranges used by the executable model file and/or the EDA tool itself for algorithmic modeling

Both of these files are provided by the SERDES device vendor.

Note that the term "DLL" (for Dynamic Link Library) is used throughout this document to refer to the executable model file, regardless of the file name extension actually used on any specific operating system. For example, the executable model file name extension commonly used on Linux-based operating systems is "so".

This section defines how the components of an algorithmic model are specified in a .ibs file.

There are scenarios where receiver and transmitter circuits do not have prior information about their analog channel. Advanced models can perform link training communication to tune the transmitter equalizer parameters for optimized performance and adapt to the signature of any analog channel. This is done when transmitter tap parameters are re-configurable and receivers help them to be configured. Advanced communication specifications such as PCI express, USB, Fibre Channel, and IEEE 802.3 define link training protocols for transmitters and receivers. If both the transmitter and receiver AMI executable models support the same link training protocol (Back-Channel Interface Protocol), the EDA tool will facilitate the communication between the executable models enabling link training. Another name for link training in the industry is Auto-Negotiation.

A link training algorithm can either emulate what the silicon is actually doing, or it can use channel analysis methods to determine the optimal Tx equalization settings. This ability will also allow Rx AMI models to determine the Tx equalizations settings for channels that do not have automatic link training capabilities.

Channels with Repeaters will require that the Downstream Rx be able to control all upstream equalization.

Communications between the Rx and Tx executable models are in messages that both the Rx and Tx executable models understand, and the EDA tool does not need to understand. These agreed upon messages are called a Back-Channel Interface Protocol. This specification does not describe the details of the Back-Channel Interface Protocol but only a method to make the communication work.

This specification describes an underlying mechanism for the AMI .ami file and the executable model to allow information to be transferred from the Tx to the Rx and from the Rx to the Tx without requiring the EDA tool to understand the content of this information, or even for the EDA tool to know that back-channel communications is occurring.

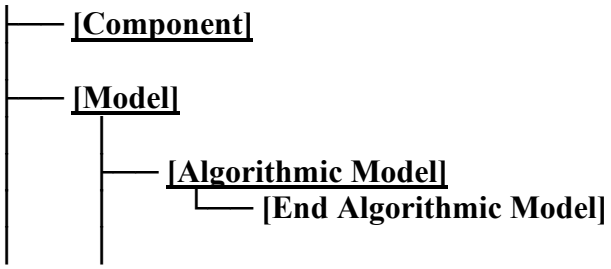
With the information provided in this specification, IC Vendors can develop models that support Link Training in current IBIS-AMI EDA tools.

The structure of the executable model file, methods for passing data to and from the executable model file and how the executable model file is called from the EDA tool are described in Section 10.2. Section 10.3 describes the AMI parameter definition file syntax and usage.

References to algorithmic models may be included in .ibs files using the following keywords:

[Algorithmic Model]
[End Algorithmic Model]

The placement of these keywords within the hierarchy of IBIS is shown below:



10.1.2 KEYWORD DEFINITIONS

Keywords: **[Algorithmic Model], [End Algorithmic Model]**

Required: No

Description: Used to reference an executable model file and accompanying AMI parameter definition file. This executable model file encapsulates signal processing functions, while the AMI parameter definition file includes configuration information for the model and EDA tool. In the case of a receiver, the executable model file may additionally include clock and data recovery functions. The executable model file can receive and modify waveforms from the analog channel, where the analog channel consists of the transmitter output stage, the transmission channel itself and the receiver input stage. This data exchange is implemented through a set of software functions. The signature of these functions is elaborated in Section 10.2 of this document. The function interface must comply with the ANSI "C" language.

Note that, while the file is described here as an "executable model file", the file is a compiled library of functions that may or may not be itself executable.

Sub-Params: Executable, Executable_Rx, Executable_Tx

Usage Rules: The [Algorithmic Model] keyword must be positioned within a [Model] section and it may appear only once for each [Model] keyword in a .ibs file. It is not permitted under the [Submodel] keyword or in [Model]s which are of Model_type Terminator, Series or Series_switch.

The [Algorithmic Model] always processes a single waveform regardless whether the model is single-ended or differential. When the model is differential, the waveform passed to the [Algorithmic Model] must be a difference waveform.

[Algorithmic Model], [End Algorithmic Model]:

Begins and ends an algorithmic model section, respectively.

Executable:

Three entries follow the Executable subparameter on each line:

Platform_Compiler_Bits Executable_Model_File AMI_Parameter_File

The Platform_Compiler_Bits entry provides the name of the operating system, compiler and its version and the number of bits the executable model file is compiled for. It is a string without whitespaces, consisting of three fields separated by an underscore ("_"). The first field consists of the name of the operating system followed optionally by its version. The second field consists of the name of the compiler followed by optionally by its version. The third field is an integer indicating the platform architecture. If the version for either the operating system or the compiler contains an underscore, it shall be converted to a hyphen "-". This is so that an underscore is only present as a separation character in the entry.

The architecture entry can be either "32" or "64". Examples of Platform_Compiler_Bits:

Linux_gcc3.2.3_32
Solaris5.10_gcc4.1.1_64
Solaris_cc5.7_32
Windows_VisualStudio7.1.3088_32
HP-UX_accA.03.52_32

The EDA tool will check for the compiler information and verify if the executable model file is compatible with the platform (operating system and architecture).

Multiple occurrences, without duplication, of Executable are permitted, to support executable model files for many combinations of operating system, architecture, and compiler for the same algorithmic model.

The Executable_Model_File provides the executable model file reference. This shall be an external file. The executable model file shall reside in the same directory as the.ibs file or in a relative path under that directory. See Section 10.2 for details.

The AMI_Parameter_File entry provides the name of the AMI parameter definition file reference, which shall have an extension of "ami". This shall be an external file. The .ami file shall reside in the same directory as the .ibs file or in a relative path under that directory. See Section 10.3, "AMI PARAMETER DEFINITION FILE STRUCTURE", for details.

Executable is prohibited if the Model_type for the associated [Model] is "I/O", "I/O_open_drain", "I/O_open_sink", "I/O_open_source", or "I/O_ECL".

Executable_Rx, Executable_Tx:

The Executable_Rx and Executable_Tx subparameters are alternatives to the Executable subparameter, for I/O-capable buffers. The arguments (fields) supported are syntactically identical to the Executable subparameter. At least one Executable_Rx or one Executable_Tx subparameter is required if the Model_type for the associated [Model] is "I/O", "I/O_open_drain", "I/O_open_sink", "I/O_open_source", or "I/O_ECL". For all other Model_types where [Algorithmic Model] is supported, only the Executable subparameter is permitted. In these cases, the direction of the associated [Algorithmic Model]s shall be assumed by the EDA tool to follow the [Model] Model_type declaration.

It is assumed that the [Model] Model_type, use of [Algorithmic Model] Executable_Rx and/or Executable_Tx subparameters, and the directions of .ami file ReservedParameters are consistent (e.g., that a [Model] of Model_type I/O shall have associated [Algorithmic Model] Executable_Rx and/or Executable_Tx subparameters, each with unique .ami file associations where the .ami files use only Tx-capable and only Rx-capable Reserved Parameters, respectively).

For any given I/O [Model], the [Algorithmic Model] Executable_Rx and Executable_Tx subparameters present shall each refer to unique .ami files (Parameter_Name argument). A single executable may be configured to process both transmit and receive waveform information and so may be used for both directions; unique AMI parameter definition files are required for each direction, however.

The EDA tool is responsible for determining, through interaction with the user, the particular direction and associated files to use for a given simulation.

Examples:

Example of Executable in [Algorithmic Model]:

```
[Algorithmic Model]
| The Model_type for the associated [Model] is something other than "I/O"
| or its variants

Executable Windows_VisualStudio_32 tx_getwave.dll tx_getwave_params.ami
Executable Solaris_cc_32 libtx_getwave.so tx_getwave_params.ami
|
```

```
[End Algorithmic Model]
```

Example of Executable_Rx and Executable_Tx for Bi-directional Model in [Algorithmic Model]:

```
[Algorithmic Model]
| The Model_type for the associated [Model] must be "I/O"
| "I/O_open_drain", "I/O_open_sink", "I/O_open_source", or "I/O_ECL".
|
Executable_Tx Windows_VisualStudio_32 tx_getwave.dll tx_getwave_params.ami
Executable_Tx Solaris_cc_32 libtx_getwave.so tx_getwave_params.ami
|
Executable_Rx Windows_VisualStudio_32 rx_getwave.dll rx_getwave_params.ami
Executable_Rx Solaris_cc_32 libtx_getwave.so rx_getwave_params.ami
|
[End Algorithmic Model]
```

10.2 AMI EXECUTABLE MODEL FILE PROGRAMMING GUIDE

This section is organized as an interface and programming guide for the executable model file referenced by the [Algorithmic Model] keyword described in Section 10.1. Section 10.3 serves as a reference document for the AMI parameter definition file structure for model makers and software engineers.

10.2.1 OVERVIEW

The dynamically-loaded executable model file of a Serializer-Deserializer (SERDES) transmitter or receiver implements an Application Programming Interface (API) containing up to five functions: “AMI_ Resolve”, “AMI_Resolve_Close”, “AMI_Init”, “AMI_GetWave” and “AMI_Close”. The interfaces to these functions are designed to support three different phases of the simulation process: initialization, simulation of a segment of time, and termination of the simulation.

These functions (AMI_Resolve, AMI_Resolve_Close, AMI_Init, AMI_GetWave and AMI_Close) should all be supplied in a single executable model file, and their names and signatures must be as described in this section. The executable model files may reference other dynamic libraries as long as they are part of the operating system, or are delivered together with the executable model file by the model maker. To prevent failures related to dynamic library dependencies, models should be tested on each operating system for which they are released using a dynamic library checking tool such as ldd or Dependency Walker on a “clean” system which has only an operating system installed that is compatible with the AMI models being tested.

The five functions can be included in the executable model file in one of the following four combinations:

Case 1: Executable model file has AMI_Init, AMI_GetWave and AMI_Close.
Case 2: Executable model file has AMI_Init and AMI_Close.
Case 3: Executable model file has AMI_Resolve, AMI_Resolve_Close, AMI_Init, AMI_GetWave and AMI_Close.

Case 4: Executable model file has AMI_Resolve, AMI_Resolve_Close, AMI_Init and AMI_Close."

Please note that the functions AMI_Init and AMI_Close are always required.

The interfaces to these functions are defined from three different perspectives. In addition to specifying the signature of the functions to provide a software coding perspective, anticipated application scenarios provide a functional and dynamic execution perspective, and a specification of the software infrastructure provides a software architecture perspective. Each of these perspectives is required to obtain interoperable software models.

Notes:

1. Throughout this section, terms "long", "double" etc. are used to indicate the data types in the C programming language as published in ISO/IEC 9899-1999.

2. Throughout this section, text strings inside the symbols "<" and ">" should be considered to be supplied or substituted by the model maker. Text strings inside "<" and ">" are not reserved and can be replaced.

10.2.2 APPLICATION SCENARIOS

The next three sections provide an overview of the two simulation types supported for algorithmic models by IBIS. Statistical simulations require that the algorithm in the executable model file is linear and time-invariant (LTI). Time-domain simulations do not have this requirement. Therefore, executable model files used in time-domain simulations may also contain non-linear and/or time-variant (non-LTI) algorithms.

System simulations will commonly involve a transmitter (Tx) and a receiver (Rx) executable model file, each of which may perform filtering in the AMI_Init function, the AMI_GetWave function, or both (i.e., a "dual" algorithmic model). In the case of a "dual" algorithmic model, the filtering functionality in the AMI_Init and AMI_GetWave functions are each intended to be independent representations of the device's equalization. Users of a dual model can elect to use either the AMI_Init or AMI_GetWave filtering functionality, but not combine both simultaneously.

While the primary purpose of the AMI_Init function is to perform the required initialization steps, it may also include LTI signal processing algorithms. Therefore, statistical simulations may be performed using the AMI_Init function alone.

Even though time-domain simulations may also be performed with the LTI AMI_Init and/or LTI AMI_GetWave functions, AMI_GetWave functions containing non-LTI algorithms may only be simulated in the time domain.

An additional flow is provided to illustrate the use of dependent model parameters.

STATISTICAL SIMULATIONS

1. From the system netlist, the EDA tool determines that a given buffer is described by an IBIS [Model].

2. From the IBIS [Model], the EDA tool determines that the buffer is described in part by an [Algorithmic Model].

3. The EDA tool loads the executable model file referenced by [Algorithmic Model], and obtains the addresses of the AMI_Init, AMI_GetWave, and AMI_Close functions.

4. The EDA tool loads the corresponding AMI parameter definition file (.ami file) and assembles the arguments for the AMI_Init function. These arguments include an impulse response matrix, a memory handle for the dynamic memory used by the executable model, the parameters for configuring the algorithmic model, and optionally the impulse response(s) of any crosstalk interferers.

5. The EDA tool calls the AMI_Init function with the arguments previously prepared. The AMI_Init function of the transmitter and receiver [Algorithmic Model]s are called separately as described in the reference flow below.

6. The AMI_Init function parses the configuration parameters, allocates dynamic memory, places the address of the start of the dynamic memory into the memory handle and modifies the impulse response by the filter response of the [Algorithmic Model].

7. The EDA tool completes the rest of the simulation/analysis using the impulse response calculated by the AMI_Init function which is a complete representation of the behavior of a given [Algorithmic Model] combined with the channel.

8. Before exiting, the EDA tool calls the AMI_Close function, giving it the address of the memory handle for the [Algorithmic Model].

9. The AMI_Close function de-allocates the dynamic memory used by the [Algorithmic Model] and performs whatever other clean-up actions are required.

10. The EDA tool terminates execution.

TIME-DOMAIN SIMULATIONS

1. From the system netlist, the EDA tool determines that a given buffer is described by an IBIS [Model].

2. From the IBIS [Model], the EDA tool determines that the buffer is described in part by an [Algorithmic Model].

3. The EDA tool loads the executable model file referenced by [Algorithmic Model], and obtains the addresses of the AMI_Init, AMI_GetWave, and AMI_Close functions.

4. The EDA tool loads the corresponding AMI parameter definition file (.ami file) and assembles the arguments for the AMI_Init function. These arguments include an impulse response matrix, a memory handle for the dynamic memory used by the [Algorithmic Model], the parameters for configuring the [Algorithmic Model], and optionally the impulse response(s) of any crosstalk interferers.

5. The EDA tool calls the AMI_Init function with the arguments previously prepared. The AMI_Init function of the transmitter and receiver [Algorithmic Model]s are called separately as described in the reference flow below.

6. The AMI_Init function parses the configuration parameters, allocates dynamic memory, places the address of the start of the dynamic memory into the memory handle and (optionally) modifies the impulse response by the filter response of the [Algorithmic Model]. The EDA tool may make use of the impulse response returned by the AMI_Init function in its further analysis if needed.

7. The EDA tool generates a time-domain digital input waveform bit pattern (stimulus). A long bit pattern (and simulation) may be broken up into multiple time segments (or blocks) by the EDA tool. For example, if one million NRZ bits are to be simulated, the analysis may be executed in 1000 time segments of 1000 bits each. The segments are not required to be equally sized and are not required to contain an integer number of bits.

8. For each time segment, the EDA tool calls the AMI_GetWave function of the transmitter (if it exists), giving it the digital input waveform and the address of the memory handle for the [Algorithmic Model].

9. For the AMI_GetWave function of the receiver, the EDA tool takes the output from the transmitter AMI_GetWave function (if it exists) and combines it (for example by convolution) with the channel impulse response to produce an analog waveform and passes this result to the receiver AMI_GetWave function for each time segment of the simulation. If the transmitter AMI_GetWave function doesn't exist, the EDA tool takes the output of the transmitter AMI_Init function and combines that (for example by convolution) with the digital stimulus bit pattern to produce the analog waveform for the receiver AMI_GetWave function.

10. The output waveform of the receiver AMI_GetWave function represents the voltage waveform at the decision point of the receiver. The EDA tool completes the simulation/analysis with this waveform.

11. Before exiting, the EDA tool calls the AMI_Close function, giving it the address of the memory handle for the [Algorithmic Model].

12. The AMI_Close function de-allocates the dynamic memory used by the [Algorithmic Model] and performs whatever other clean-up actions are required.

13. The EDA tool terminates execution.

REFERENCE FLOWS

The next two sections define a reference simulation flow for statistical and time-domain system analysis simulations. Other methods of calling models and processing results may be employed, but the final simulation waveforms are expected to match the waveforms produced by this reference simulation flow.

A system simulation usually involves a transmitter (Tx) and a receiver (Rx) model with a passive channel placed between them.

STATISTICAL SIMULATION REFERENCE FLOW

Step 1. The EDA tool obtains the impulse response for the analog channel. This represents the combined impulse response of the transmitter's analog output, the channel and the receiver's analog front end. The transmitter's output or receiver's input characteristics must not include any filtering effects, for example equalization, in this impulse response, although it may include any parasitics which are included in the Tx or Rx analog model.

Step 2. The output of Step 1 is presented to the Tx executable model file's AMI_Init function and the Tx AMI_Init function is executed. The impulse response returned by the Tx AMI_Init function is passed onto Step 3.

Step 3. The output of Step 2 is presented to the Rx executable model file's AMI_Init function and the Rx AMI_Init function is executed. The impulse response returned by the Rx AMI_Init function is passed onto Step 4.

Step 4. The EDA tool completes the rest of the simulation/analysis using the impulse response calculated in Step 3 by the Rx executable model file's AMI_Init function which is a complete representation of the behavior of a given [Algorithmic Model] combined with the channel.

Note that in normal (non-repeater) statistical and time-domain simulations the content of the input impulse_matrix to the Tx's AMI_Init is independent of the Tx's Tx_Impulse_Input parameter value because passing different contents of the input impulse_matrix to the Tx's AMI_Init based on the Tx's Tx_Impulse_Input value always yields identical simulation results.

TIME-DOMAIN SIMULATION REFERENCE FLOW

Step 1. The EDA tool obtains the impulse response for the analog channel. This represents the combined impulse response of the transmitter's analog output, the channel and the receiver's analog front end. The transmitter's output or receiver's input characteristics must not include any filtering effects, for example equalization, in this impulse response, although it may include any parasitics which are included in the Tx or Rx analog model.

Step 2. The output of Step 1 is presented to the Tx executable model file's AMI_Init function and the Tx AMI_Init function is executed. The Tx AMI_Init function may modify the impulse response or choose to leave it unchanged.

Step 3. The output of Step 2 is presented to the Rx executable model file's AMI_Init function and the Rx AMI_Init function is executed. The Rx AMI_Init function may modify the impulse response or choose to leave it unchanged.

Note: The Rx executable model file writer should keep in mind that it is not guaranteed that the impulse response that is presented to the Rx AMI_Init function will always include the effects of the Tx filter. Therefore, the Rx AMI_Init function may not be able to perform accurate optimization under all circumstances. For this reason, the parameters of the Rx AMI_Init function should always default to valid values or have a mechanism to accept user-defined coefficients and allow the user to turn off any automatic optimization routines to ensure successful simulations.

Step 4. The EDA tool produces a digital stimulus waveform. A digital stimulus waveform is 0.5 when the stimulus is "high", -0.5 when the stimulus is "low", and may have a value between -0.5 and 0.5 such that transitions occur when the stimulus crosses 0.

Step 5. If Tx GetWave_Exists is True the output of Step 4 is presented to the Tx executable model file's AMI_GetWave function and the Tx AMI_GetWave function is executed. The output of the Tx AMI_GetWave function is passed on to Step 6.

Step 6a. If Tx GetWave_Exists is True and Rx GetWave_Exists is True, the output of Step 5 is convolved with the output of Step 1 by the EDA tool and the result is passed on to Step 7.

Step 6b. If Tx GetWave_Exists is False and Rx GetWave_Exists is True, the output of Step 4 is convolved with the output of Step 2 by the EDA tool and the result is passed on to Step 7.

Step 6c. If Tx GetWave_Exists is False and Rx GetWave_Exists is False, the output of Step 4 is convolved with the output of Step 3 by the EDA tool and the result is passed on to Step 8.

Step 6d. If Tx GetWave_Exists is True and Rx GetWave_Exists is False, the EDA tool performs one of the two following operations and passes the result to Step 8.

- Not utilize the Tx AMI_GetWave functionality (the output of Step 5 is discarded), by treating the Tx AMI model as if the Tx GetWave_Exists was False. Convolve the output of Step 4 with the output of Step 3.
- Convolve the output of Step 5 with the output of Step 1 and a filter that represents the Rx's gain and equalization, which may be determined using one of the following methods:

    1. Deconvolving the output with the input impulse response of the Rx AMI_Init function.
       Note:
        a. Since the Rx AMI_Init function contains a linear and time invariant algorithm, the Rx gain and equalization may be represented by a filter.
        b. Since the output of the Rx AMI_Init function (output of Step 3) is an impulse response modified by the Rx gain and equalization (e.g., by convolving the input of the Rx AMI_Init function with the impulse response of the Rx filter), the impulse response of the Rx filter may be obtained by deconvolving the output of Step 3 with the input presented to Step 3.
    2. The EDA tool may add an aggressor column that is initialized to a "unit impulse response".
       Note:
        a. If the EDA tool does add an aggressor column that is initialized to a unit impulse response, the tool shall also correspondingly increase the value of the aggressor argument of AMI_Init by one.
        b. A model that uses the crosstalk columns of the impulse_matrix to optimize its equalization shall ignore any column that contains a "unit impulse response" for the purpose of optimizing its equalization.
        c. EDA tools should be aware that a pre-AMI Version 7.2 Rx model may optimize its equalization based on the contents of the aggressor columns of the impulse_matrix and does not ignore unit impulse response columns, rendering method 2 inapplicable.

Note: For the scenario where the Tx AMI_Init function does NOT include equalization effects (i.e., does not modify the impulse response of the channel), Step 6d is functionally equivalent to simply convolving the output of Step 5 with the output of Step 3.

Step 7. If Rx GetWave_Exists is True the output of Step 6 is presented to the Rx executable model file's AMI_GetWave function and the Rx AMI_GetWave function is executed. The output of the Rx AMI_GetWave function is passed on to Step 8.

Step 8. The output of Step 6c, 6d or 7 becomes the simulation waveform output at the Rx decision point. Step 7 optionally may also return clock times, which may be post-processed by the simulation tool or presented to the user as is.

Steps 4 through 8 can be called once or can be called multiple times to process the full analog waveform. Processing the full analog waveform using multiple calls, where each call passes the model a block of samples, reduces the memory requirements when performing long simulations. This allows AMI_GetWave to return model status after processing each block (note that the size of a block is not limited to the number of samples needed to define an integer number of complete symbols). Once all blocks of the input waveform have been processed, Tx AMI_Close and Rx AMI_Close are called to perform any final processing and release allocated memory.

DEPENDENT MODEL PARAMETERS

The usage of the dependent model parameter API is described below.

1. The user selects IBIS model and specifies corner and data rate.
2. The EDA tool initializes AMI_parameters_out to NULL.
3. If Resolve_Exists is False, go to step 9.
4. If Resolve_Exists is True, the EDA tool allocates memory for the AMI_parameters_in string and writes to it name-value pairs of all parameters of Usage type In.
5. The EDA tool calls AMI_Resolve before analog channel impulse characterization.
6. The executable model computes dependent parameter values according to independent parameter values in AMI_parameters_in, symbol_time, corner and model_name.
7. The executable model allocates memory for the AMI_parameters_out string and writes to it name-value pairs of dependent parameters.
8. The EDA tool sets/adjusts analog model parameters if their values are returned by AMI_Resolve in AMI_parameters_out. EDA tool calls AMI_Resolve_Close to release the memory allocated by the executable model in AMI_Resolve.
9. The EDA tool characterizes analog channel impulse responses and finishes the rest of the simulation.

Note that dependent parameters are of Usage Dep, and their values used in the simulation are set by the call to AMI_Resolve before the call to AMI_Init. Values of parameters of Usage InOut returned by the AMI_Init and AMI_GetWave functions shall not affect the dependent parameter values used in the simulation.

The dependent parameter API provides model vendors scalability, extensibility and flexibility to implement dependency relations. It also conceals the dependency formula and allows any complex dependency relation.

Examples:

```
(Rx_model
  (Reserved_Parameters
    (AMI_Version (Usage Info) (Type String) (Value "8.0")
      (Description "This is a v8.0 AMI file."))
    (Resolve_Exists (Usage Info) (Type Boolean) (Value True)
      (Description "Indicates whether the executable model implements
            AMI_Resolve."))
    (Model_Name (Usage In) (Type String) (Value "placeholder")
      (Description "IBIS model name"))
    (Rx_Receiver_Sensitivity (Usage Out) (Type Float) (Range 0.0 0.0 0.01)
      (Description "Value depends on OP_mode and data rate"))
    (Init_Returns_Impulse (Usage Info) (Type Boolean) (Default True)
      (Description "Impulse response is returned"))
    (GetWave_Exists (Usage Info) (Type Boolean) (Default True)
      (Description "GetWave Exists"))
  )
  (Model_Specific
    (my_file (Usage Dep) (Type String) (Value "placeholder")
      (Description "Custom file input.  Value depends on OP_mode"))
    (my_corner (Usage In) (Type String) (Corner "Typ" "Min" "Max")
(Description "Informs the executable model which corner is selected by
  user"))
    (OP_mode (Usage In) (Type Integer) (List 0 1 2 3)
      (Description "Operation mode"))
    …
  )
)
```

In this example, the Rx analog model is represented with a 4-port Touchstone file specified by parameter my_file. Both Rx_Receiver_Sensitivity and my_file depend on the IBIS model name, parameter my_corner, and parameter OP_mode, which specifies the device operation mode. Rx_Receiver_Sensitivity also depends on symbol_time. Parameters Model_Name, my_corner and OP_mode, having usage type In, are included in both input parameter strings to AMI_Resolve and AMI_Init. my_file is of usage type Dep, and its dependency on Model_Name, my_corner and OP_mode is resolved in AMI_Resolve, which returns the value of my_file. Rx_Receiver_Sensitivity is of usage type Out, and its dependency on Model_Name, my_corner, OP_mode and symbol_time is resolved in AMI_Init, which returns the value of Rx_Receiver_Sensitivity.

The AMI parameters and their syntax in the previous example are explained in the sections below.

10.2.3 FUNCTION SIGNATURES

This section defines the structure and parameters used with required and optional functions.

Function: **AMI_Init**

Required: Yes

Declaration: `long AMI_Init (double *impulse_matrix,`

```
long number_of_rows,
long aggressors,
double sample_interval,
double symbol_time,
char *AMI_parameters_in,
char **AMI_parameters_out,
void **AMI_memory_handle,
char **msg)
```

Arguments:

impulse_matrix

"impulse_matrix" points to a memory location where the collection of channel voltage impulse responses, called the "impulse response matrix", is stored in the form of a single dimensional array of floating-point numbers. The impulse responses pointed to by the "impulse_matrix" argument are both input and output. The EDA tool provides the impulse responses. The algorithmic model is expected to modify the impulse responses in place by applying a filtering behavior, for example, an equalization function, if modeled in the AMI_Init function. The impulse response values are uniformly spaced in time. The sample spacing is determined by the EDA tool and passed to the algorithmic model through the AMI_Init function's "sample_interval" argument.

The first column of the impulse response matrix is the impulse response for a through channel, a channel that serves as a communication path between a transmitter/receiver pair. The rest of the columns contain the impulse responses of crosstalk channels. Crosstalk channels describe the paths between aggressor transmitters and victim receiver(s). Transmitters which are not part of a through channel between a certain transmitter/receiver pair are all considered aggressor transmitters with respect to that through channel. The receiver of the through channel in consideration is referred to as the victim receiver. The crosstalk impulse responses may be placed into the impulse response matrix in any order.

If the Reserved Parameter Tx_Impulse_Input is "Separate" then a new column shall be added to the impulse_matrix that shall contain the cumulative impulse response of all upstream models and channels of this Tx.

Note that EDA tools, for AMI models with AMI_Version 7.2 and later, are allowed to determine the model filter impulse response by adding an aggressor column that contains a "unit impulse response" to determine the filter equalization. A unit impulse response contains all zeros except the first value, which shall equal 1.0/sample_interval. Any model that uses the contents of the aggressor columns to optimize its equalization should ignore columns that contain a unit impulse response for the purpose of optimizing its equalization. However, the model should still apply equalization and gain to these columns.

The single dimensional array of "impulse_matrix" is formed by concatenating the columns of an impulse response matrix, starting with the first column and ending with the last column. The matrix elements can be retrieved or identified using the following relationships:

impulse_matrix[idx] = impulse response matrix element (row, col)

where:

- *idx* = col * number_of_rows + row

- *row* is the row index ranging from 0 to number_of_rows-1
- *col* is the column index ranging from 0 to aggressors

Each impulse response in the impulse response matrix must have the same sample spacing and the same length.

To include any crosstalk effects in the Reference Flows described in this section of this specification, the crosstalk impulse responses must be included in the "impulse_matrix" and passed to the transmitter and receiver AMI_Init functions. If present, any filtering in the transmitter and/or receiver AMI_Init function(s) must also be applied to the crosstalk impulse responses to properly account for the crosstalk effects.

Note that the "impulse_matrix" will contain a different set of crosstalk impulse responses for the transmitter and receiver AMI_Init calls, even for a transmitter/receiver pair of the same through channel. A transmitter's AMI_Init function operates on those impulse responses which originate from that transmitter, including the through channel and crosstalk channel impulse responses. A victim receiver's AMI_Init function, on the other hand, operates on all of those impulse responses which are received by that victim receiver, including the through channel and crosstalk channel impulse responses.

As an illustration, consider a crosstalk analysis with five channels numbered 1 through 5, where channel 3 in the center is the through channel of the transmitter/receiver pair Tx3/Rx3, and Rx3 is the victim receiver. In this case channels 1, 2 and 4, 5 are the aggressor channels with the aggressor transmitters Tx1, Tx2, Tx4 and Tx5. If the five "impulse_matrix"-es of the five transmitters' AMI_Init functions contain the following data:

```
************************************
      impulse_matrix impulse_matrix
         column 1       column 2

 Tx1      IR1_1          IR1_3
 Tx2      IR2_2          IR2_3
 Tx3      IR3_3
 Tx4      IR4_4          IR4_3
 Tx5      IR5_5          IR5_3
************************************
```

then the "impulse_matrix" passed into the victim receiver's (Rx3) AMI_Init function will contain the following data:

```
*****************************************************************************
   impulse_matrix impulse_matrix impulse_matrix impulse_matrix impulse_matrix
      column 1       column 2       column 3       column 4       column 5

Rx3 Tx3Init(IR3_3) Tx1Init(IR1_3) Tx2Init(IR2_3) Tx4Init(IR4_3) Tx5Init(IR5_3)

*****************************************************************************
```

where "IRi_j" represents the impulse response from the transmitter on channel i to the receiver on channel j, Tx1Init() .. Tx5Init() represents the output of a transmitter's AMI_Init function which

modified the impulse response denoted inside the parentheses. Note that while in this example the "impulse_matrix" of each transmitter's AMI_Init function contains at most one crosstalk impulse response, the victim receiver's "impulse_matrix" contains four crosstalk impulse responses. Also, using the above notation note that the first index number of each impulse response passed to the transmitter's AMI_Init function matches the transmitter's channel number, and the second index number of each impulse response passed to the receiver's AMI_Init function matches the receiver's channel number.

It is the EDA tool's responsibility to rearrange the content of the "impulse_matrix" between the transmitter and receiver AMI_Init calls.

The EDA tool is also responsible to limit the number of crosstalk channel impulse responses in "impulse_matrix" so that they shall not exceed "Max_Init_Aggressors" as specified in the corresponding AMI parameter definition file of the algorithmic model. Consequently, the "aggressors" parameter of the AMI_Init function shall never contain a greater value than the value provided in "Max_Init_Aggressors" of the corresponding AMI parameter definition file. While the allocated memory space for "impulse_matrix" may be larger, it is assumed that there is no meaningful data in that space beyond the last column of the impulse response matrix that is stored in it.

The AMI_Init function must not change the size or organization of "impulse_matrix" that it was given in any way.

number_of_rows

The number of rows in the impulse_matrix.

aggressors

The number of aggressors in the impulse_matrix.

sample_interval

This is the sampling interval of the "impulse_matrix" passed into the AMI_Init function and the "wave" passed into the AMI_GetWave function. The sample_interval is determined by the EDA tool and it is usually a fraction of the symbol_time. The "impulse_matrix" and "wave" returned by the algorithmic model must have the same "sample_interval" as the original "impulse_matrix" and "wave" that was passed into the algorithmic model. The unit for sample_interval is the second.

Impulse responses in "impulse_matrix" and waveforms in "wave" should be treated as continuous analog waveforms by the algorithmic models. For this reason, algorithmic models must be able to produce valid results at any sample_interval. Any internal analog to digital conversion or resampling is the responsibility of the algorithmic model. In case the algorithmic model is unable to operate at a given sample_interval, it should abort gracefully with an exit code 0 (failure) and appropriate messaging. An illustrative code snippet is shown below:

```
sample_interval = (lowest_symbol_time / 64)
```

symbol_time

symbol_time is the unit interval (UI) of the current data, e.g., 100 ps, 200 ps etc. which equals 1/baud rate. For NRZ signaling, it is equivalent to bit time. The executable model file may use this information along with the impulse_matrix to initialize the filter coefficients. The unit for symbol_time is the second.

AMI_parameters_in

The AMI_parameters_in argument is a pointer to a string. Memory for the string is allocated and de-allocated by the EDA tool. Input parameters from the AMI parameter definition file (i.e., Usage In and Usage InOut parameters) and their associated values are passed to the algorithmic model using a string that has been formatted according to the tree structure defined below.

The AMI_parameters_in argument must always be present in the AMI_Init function call and it must always contain the address of a valid string. The string must always contain at least the root name of the parameter tree from the corresponding AMI parameter definition file, even if there are no parameters to pass to the algorithmic model. It is strongly recommended that the receiving algorithmic model compare this root name to its built-in root name. It is also strongly recommended that the model report mismatches as part of its message string (msg). The model is not required to return Return Value 0 upon exiting.

All requirements documented above for the formatting of parameters of Usage In for the AMI_parameters_in string also apply to parameters defined as Usage InOut when used in the AMI_parameters_in string.

Examples:
Examples of tree structures used for formatting and passing parameters:

```
(mySampleAMI_1)

(mySampleAMI_2
    (tx
      (taps 4)
      (spacing sync)
    )
  )

(mySampleAMI_3
    (branch1
      (leaf1 value1)
      (leaf2 value2)
      (branch2
        (leaf3 value3)
        (leaf4 value4)
      )
      (leaf5 value5 value6 value7)
    )
  )
```

The syntax for the parameter string is:

1. The parameter and message strings passed through the AMI_parameters_in, AMI_parameters_out and msg arguments must not be enclosed in double quotes.
2. Neither names nor individual values except for string literals may contain whitespace characters.
3. Parameter name/value pairs are always enclosed in parentheses, with the value separated from the name by whitespace.
4. A parameter value in a name/value pair can be either a single value or a list of values separated by whitespace.
5. Parameter name/value pairs can be grouped together into parameter groups by starting with an open parenthesis followed by the group name followed by the concatenation of one or more name/value pairs followed by a close parenthesis.
6. Parameter name/values pairs and parameter groups can be freely intermixed inside a parameter group.
7. The top-level parameter string shall be the root name and shall be a parameter group (not a parameter value or a parameter name/value pair).
8. Whitespace is ignored, except as a delimiter between the parameter name and value.
9. Parameter values can be expressed either as a string literal, Boolean literal (True or False), decimal number, or a floating-point number in the standard ANSI “C” notation (e.g., 2.0e-9). String literal values are delimited using a double quote (") and no double quotes are allowed inside the string literals. Empty string literals are denoted by two successive double quote characters.
10. A parameter can be assigned an array of values by enclosing the parameter name and the array of values inside a single set of parentheses, with the parameter name and the individual values all separated by whitespace.

The modified BNF specification for the syntax is:

```
<tree>:
  <branch>

<branch>:
  ( <branch name> <leaf list> )

<leaf list>:
  <branch>
  <leaf>
  <leaf list> <branch>
  <leaf list> <leaf>

<leaf>:
  ( <parameter name> whitespace <value list> )

<value list>:
  <value>
  <value list> whitespace <value>

<value>:
```

```
<string_literal>
<Boolean_literal>
<decimal number>
<decimal number>e<exponent>
<decimal number>E<exponent>
```

AMI_parameters_out

The AMI_parameters_out argument is a pointer to a string pointer. Memory for the string is allocated and de-allocated by the algorithmic model. The model returns a pointer to the string as the contents of this argument. The string must be formatted using the tree structure described in AMI_parameters_in above. The AMI_Init function may use this string to return parameters to the EDA tool.

The AMI_parameters_out argument shall always be present in the AMI_Init function call and the EDA tool must always provide a valid (non-zero) address value in it. The EDA tool must also initialize the memory content at that address to zero (null pointer) prior to calling the AMI_Init function, so that after the execution of the function it can determine whether the function returned a valid string pointer at that address. However, the AMI_Init function shall always return a pointer, at the address provided in this argument, to a string which contains the root name, even if the function does not have any parameters to return to the EDA tool.

The root name shall be the same root name contained within the executable model. Note that the root name shall always be included in the string, regardless of any other contents of that string. For AMI models using AMI_Version 7.2 and later, this executable model root name shall match the root name of the parameter tree from the corresponding AMI parameter definition file. The two root names must be compared by the EDA tool. The EDA tool must report any root name mismatch detected, but may choose to continue or stop simulation at this point.

All requirements documented above for the formatting of parameters of Usage Out for the AMI_parameters_out string also apply to parameters defined as Usage InOut when used in the AMI_parameters_out string.

AMI_memory_handle

Used to point to local storage for the algorithmic block being modeled and shall be passed back during the AMI_GetWave calls. An illustrative code snippet is shown below:

```
my_space = allocate_space( sizeof_space );
status = store_all_kinds_of_things( my_space );
*serdes_memory_handle = my_space;
```

The memory pointed to by AMI_memory_handle is allocated and de-allocated by the model.

msg

The msg argument is a pointer to a string pointer. Memory for the string is allocated and de-allocated by the algorithmic model. The model returns a pointer to the string as the contents of this argument. The AMI_Init function may use this string to send a descriptive, textual message to the EDA tool to be displayed in the user interface and/or to be saved in a log file.

While the msg argument must always be present in the AMI_Init function call and the EDA tool must always provide a valid (non-zero) address value in it, algorithmic models are not required to return anything at that address to the EDA tool. For this reason, the EDA tool must also initialize the memory content at that address to zero (null pointer) prior to calling the AMI_Init function, so that after the execution of the function it can determine whether or not the function returned a valid string pointer at that address. If the AMI_Init function does not have a message string to return to the EDA tool, it may take the following actions:

- ignore the address provided in this argument (leaving the EDA tool provided null pointer at that address)
- return a null pointer at the address provided in this argument (redundantly rewriting the EDA tool provided null pointer at that address)

Return Value

1 for success
0 for failure

Algorithmic models shall return a failure code (0) if and only if the function call fails due to a program execution error. In all other cases the return code shall be "success" (1), even if the function cannot operate properly due to some functional problems. For example, if a function includes a CDR which is unable to get into a stable mode, the function shall still return a success code (1). Examples for returning a failure code (0) may include an invalid data type, a null pointer during run time, or anything that prevents the successful execution of the model's code.

The authors of Algorithmic Models are encouraged to provide feedback to the EDA tool's users through the various available messaging options about any difficulties the model encounters during execution, regardless of what the value of the function's return code is.

Function: **AMI_Impulse**

Required: No, and illegal before AMI_Version 7.1

Declaration:

```
long AMI_Impulse (double *impulse_matrix,
                  char *BCI_parameters_in,
                  char **BCI_parameters_out,
                  char **AMI_parameters_out,
                  void *AMI_memory)
```

Arguments:

impulse_matrix

Same as impulse_matrix argument defined under AMI_Init above.

Note that since both AMI_Init and AMI_Impulse modify the impulse_matrix in place, the EDA tool could maintain the original impulse_matrix and use different memory for the impulse_matrix input to the AMI_Init and AMI_Impulse functions.

Note that the AMI_Impulse function uses the number_of_rows, aggressors, sample_interval, and symbol_time that were passed to the AMI_Init call.

BCI_parameters_in

The BCI_parameters_in argument is a pointer to a string. This pointer is returned in the BCI_parameters_out argument by a previous call to an AMI_Impulse function in another executable model in the channel. Memory for the string is allocated and de-allocated by the previous call to an AMI_Impulse function in another executable model in the channel. The string must be formatted as defined by the BCI_Protocol. On the first call to the primary Tx AMI_Impulse function, this pointer shall be the Null pointer (0).

BCI_parameters_out

The BCI_parameters_out argument is a pointer to a string pointer. Memory for the string is allocated and de-allocated by the algorithmic model. The model returns a pointer to the string as the contents of this argument. The string must be formatted as defined by the BCI_Protocol.

The EDA tool must initialize the memory content at this address to zero (null pointer) prior to calling the AMI_Impulse function, so that after the execution of the function it can determine whether or not the function returned a valid string pointer at that address.

AMI_parameters_out

Same as AMI_parameters_out argument defined under AMI_Init above.

AMI_memory

This is the memory pointer which was allocated during the AMI_Init call.

Return Value

1 for success
0 for failure

Algorithmic models shall return a failure code (0) if and only if the function call fails due to a program execution error. In all other cases the return code shall be "success" (1), even if the function cannot operate properly due to some functional problems. For example, if a function includes a CDR which is unable to get into a stable mode, the function shall still return a success code (1). Examples for returning a failure code (0) may include an invalid data type, a null pointer during run time, or anything that prevents the successful execution of the model's code.

The authors of Algorithmic Models are encouraged to provide feedback to the EDA tool's users through the various available messaging options about any difficulties the model encounters during execution, regardless of what the value of the function's return code is.

Function: **AMI_GetWave**

Required: No

Declaration:
```
long AMI_GetWave (double *wave,
                  long wave_size,
                  double *clock_times,
                  char **AMI_parameters_out,
                  void *AMI_memory)
```

Arguments:

wave

"wave" points to a memory location where a uniformly sampled vector of a time-domain waveform is stored. The waveform pointed to by the "wave" argument is both input and output. The EDA tool provides the wave. The algorithmic model is expected to modify the waveform in place by applying a filtering behavior, for example, an equalization function, if modeled in the AMI_GetWave function. The sample spacing is determined by the EDA tool and passed to the algorithmic model through the AMI_Init function's "sample_interval" argument.

Depending on the EDA tool and the analysis/simulation method chosen, the input waveform could include many components. For example, the input waveform could include:

- The waveform for the primary channel only.
- The waveform for the primary channel plus crosstalk and amplitude noise.
- The output of a time-domain circuit EDA tool such as SPICE.

The sample values are nominally symmetric around zero volts. The algorithmic model's logic threshold may contain a residual non-zero offset, however, that offset will usually be small compared to the input or output differential voltage.

The output waveform is expected to be the waveform at the decision point of the receiver (that is, the point in the receiver where the choice is made as to whether the data bit is a "1" or a "0" for NRZ, or, in the case of PAMn, where the choice is made as to whether the symbol is a "0", "1", … or "n-1"). It is understood that for some receiver architectures, there is no one circuit node which is the decision point for the receiver. In such a case, the output waveform is expected to be the equivalent waveform that would exist at such a node, were it to exist.

wave_size

This is the number of samples in the waveform vector that is in the AMI_GetWave function argument "wave". The length of this waveform is determined by the EDA tool. The value of "wave_size" may be different between AMI_GetWave calls within the same simulation. The "wave" returned by the algorithmic model must have the same number of samples as the original "wave" that was passed into the algorithmic model. Algorithmic models must be able to produce valid results with any wave_size. In case the algorithmic model is unable to operate with a given wave_size, it should abort gracefully with an exit code 0 (failure) and appropriate messaging.

clock_times

Vector to return clock times, or, when Rx_Use_Clock_Input is set to "Times" or "Wave", a vector to input clock times or a clock waveform, respectively. If a model does not have the Rx_Use_Clock_Input parameter, it is considered to have a "single input" Rx AMI_GetWave

function. If a model has the Rx_Use_Clock_Input parameter that is set to “Times” or “Wave”, it is considered to have a “dual input” Rx AMI_GetWave function. The clock times or clock waveform are referenced to the start of the simulation (the first AMI_GetWave call). Memory for clock_times is allocated by the EDA tool and must be large enough to hold the clock times or clock waveform expected during the AMI_GetWave calls. When the clock times are the output of single or dual input Rx AMI_GetWave functions, the sampling times equal clock_times + ½ UI + offset, where offset is defined by Reserved Parameters PAM_Offsets or PAM4_UpperEyeOffset, PAM4_CenterEyeOffset and PAM4_LowerEyeOffset. In the absence of these parameters, offset is assumed to be 0. All the clock time values must be non-negative except the value after the last valid clock time, which shall be -1 to indicate the end of the clock_times vector during each AMI_GetWave call. If an Rx AMI_GetWave only returns -1 during all AMI_GetWave calls then the model does not generate clock times.

Except for Redriver receiver models, it is highly recommended that all receiver models return valid clock times. A receiver model that specifies the Rx_Use_Clock_Input parameter must return valid clock times. The unit of the clock time values is seconds and the unit of the waveform samples is volts.

The clock time values within the clock_times vector should be strictly monotonic, both within the clock_times vector returned from a single call to AMI_GetWave and between successive calls to AMI_GetWave. That is, within a given clock_times vector each successive valid value is greater than the value that preceded it, and the first valid value from a given call to AMI_GetWave must be greater than the last valid value from the preceding call to AMI_GetWave. Any non-strictly-monotonic behavior of clock times (including two identical values) should be considered by the EDA tool as an algorithmic model failure.

Each valid value in the clock_times vector returned by single input Rx AMI_GetWave function calls or dual input Data Rx AMI_GetWave function calls shall be used to sample the output waveform of the AMI_GetWave function by adding ½ UI to the clock times, regardless of whether that waveform sample occurs in the waveform segment being returned by the current call to AMI_GetWave, or in the waveform segment to be returned by the next AMI_GetWave call. Care should be taken in implementation of clock_times to ensure that the calculations used always maintain full double-precision floating-point accuracy across multi-million-symbol simulations.

In SerDes receivers it is possible for the CDR to go out of lock, resulting in clock times that have no definite relationship to the output wave. It is possible for the CDR to be suppressed for an undefined number of symbols until the first clock time value is written to the clock_times vector.

AMI_parameters_out

The AMI_parameters_out argument is a pointer to a string pointer. Memory for the string is allocated and de-allocated by the algorithmic model. The model returns a pointer to the string as the contents of this argument. The string must be formatted using a tree structure, as described in AMI_parameters_in above. The AMI_GetWave function may use this string to return parameters to the EDA tool.

While the AMI_parameters_out argument must always be present in the AMI_GetWave function call and the EDA tool must always provide a valid (non-zero) address value in it, executable model

files are not required to return anything at that address to the EDA tool. For this reason, the EDA tool must also initialize the memory content at that address to zero (null pointer) prior to calling the AMI_GetWave function, so that after the execution of the function it can determine whether or not the function returned a valid string pointer at that address. If the AMI_GetWave function does not have any parameters to return to the EDA tool, it must return a pointer at the address provided in this argument to a string which contains nothing but the root name. Note that the root name must always be included in the string.

AMI_memory

This is the memory which was allocated during the AMI_Init call.

Return Value

1 for success
0 for failure

Executable model files shall return a failure code (0) if and only if the function call fails due to a program execution error. In all other cases the return code shall be "success" (1), even if the function cannot operate properly due to some functional problems. For example, if a function includes a CDR which is unable to get into a stable mode, the function shall still return a success code (1). Examples for returning a failure code (0) may include an invalid data type, a null pointer during run time, or anything that prevents the successful execution of the model's code.

The authors of executable model files are encouraged to provide feedback to the EDA tool's users through the various available messaging options about any difficulties the model encounters during execution, regardless of what the value of the function's return code is.

Function: **AMI_Close**

Required: Yes

Declaration: `long AMI_Close(void * AMI_memory)`

Arguments:

AMI_memory

Same as for AMI_GetWave. See **AMI_GetWave**.

Return Value

1 for success
0 for failure

Executable model files shall return a failure code (0) if and only if the function call fails due to a program execution error. In all other cases the return code shall be "success" (1), even if the

function cannot operate properly due to some functional problems. For example, if a function includes a CDR which is unable to get into a stable mode, the function shall still return a success code (1). Examples for returning a failure code (0) may include an invalid data type, a null pointer during run time, or anything that prevents the successful execution of the model's code.

The authors of executable model files are encouraged to provide feedback to the EDA tool's users through the various available messaging options about any difficulties the model encounters during execution, regardless of what the value of the function's return code is.

Function: **AMI_Resolve**

Required: No, and illegal before AMI_Version 6.1

Declaration:
```
AMI_Resolve (double symbol_time,
       char * AMI_parameters_in,
       char ** AMI_parameters_out)
```

Arguments:

symbol_time

symbol_time is the unit interval (UI) of the current data, e.g., 100 ps, 200 ps, etc. which equals 1/baud rate. For NRZ signaling, it is equivalent to bit time. The unit for symbol_time is the second.

AMI_parameters_in

Input argument. The format and content of this string are the same as that of the AMI_parameters_in argument in AMI_Init.

AMI_parameters_out

Output argument, pointer to a string that contains name-value pairs of dependent parameters of Usage Dep. The format of this string is the same as that of the AMI_parameters_out argument in AMI_Init.

Function: **AMI_Resolve_Close**

Required: No, unless AMI_Resolve exists; illegal before AMI_Version 6.1

Declaration:
```
AMI_Resolve_Close (char * AMI_parameters_out)
```

Arguments:

AMI_parameters_out

The AMI_parameters_out pointer returned by AMI_Resolve.

The Reserved Parameter Tx_Impulse_Input determines the content of the impulse_matrix input to the Tx AMI_Init function and what the AMI_Init function does to the output of the impulse_matrix as described below.

1. The AMI_Init function modifies the impulse response of the through channel in the impulse_matrix in place by applying its gain and equalization to the first column of the impulse_matrix.
2. The AMI_Init function modifies the crosstalk channel columns of the impulse_matrix in place by applying its gain and equalization to the aggressor columns.
3. The content of the input impulse_matrix is determined by the value of parameter Tx_Impulse_Input as described in the parameter definition.

Note when parameter Tx_Impulse_Input is not present, or is “Downstream”, then the normal non-repeater flow is unchanged (except an aggressor unit impulse response may now be added to the impulse matrix).

10.2.4 CODE SEGMENT EXAMPLES

```
extern long AMI_GetWave (wave, wave_size, clock_times, AMI_parameters_out,
AMI_memory);

   my_space = AMI_memory;

    clk_idx = 0;
    time = my_space->prev_time + my_space->sample_interval;
    for(I = 0; I < wave_size; i++)
      {
      wave = filterandmodify(wave, my_space);
      if (clock_times && found_clock (my_space, time))
        clock_times[clk_idx++] = getclocktime (my_space, time);
      time += my_space->sample_interval;
      }
   clock_times[clk_idx] = -1;   //terminate the clock vector
   Return 1;
```

10.3 AMI PARAMETER DEFINITION FILE STRUCTURE

10.3.1 INTRODUCTION

The information provided in this section is applicable to the content of the AMI parameter definition file (also called a .ami file). Note that the rules described below deviate from the rules for .ibs files.

10.3.2 AMI PARAMETER DEFINITION FILE ORGANIZATION

The AMI parameter definition file is organized as a "tree", with "leaves" and "branches" off a single "root" identified by a unique root name. A branch may contain an AMI parameter, which itself contains individual leaves, describing features of the model. Branches, unless otherwise noted, may themselves be used to group other branches.

The file shall contain a distinct section or branch named "Reserved_Parameters" beginning and ending with parentheses. The file may also contain another section or branch named "Model_Specific", beginning and ending with parentheses. "Reserved_Parameters" and "Model_Specific" are the only branches permitted to be connected to the root of the tree.

The AMI parameter definition file shall be organized as shown in the structure below. Note that the first string of non-blank, non-comment characters of the AMI parameter definition file, excluding the opening parenthesis, is the root name.

```
(my_AMIname                    | Root name for AMI parameter definition
                               | file
      (Reserved_Parameters     | Required heading to start the
                               | required Reserved_Parameters
                               | section
            ...
            (Reserved Parameter text starting with AMI_Version)
            ...
      )                        | End of Reserved_Parameters
                               | section
      (Model_Specific          | Required heading to start the
                               | optional Model_Specific section
            ...
            (Model Specific Parameter text)
            ...
      )                        | End of Model_Specific section
      (Description <string>)   | description of the model
                               | (optional)
)                              | End my_AMIname AMI parameter definition
                               | file
```

General Rules:

- The content of the AMI parameter definition file is case sensitive.
- Only the pipe ("|") character is acceptable as a comment character regardless of what the calling .ibs file uses for the comment character.

- The line length of the AMI parameter definition file is not limited to a specific number of characters.
- The root name in the file shall contain an arbitrary non-empty string (with at least one non-white-space character) that does not need to match the file name.
- A branch name under the Model_Specific branch shall contain a non-empty string with at least one non-white-space character. For example, this syntax for parameter xyz preceded by double parenthesis indicating a branch with no name is illegal: ((xyx (Usage Info) (Type Integer) (Value 1))).
- Whitespace in the AMI parameter definition file may be one or more space, tab, and/or line termination characters.
- The "Reserved_Parameters" section is required while the "Model_Specific" section is optional.
- For AMI_Version 5.1 and above, the Reserved_Parameters branch shall appear before the Model_Specific branch. Branches may be in any order in the AMI parameter definition file. The "|" character is the comment character. Any text after the "|" character until the end of the line will be ignored by the parser.
- Scaling factors or suffixes, such as p, n, etc., are not permitted in the AMI parameter definition file.
- Scientific and floating-point notation are permitted.
- Note that Description is considered a leaf that may be optionally used within the individual "Reserved_Parameters" or "Model_Specific" branches. Description is also the only leaf that may be directly connected to the root.
- Leaves may not connect to other leaves, except in the case of Labels for Table (see below).

Note:

1. Throughout this section, text strings inside the symbols "<" and ">" should be considered to be supplied or substituted by the model maker. Text strings inside "<" and ">" are not reserved and can be replaced.

10.3.3 PARAMETER RULES SUMMARY

The features of a model described in an AMI parameter definition file are called AMI parameters, and are grouped into the branches Reserved Parameters and Model Specific Parameters. AMI parameters are themselves branches, but may only contain leaves and not other branches.

Branches may define AMI parameters and/or other branches. A branch which contains one or more sub-branches may only contain the (Description <string>) leaf in addition to the sub-branches. Each sub-branch of a branch must have a unique name.

All AMI parameter branches shall contain descriptor leaf entries formatted as follows:

(<parameter_name>
(Usage <usage>)
(Type <data_type>)
({Format} <data_format> <data>)

(List_Tip) | only with ({Format} List) as discussed below
(Default <value>)
(Description <string>))

AMI parameter branches shall contain the leaves Type, Usage, and any of the following leaves:

Default
<data_format> or Format <data_format>
List_Tip | only with List as discussed below

All leaves of the AMI parameter definition file shall begin with one of the following reserved words:

Type
Usage
Description
Default
<data_format> or Format <data_format>
List_Tip | only with List as discussed below

Multiple leaves containing the same reserved word are not allowed within an AMI parameter branch.

Notes:

1) The order of the leaf entries within an AMI parameter branch is not important.
2) The word Format is optional as indicated by the curly braces "{" and "}" and may be ignored by EDA tools (the examples do not show the word Format).
3) Certain Reserved Parameter names allow only certain <data_format> selections, as described below.
4) The <data_format> selections of Value and Default are always mutually exclusive. Certain parameters may require Value or Default, but Value and Default are not allowed to be present together for the same parameter.
5) <data_format> is always required for selections other than Value.
6) Default is optional for <data_format> Range, List, Corner, Increment and Steps.
7) Default is not allowed for Usage Out parameters.
8) Default is not allowed for <data_format> Table, Gaussian, Dual-Dirac and DjRj.
9) Additional rules apply when <data_format> is Table. The format for <data> describes a set of rows containing data values. Each row has its set of column data values enclosed by parentheses "(" and ")". Each row contains the same number of column values. Any or all of these columns may have different data types. For this case the <data_type> argument is either a list of data types (one for each column), or a single data type. If it is a single data type then this type shall be applied to all of the columns in each row.
10) <data_format> Corner is not allowed for Usage Out.
11) Description is optional.

10.3.4 RESERVED WORD RULES

Usage, Type, Format and Default and their allowed values are reserved names in the parameter definition (.ami) file.

Usage <usage>:

Required for all AMI parameters, where <usage> must be substituted by one of the following:

In

Parameter value is a required input to the AMI model.

Out

Parameter value is coming from the AMI_Init and/or AMI_GetWave functions.

Info

Information for user or EDA tool.

InOut

Parameter value is a required input to the AMI model. The AMI_Init and/or AMI_GetWave functions may return a different value.

Dep

Parameter value is to be assigned by the AMI_Resolve function. Dep is illegal before AMI_Version 6.1.

Note that the purpose of Usage Out or InOut is to provide a mechanism for the algorithmic model to return values to the EDA tool to be used as described by this specification.

Type <data_type> or **Type** <type1><type2>…:

Required, where <data_type> is replaced by one of the entries listed below. For {Format} Table, separate <type1> <type2>… entries are permitted for each column as discussed below, but Type Tap is not permitted.

Float

Float numbers are in general represented by a floating-point number that may be scaled using a decimal exponent. A floating-point number is represented by the significant digits, and optionally a sign and decimal point. For example, -1.23e-3, 123e-3, 1.23, 1 are all of Type Float.

Integer

Integers are numbers which are written without a fractional or decimal component, and fall within -2147483648 and 2147483647. If scientific notation is used, then the exponent must be positive. For example, 65, 7, and -756, 123e3 are integers, but 1.6, 123e99 or 123e-2 are not integers.

String

String is a sequence of ASCII characters enclosed in double quote (") characters (hexadecimal 0x22). As defined in ANSI Standard X3.4-1986, the allowable ASCII characters consist of hexadecimal 0x20, 0x21, 0x23 to 0x7E, and the ASCII control characters 0x09 (HT), 0x0A (LF), and 0x0D (CR) for defining tabs and line termination sequences. The double quote (") character (hexadecimal 0x22) is not allowed inside strings.

Boolean
Acceptable values are True and False, without quotation marks. Boolean Type values are not considered strings.

Tap
(For use by Tx and Rx equalizers)

The type Tap accepts only floating-point values. Note that if the type Tap is used and the parameter value provided is not a number, this shall be considered an error condition for which EDA tool behavior is not specified.

A tapped delay line can be described by creating a separate parameter for each tap weight and grouping all the tap weights for a given tapped delay line in a single parameter group which is given the name of the tapped delay line. If in addition the individual tap weights are each given a name which is their tap number (i.e., "-1" is the name of the first precursor tap, "0" is the name of the main tap, "1" is the name of the first postcursor tap, etc.) and the tap weights are declared to be of type Tap, then the EDA tool can assume that the individual parameters are tap weights in a tapped delay line, and use that assumption to perform tasks such as optimization. The model developer is responsible for choosing whether or not to follow this convention.

A complete equalizer example featuring the Tap Type is provided later in this section.

UI
Unit Interval. 1 UI is the inverse of the symbol rate. For example, 1 UI of 100 ps for an NRZ link transmits 10 Gb/s, while the same 100 ps UI for a PAM4 link transmits 20 Gb/s (or 10 Gsymbols/s). Parameters of Type UI accept values in units of UI (i.e., symbol time). The parameter may take on either floating-point or integer values.

Format <data_format> <data> or **<data_format>**<data>:

Format defines the context or arrangement of the data being presented to the EDA tool. For Usage In and Usage InOut, the EDA tool may accept data provided by the user according to the Format specified in the .ami file. Format is required, except for the <data_format> selection of Value as noted below. The word "Format" as part of the Format <data_format> <data> sequence is optional. Unless otherwise noted, Usage Out arguments are effectively ignored by EDA tools. However, Format may determine how Usage Out data are presented to the user by the EDA tool, particularly when data are returned by the executable model (for example, data of Format Table; see "Table" below). Data of Usage Dep, Usage Info or Usage Out should not be passed to the executable model by the EDA tool, unlike data of Usage In or InOut, which should always be passed to the executable model by the EDA tool.

Valid entries for the <data_format> and <data> fields are:

Value <value>

Value consists of a single value of data. For Usage In and InOut, the model maker may provide any value without any restrictions within the constraints of the Type of the variable. Note that Value and Default (defined below) are mutually exclusive and shall not be used together for the same parameter.

Range <typ value> <min value> <max value>
This defines a continuous range for which the user may select, for Usage In and InOut, any value greater than or equal to <min value> and less than or equal to <max value> within the constraints of the Type of the variable. The signs of typ, min, and max may be positive or negative and the values shall be min <= typ <= max.

List <value> <value> <value> <value> ... <value>
This defines a discrete set of values from which the user may select, for Usage In and InOut, one value. Duplicate values are permitted. The first value shall be assumed to be the default, if the optional selection Default is not present. If the optional selection Default is used with List, the argument to Default must be an explicit member of the List and will override the use of the first List value as default (for example, (List 0 1 2 3 4) implies that the default value for the List is 0, while (List 0 1 2 3 4)(Default 2) means that the default value for the List is 2, not 0).

List_Tip <entry><entry><entry><entry>…<entry>
This is an optional leaf of a parameter with Format **List** and it is followed by a String entry for each entry in the **List**. The number of entries in List_Tip must be the same as the number of entries in **List**. The n^{th} entry in List_Tip shall correspond to the n^{th} entry in **List**. Quoted null entries are not permitted. All entries in List_Tip shall be unique, except that if two entries in **List** are the same, then the corresponding List_Tip entries must also be the same. List is required for List_Tip to be entered, and the word Format before List_Tip as in (Format List_Tip,,,) is not allowed.

Example:

```
(Strength (Usage In) (Type Integer) (Description "Strength of Driver")
      (List 0 1 2 3 4) (Default 2)
      (List_Tip "Extra Weak" "Weak" "Nominal" "Strong" "Extra Strong"))
```

Corner <typ value> <slow value> <fast value>
Corner is not allowed with Usage Out parameters. For Usage In and InOut, the selection of one value is automatically carried out by the EDA tool based on its internal simulation corner setting.

Increment <typ> <min> <max> <delta>
The Increment Format, for Usage In and InOut, defines a set of discrete values not smaller than min and not larger than max, from which the user may select. Those values are defined in increments of delta with respect to typ, as typ + N*delta, where N is any integer (positive, negative, or zero) for which the value satisfies the expression where min <= typ + N*delta <= max.

The sign of delta shall be positive. The signs of typ, min, and max may be positive or negative and the values shall be min <= typ <= max.

Steps <typ> <min> <max> <# steps>
The Steps Format operates exactly like Increment with <delta> == (<max>-<min>)/<# steps>.

Table and optional leaf **Labels**

The Format Table states that this parameter consists of one or more columns of data, with each row delimited by parentheses "(" and ")". All rows must contain the same number of entries (columns). At least one row shall be included. Default is illegal when Format Table is used.

The column entries shall be of Type Float, UI, Integer, String or Boolean.

Type Tap is illegal under Table. If only one Type is provided, then all Table entries shall be of the specified type.

(Type <type>)

For Table only, Type can also be used to designate the entries for each column. In this case, type entries shall be given for each column in the Table:

(Type <type1> <type2> <type3> ...)

Labels is an optional leaf within Table and it is followed by a String entry for each column in the Table. Quoted null entries are permitted. Labels shall be positioned immediately before the first row in a Table and are of the form:

(Labels <"label1"> <"label2"> <"label3"> ...)

If Table is used for a Reserved Parameter, the rules for the number of columns and their meaning are described in the Reserved_Parameters section.

The EDA tool and the executable model file shall always transmit the entire contents of a table through the AMI_parameters_in or AMI_parameters_out string (defined in Section 10.2 and illustrated in the examples below). Only the parameter name and values in the table are included in the parameter string. The values in each row of the table are flattened into a single row of values without the parentheses surrounding each row when producing the parameter string.

For Usage Out and InOut, the number of rows returned by the executable model file may differ from the number of rows documented in the AMI parameter definition file, but a minimum of one row shall be returned. Multiple AMI_GetWave calls are not required to return the same number of rows. For Usage Out, a one-row Table is required in the AMI parameter definition file to serve as a template for single and multi-row tables. This can be used by the EDA tool to reconstruct a sequence of data values returned by the executable model into a table with as many rows as needed, and optionally for parameter initialization before being replaced by the actual Table data returned by the executable model.

Example:

Single Row Table where all numbers are Float (note that "1" is a legal float entry):

```
(fwd (Usage In) (Type Float)
       (Table
        (1 -0.169324  1.40308  0.33024)
      )
      (Description "Application Description")
)
```

The EDA tool sends to the executable model file in the parameter string:

```
(fwd  1 -0.169324  1.40308  0.33024)
```

Single Row, all numbers would be encoded as integers by the EDA tool:

```
(bit_pattern (Usage In) (Type Integer)
       (Table
        (1 1 1 1 0 0 0 1 0 0 1)
      )
      (Description "Bit Pattern Sequence")
)
```

The EDA tool sends to the executable model file in the parameter string:

```
(bit_pattern  1 1 1 1 0 0 0 1 0 0 1)
```

Multiple row Table example with Labels:

The optional Labels line is added above the first row. It is not sent or returned to/from the executable model file, but is available to the EDA tool for information.

```
(poles (Usage InOut) (Type Float)
      (Table
            (Labels "complex_conj_flag" "real_part" "imag_part")
            (1    -5e8        0)
            (2    -9.4e8      8.3e8)
            (1    -7.3e8      0)
      )
      (Description "Two real poles and one complex pole")
)
```

The EDA tool sends to the executable model file in the parameter string:

```
(poles  1  -5e8  0  2  -9.4e8  8.3e8  1  -7.3e8  0)
```

An updated set with a different number of pole and row entries can be returned with a similar sequence to be converted back into the same or a different number of rows.

Type used to specify the type entry for each column (the example above is modified with Type entries for each column):

```
(poles (Usage InOut) (Type Integer Float Float)
      (Table
            (Labels "complex_conj_flag" "real_part" "imag_part")
            (1    -5e8        0)
            (2    -9.4e8      8.3e8)
            (1    -7.3e8      0)
      )
      (Description "Two real poles and one complex poles")
)
```

The encoding in the previous example is sent to the EDA tool and returned to the executable model file.

Example of two rows with Type entries for each column (the fourth column numbers are interpreted as UI values):

```
(pdf (Usage In) (Type Integer Integer Float UI Float)
      (Table
```

```
            (Labels "Row" "Bin number" "Time" "UI" "Probability")
            (1   -5   -5e-9   -1     1e-5)
            (2   -4   -4e-9   -0.8   1e-4)
      )
      (Description "Probability Distribution Function Table")
)
```

The EDA tool sends to the executable model file in the parameter string:

```
(pdf  1  -5  -5e-9  -1  1e-5  2  -4  -4e-9  -0.8  1e-4  ...)
```

Example above, but with Usage Out (only one row is necessary in the AMI parameter definition file):

```
(pdf (Usage Out) (Type Integer Integer Float UI Float)
      (Table
            (Labels "Row" "Bin number" "Time" "UI" "Probability")
            (1   -5   -5e-9   -1     1e-5)
      )
      (Description "Probability Distribution Function Table")
)
```

One row is provided as a template, but the executable model file can return, in the parameter string, different data and more than one row such as shown.

```
(pdf  1  -6  -6e-9  -1.2  3e-6  2  -5  -5e-9  -1  9e-6 ...)
```

Gaussian <mean> <sigma>
Gaussian defines a statistical distribution as the data Format, with mean and sigma (standard deviation) specified by the "mean" and "sigma" floating-point entries, respectively. Gaussian mean and sigma values are assumed to be in units of UI when declared as Type UI. Reserved_Parameters may define units for Gaussian values declared as Type Float, as detailed below.

Dual-Dirac <mean> <mean> <sigma>
Dual-Dirac consists of a composite of two Gaussian data sets. Two separate means are defined, but with a common sigma for each. Both mean entries and the sigma entry are floating-point values and are assumed to be in units of UI when declared as Type UI. Reserved_Parameters may define units for Dual-Dirac values declared as Type Float, as detailed below.

DjRj <minDj> <maxDj> <sigma>
DjRj defines the combination of deterministic jitter (Dj) and random jitter (Rj) values, by convolution. Rj is assumed to take on a Gaussian distribution with standard deviation value "sigma", while Dj is assumed to have a uniform distribution with minimum and maximum values "minDj" and "maxDj", respectively. All entries shall be floating-point, and are assumed to be in units of UI when declared as Type UI and in units of seconds when Type Float.

Default <value>:

When used with single value data, Default and Value are mutually exclusive, and shall not be used together for the same parameter. In these situations, Default is a synonym of Value and does not imply any additional meaning or actions. Default is not allowed for any Usage Out parameter types, and Table, Gaussian, Dual-Dirac and DjRj. Default is optional for Range, List, Corner, Increment and Steps. When Default is specified for any of these parameter types, it shall be used by the EDA tool to pick one value from all the possibilities for that parameter if the user does not make such a selection.

If a Default <value> is specified, its value shall have the same Type as the parameter. For example, if Type is Boolean, <value> shall be either True or False; if Type is Integer, <value> shall be an integer. Also, if Default is specified, <value> shall be a member of the set of allowed values of the parameter. If Default is not specified, the default value of the parameters shall be assumed by the EDA tool to be the <typ> value.

Description <string>:

Description is a leaf that may appear in multiple locations, including after a Model Specific Parameter, after a Reserved Parameter or after the name of the Algorithmic model. The location of Description will determine whether it describes a parameter or the Algorithmic model as a whole.

The string following Description is used by the EDA tool to convey information to the end-user. Description <string> is optional, but it is highly recommended for describing the Algorithmic model and the Model Specific Parameters of the Algorithmic model. The Description string may span multiple lines, but it is recommended that the text contained in the Description string should not exceed 1024 characters per line.

10.3.5 COMBINATION AND CORNER RULES

For Usage Out parameters, ({Format} <data_format> <data>) may be ignored by the EDA tool where not prohibited, except when <data_format> is Table. In this case, a Table of at least one row is required in <data> to serve as a template for single and multi-row tables (see "Format" and "Table" descriptions above).

Formats Value, Corner and List can be of any defined Types whereas Formats Range, Increment and Steps can be of Types Float, UI, Integer and Tap only. Formats Gaussian, Dual-Dirac and DjRj can only be of Types Float and UI. For Format Table, the column entries shall be of Type Float, UI, Integer, String or Boolean. Type Tap is illegal for Format Table. If only one Type is provided, then all Table entries shall be of the specified type. Type can also be used to designate the entries for each column in the table. More information is provided in the definition of the Table format.

Note that modeling and simulating different corner cases is a fundamental concept in IBIS. For each model instance, the EDA tool will make use of either the "Typ", "Min" or "Max" data provided in the .ibs file, according to the user's simulation setup.

As described in Section 9, "NOTES ON DATA DERIVATION METHOD" of this document, the "Min" and "Max" data for the I-V tables and their corresponding voltage reference keywords, [Ramp] and V-T tables represent the slow and fast behavior of the device, respectively. Following the conservative approach, the "Max" value of C_comp represents the slow, and the "Min" value of C_comp represents the fast behavior of the device.

For Usage In and Usage InOut AMI parameters defined as Format Corner, the EDA tool shall pick one of the three supplied values (<typ value>, <slow value>, <fast value>) in the AMI parameter definition file for any given model instance. This selection is governed by the same internal corner variable in the EDA tool that controls the selection of the "Typ", "Min", "Max" model data. <typ value> corresponds to "Typ", <slow value> corresponds to "Min" (slow or weak performance) and <fast value> corresponds to "Max" (fast or strong performance). For AMI parameters, <slow value> does not have to be less than <fast value>.

For AMI parameter Formats "Range", "Increment" and "Steps" <min value>, <max value> does not imply slow and fast corners, and for Usage In and Usage InOut the user may select any value provided by these parameters regardless of what corner is used for the simulation. If the user does not make a selection for parameter Formats "Range", "List", "Increment" and "Steps", the EDA tool shall automatically use the value defined by Default, if it exists, or the <typ value> otherwise (regardless of what corner is used for the simulation).

When a [Model] that is associated with any of the pins listed under the [Diff Pin] keyword contains the [Algorithmic Model] keyword, the tdelay_*** parameters in the fourth, fifth and sixth columns of the [Diff Pin] keyword are ignored in AMI channel characterization simulations, i.e., they are treated as if their value would be zero.

10.3.6 PROCESSING AND PASSING PARAMETER STRING RULES

The parameter string passed in and out of the executable model file (described in the sections AMI_parameters_in, AMI_parameters_out and AMI_memory_handle above) is formatted the same way as the tree data structure in the AMI parameter definition file with the following exceptions.

The EDA tool shall process the content of the AMI parameter definition file such that

1) the "Reserved_Parameters" and "Model_Specific" branch names and their associated open and close parentheses "()" are not included in the AMI_parameters_in string, and
2) the AMI parameter branches with Usage In or Usage InOut are converted to leaves for the AMI_parameters_in string, possibly incorporating user selections. In this conversion each AMI parameter branch name becomes a leaf name in the AMI_parameters_in string and each leaf name is followed by a whitespace character, a value and a closing parenthesis ")"

The executable model shall generate a parameter string that is consistent with the content of the AMI parameter definition file so that

1) the "Reserved_Parameters" and "Model_Specific" branch names and their associated open and close parentheses "()" are not included in the AMI_parameters_out string, and
2) the AMI parameter branches Usage Out or Usage InOut are returned as leaves in the AMI_parameters_out string.

The EDA tool shall pass a string to the executable model through the AMI_parameters_in argument. This string shall contain all of the leaf-formatted Usage In and Usage InOut AMI parameters if there are any defined in the .ami file. No other information may be included in this

string. The string shall always include the root name of the parameter tree, even if there are no parameters to pass to the algorithmic model.

The executable model shall return a string to the EDA tool through the AMI_parameters_out argument. This string shall contain all of the leaf formatted Usage InOut and Usage Out AMI parameters if there are any defined in the AMI parameter definition file. No other information may be included in this string. The string shall always include the root name of the parameter tree, even if there are no parameters to return to the EDA tool.

For Usage In, the value in the AMI parameter leaves are determined by the EDA tool based on the AMI parameter branches in the AMI parameter definition file. For Usage Out, the values in the AMI parameter leaves are determined by the Algorithmic Model. For Usage InOut, the value in the AMI parameter leaves are first determined by the EDA tool based on the AMI parameter branches in the AMI parameter definition file and passed into the Algorithmic Model which may return a new value in the AMI parameter leaves after some processing.

10.3.7 SUMMARY TABLE FOR TYPE AND FORMAT

Table 18 summarizes the relationships between the different Format and Data Types for Reserved or Model Specific Parameters.

Table 18 – Allowable Data Types for Format Values

Format	Data Type					
	Float	**UI**	**Integer**	**String**	**Boolean**	**Tap**
Corner	X	X	X	X	X	X
DjRj	X	X				
Dual-Dirac	X	X				
Gaussian	X	X				
Increment	X	X	X			X
List	X	X	X	X	X	X
Range	X	X	X			X
Steps	X	X	X			X
Table	X	X	X	X	X	
Value	X	X	X	X	X	X

10.4 GENERAL RESERVED PARAMETERS

The AMI parameter definition file shall have a branch with the name "Reserved_Parameters". This branch shall contain all the Reserved Parameters for the model.

The following Reserved Parameters are used by the EDA tool and, unless otherwise noted, are required if the [Algorithmic Model] keyword is present. The entries following the Reserved Parameter names determine their Usage, Type and Default values. Their values may be defined using either Default or Value but not both. Description is optional.

Additional optional Reserved Parameters are defined in separate sections elsewhere in this document.

Parameter: **AMI_Version**

Required: Yes for AMI_Version 5.1 and above, illegal before AMI_Version 5.1

Direction: Rx, Tx

Descriptors:

Usage:	Info
Type:	String
Format:	Value
Default:	<string_literal>
Description:	<string>

Definition: Tells the EDA tool what version of the AMI modeling language is supported.

Usage Rules: AMI_Version is required in the AMI parameter definition files of AMI models which are written in compliance with the IBIS Version 5.1 or later specification(s), but it is not allowed in the AMI parameter definition files of AMI models which are written in compliance with the IBIS Version 5.0 specification. When required, this parameter shall be the first parameter defined in the Reserved_Parameters branch of the AMI parameter definition file.

The value of this parameter shall be "5.1" for AMI models written in compliance with the IBIS Version 5.1 specification, "6.0" for AMI models written in compliance with the IBIS Version 6.0 specification, "6.1" for AMI models written in compliance with the IBIS Version 6.1 specification, "7.0" for AMI models written in compliance with the IBIS Version 7.0 specification, and "7.1" for AMI models written in compliance with the IBIS Version 7.1 specification. The absence of AMI_Version indicates that the AMI model was written in compliance with the IBIS Version 5.0 specification.

The version numbers of .ibs files and AMI models do not have to match. The EDA tool is expected to execute the AMI model according to the rules of the specification which corresponds to its version number.

Other Notes: For AMI_Version 5.1 or later.

Throughout this document, the shorthand, AMI_Version <version_number>, is used to indicate the minimum AMI_Version level that is supported. If the AMI_Version is not used, then the AMI model is processed at the level defined in [IBIS Ver] 5.0. In some cases, it will be noted that a rule has changed, has become more restrictive or more relaxed for a specified AMI_Version level.

Example:

```
(AMI_Version (Usage Info) (Type String) (Value "8.0")
      (Description "Valid for AMI_Version 7.1 and above")
)
```

Parameter: **Init_Returns_Impulse**

Required: Yes

Direction: Rx, Tx

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells the EDA tool whether the AMI_Init function returns a modified impulse response.

Usage Rules: When the Boolean_literal value is set to "True", the model returns the convolution of the impulse response with the impulse response of the equalization.

Other Notes:

Examples:

```
(Init_Returns_Impulse (Usage Info) (Type Boolean) (Default True)
      (Description "Valid for all AMI_Version levels")
)

(Init_Returns_Impulse (Usage Info) (Type Boolean) (Value True)
      (Description "Valid for all AMI_Version levels")
)
```

Parameter: **GetWave_Exists**

Required: Yes

Direction: Rx, Tx

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells the EDA tool whether the AMI_GetWave is implemented in this model.

Usage Rules: Note that if Init_Returns_Impulse is set to "False", then GetWave_Exists shall be set to "True".

Other Notes:

Examples:

```
(GetWave_Exists (Usage Info) (Type Boolean) (Default True)
      (Description "Valid for all AMI_Version levels")
)

(GetWave_Exists (Usage Info) (Type Boolean) (Value True)
      (Description "Valid for all AMI_Version levels")
)
```

Parameter: **Tx_Impulse_Input**

Required: No, and illegal before AMI_Version 7.2

Direction: Tx

Descriptors:

Usage: Info
Type: String
Format: Value
Default: <string_literal>
Description: <string>

Definition: This parameter modifies the content of the impulse_matrix input to AMI_Init (10.2.3 FUNCTION SIGNATURES, AMI_Init). Value must be one of the following: "Downstream", "Combined", "Separate", or "Upstream".

Usage Rules:

If "Downstream":

Column 1 of the impulse_matrix shall contain the impulse response of the model's direct Downstream channel.

If "Combined":

Column 1 of the impulse_matrix shall contain the cumulative impulse response of all upstream models and channels convolved with the Tx direct Downstream channel.

If "Separate":

Column 1 shall contain the impulse response of the model's direct Downstream channel.

Column 'aggressors + 2' shall contain the cumulative impulse response of all upstream models and channels. The model shall not change the output of column 'aggressors + 2' (aggressors is the number of aggressors in the impulse_matrix). For a terminal Tx or Retimer Tx, the upstream impulse response is a unit impulse response.

If "Upstream":

Column 1 of the impulse_matrix shall contain the cumulative impulse response of all preceding models and channels. For a terminal Tx or Retimer Tx, the upstream impulse response is a unit impulse response.

Other Notes: If Tx_Impulse_Input is not present the default shall be "Downstream".

Example:

```
(Tx_Impulse_Input (Usage Info) (Type String) (Value "Downstream")
      (Description "The column 1 of the impulse_matrix shall contain the
      impulse response of the Tx downstream channel"))
```

Parameter: **Use_Init_Output**

Required: No, and legal only before AMI_Version 5.1

Direction: N/A (illegal combination)

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells the EDA tool whether to use AMI_Init output for AMI_GetWave input.

Usage Rules: When Use_Init_Output is set to "True", the EDA tool is instructed to use the output impulse response from the AMI_Init function when creating the input waveform presented to the AMI_GetWave function.

If the Reserved Parameter, Use_Init_Output, is set to "False", EDA tools will use the original (unfiltered) impulse response of the channel when creating the input waveform presented to the AMI_GetWave function.

The algorithmic model is expected to modify the waveform in place.

Use_Init_Output is optional. The default value for this parameter is "True".

If Use_Init_Output is "False", GetWave_Exists shall be "True".

Other Notes:

Example:

```
(Use_Init_Output (Usage Info) (Type Boolean) (Default True)
      (Description "Use_Init_Output is valid only when AMI_Version is omitted")
)
```

The following Reserved Parameters are optional. If the following parameters are not present, the values are assumed as "0".

Parameter: **Max_Init_Aggressors**

Required: No

Direction: Rx, Tx

Descriptors:

Usage: Info
Type: Integer
Format: Value
Default: <numeric_literal>
Description: <string>

Definition: Tells the EDA tool how many aggressor Impulse Responses the AMI_Init function is capable of processing.

Usage Rules: Its value is assumed "0" if Max_Init_Aggressors is not present.

Other Notes:

Examples:

```
(Max_Init_Aggressors (Usage Info) (Type Integer) (Default 5)
      (Description "Valid for all AMI_Version levels")
)

(Max_Init_Aggressors (Usage Info) (Type Integer) (Value 5)
      (Description "Valid for all AMI_Version levels")
)
```

Parameter: **Ignore_Bits**

Required: No

Direction: Rx, Tx

Descriptors:

Usage: Info
Type: Integer
Format: Value
Default: <numeric_literal>
Description: <string>

Definition: Tells the EDA tool how long the time variant model takes to complete initialization.

Usage Rules: This parameter is meant for AMI_GetWave functions that model how equalization adapts to the input stream. The value in this field tells the EDA tool how many UI of the AMI_GetWave output should be ignored.

Its value is assumed "0" if Ignore_Bits is not present.

Other Notes:

Examples:

```
(Ignore_Bits (Usage Info) (Type Integer) (Default 1000)
      (Description "Valid for all AMI_Version levels")
)
```

```
(Ignore_Bits (Usage Info) (Type Integer) (Value 1000)
      (Description "Valid for all AMI_Version levels")
)
```

Parameter: **Resolve_Exists**

Required: No, and illegal before AMI_Version 6.1

Direction: Rx, Tx

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells the EDA tool whether the model implements the AMI_Resolve/AMI_Resolve_Close function pair.

Usage Rules: If omitted, the default is False.

Other Notes: Independent parameters must be of Usage In or InOut. Dependent parameters must be of Usage Dep. Reserved Parameters with allowed usage of Out can have Usage Dep.

Usage Dep is allowed in .ami files in which the parameter "Resolve_Exists" is True.

Usage Dep distinguishes parameters returned by AMI_Resolve, which are of Usage Dep, from those returned by AMI_Init and/or AMI_GetWave, which are of Usage Out or Usage InOut, preventing a parameter from being returned by both AMI_Resolve and AMI_Init/AMI_GetWave.

Example:

```
(Resolve_Exists (Usage Info) (Type Boolean) (Value True)
    (Description "Tells the EDA tool to use AMI_Resolve function")
)
```

Parameter: **Model_Name**

Required: No, and illegal before AMI_Version 6.1

Direction: Rx, Tx

Descriptors:

Usage: In
Type: String
Format: Value
Default: <string_literal>
Description: <string>

Definition: Name of the IBIS [Model] keyword that is being used.

Usage Rules: Value specified in the .ami file is a placeholder and is ignored. The EDA tool must pass the name of the IBIS [Model] keyword that is being instantiated by the EDA tool through the input parameter strings to AMI_Resolve and AMI_Init functions as the value of this parameter.

Other Notes:

Example:

```
(Model_Name (Usage In) (Type String) (Value "placeholder")
   (Description "The name of the instantiated IBIS model")
)
```

Parameter: **Special_Param_Names**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx, Tx

Descriptors:

Usage: Info
Type: String
Format: Table
Default: (Illegal)
Description: <string>

Definition: This Reserved Parameter identifies, by name, all Model_Specific parameters that require EDA tools to perform special handling that is not described in the IBIS specification.

Usage Rules: If the .ami file contains any Model_Specific parameters associated with special operations that the model expects the EDA tool to perform beyond what is described by the IBIS specification, the name of all such Model_Specific parameters shall be listed in this Reserved Parameter.

Other Notes: A non-standard Model_Specific parameter may require action from the user or the EDA tool that is not described in the IBIS specification.

Example:

```
(Special_Param_Names (Usage Info) (Type String)
   (Description "These parameters only work in my favorite simulator.")
   (Table
       ("MyParam1")
       ("MyParam2")
       ("MyParam3")
       ("MyParam4")
   )
)
```

Parameter: **Component_Name**

Required: No, and illegal before AMI_Version 7.1

Direction: Rx, Tx

Descriptors:

Usage: In
Type: String
Format: Value
Default: <string_literal>
Description: <string>

Definition: Name of the IBIS [Component] keyword that is being used.

Usage Rules: The Value specified in the .ami file is a placeholder and is ignored. The EDA tool must pass the name of the IBIS [Component] keyword that is being instantiated by the EDA tool through the input parameter strings to the AMI_Resolve and AMI_Init functions as the value of this parameter.

If the EDA tool is used in such a way that the name of the IBIS [Component] keyword is not available during simulation, then the EDA tool shall pass in an empty string (“”).

Component_Name must be present if Signal_Name is present. Component_Name must be absent if Signal_Name is absent.

Other Notes:

Example:

```
(Component_Name (Usage In) (Type String) (Value "placeholder")
   (Description "The name of the instantiated IBIS Component")
)
```

Parameter: **Signal_Name**

Required: No, and illegal before AMI_Version 7.1

Direction: Rx, Tx

Descriptors:

Usage: In
Type: String
Format: Value
Default: <string_literal>
Description: <string>

Definition: Name of the IBIS [Pin] keyword’s signal_name sub-parameter that is being used.

Usage Rules: The Value specified in the .ami file is ignored. The EDA tool must pass the name of the IBIS [Pin] keyword’s signal_name sub-parameter that is being instantiated by the EDA tool through the input parameter strings to AMI_Resolve and AMI_Init functions as the value of this parameter.

If the EDA tool is used in such a way that the name of the IBIS [Pin] keyword’s signal_name is not available during simulation, then the EDA tool shall pass in an empty string (“”).

Signal_Name must be present if Component_Name is present. Signal_Name must be absent if Component_Name is absent.

Other Notes:

Example:

```
(Signal_Name (Usage In) (Type String) (Value "placeholder")
   (Description "The name of the instantiated IBIS Pin's signal_name sub-
   parameter")
)
```

Parameter: **Rx_Decision_Time**

Required: No, and illegal before AMI_Version 7.1

Direction: Rx

Descriptors:

Usage: Out
Type: Float, UI
Format: Value
Default: <numeric_literal>
Description: <string>

Definition: The AMI_Init model outputs a time value in seconds or UI, which is the receiver decision time of the symbol that the threshold crossing started at with respect to time zero of the impulse response returned by the model. In the AMI_Init flow, this time would represent the ideal sampling time. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules: The EDA tool in the AMI_Init flow uses this information in determining the cursor, precursor, and post cursor locations. Rx_Decision_Time takes precedence over Rx_Clock_Recovery_Mean for statistical (Init) processing.

While it is highly recommended to include the Rx_Decision_Time parameter, if omitted, the EDA tool when in the AMI_Init flow will have to determine the receiver decision time on its own.

Example:

```
(Rx_Decision_Time (Usage Out) (Type Float) (Value 0.0)
      (Description "The time value in seconds, which is the receiver decision
      time of the symbol that the threshold crossing started at with respect to
      time zero of the impulse response returned by the model."))
```

Parameter: **DC_Offset**

Required: No, and illegal before AMI_Version 7.1

Direction: Rx

Descriptors:

Usage: In
Type: Float
Format: Value
Default: <numeric_literal>
Description: <string>

Definition: The input value of DC_Offset is the mean value of the steady state high and low voltages of the analog channel step response at the Rx pad.

Usage Rules: If the impulse response was generated by differentiating the analog channel step response, then the input value of DC_Offset should be the same as the average of the step response initial and final voltages.

It is assumed that the waveform input to the Rx AMI_GetWave function is the physical Rx input waveform minus the input value of this DC_Offset. The Rx AMI_GetWave function may choose to reconstruct the physical input waveform by adding the input value of DC_Offset to the input waveform.

The Rx AMI_GetWave output waveform returned by the AMI model shall swing around zero volts.

Other Notes:

The EDA tool ignores the DC_Offset value specified in the .ami file. It is the responsibility of the EDA tool to determine the input value of DC_Offset. The EDA tool may use any method to do this. The EDA tool may use the input value of DC_Offset to post process data returned by the AMI model to graphically compare the waveform output of Rx AMI_GetWave to the input waveform without the DC_Offset subtracted.

a. For the example below, assume that the EDA tool determines the DC_Offset Value is 0.1 V and then passes it to the executable model
b. Rx AMI_GetWave returns the output waveform in the range from -0.5 V to 0.5 V
c. The EDA tool may shift the output waveform by the DC_Offset Value of 0.1 V to a range from -0.4 V to 0.6 V

Example:

```
(DC_Offset (Usage In) (Type Float) (Value 0.0)
      (Description "The EDA tool is responsible for determining the input value
      sent to the executable model."))
```

Parameter: **Rx_Use_Clock_Input**

Required: No, and illegal before AMI_Version 7.1

Direction: Rx

Descriptors:

Usage:	In
Type:	String
Format:	List, Value
Default:	<string_literal>
Description:	<string>

Definition: Specifies the content of the Data Rx AMI_GetWave clock_times input supported by the model. The three possible content types are: (1) to be ignored, (2) the clock_times and (3) the wave output of the Clock Rx AMI_GetWave. For types (2) and (3) the clock_times input represents the external clock used by the Data Rx AMI_GetWave function. If this parameter is present in the .ami file, the EDA tool is responsible to pass the selected value to the AMI_Init function.

Usage Rules: Allowed values are “None”, “Times” and “Wave”. If omitted, the default is “None”. If “None” is selected, then the content of clock_times will be ignored by the model. If “Times” is selected, then the EDA tool will use the clock_times values that were output by the Clock Rx AMI_GetWave call as the clock_times values in the call to the Data Rx AMI_GetWave. If “Wave” is selected, then the EDA tool will use the wave values that were output by the Clock Rx AMI_GetWave call as the clock_times values in the call to the Data Rx AMI_GetWave.

Other Notes: For the “Wave” option, the Data and Clock input waveforms shall have the same block size and sample_interval, and their sample times shall cover the same time span in each AMI_GetWave call.

For the “Wave” or “Times” option, the Clock shall have an AMI executable model with an AMI_GetWave function. When the Data model’s Rx_Use_Clock_Input parameter includes the “Times” option, the corresponding Clock AMI_GetWave function shall return a clock times vector.

The EDA tool should pass the clock times vector or clock waveform generated by each of the Clock AMI_GetWave function calls into the corresponding dual input Data Rx AMI_GetWave function call without altering it in any way.

Example:

```
(Rx_Use_Clock_Input (Usage In) (Type String) (List "None" "Times")
      (Description "The model can use the Clock AMI_GetWave output clock_times
      to sample the Data waveform at the Rx Data latch"))
```

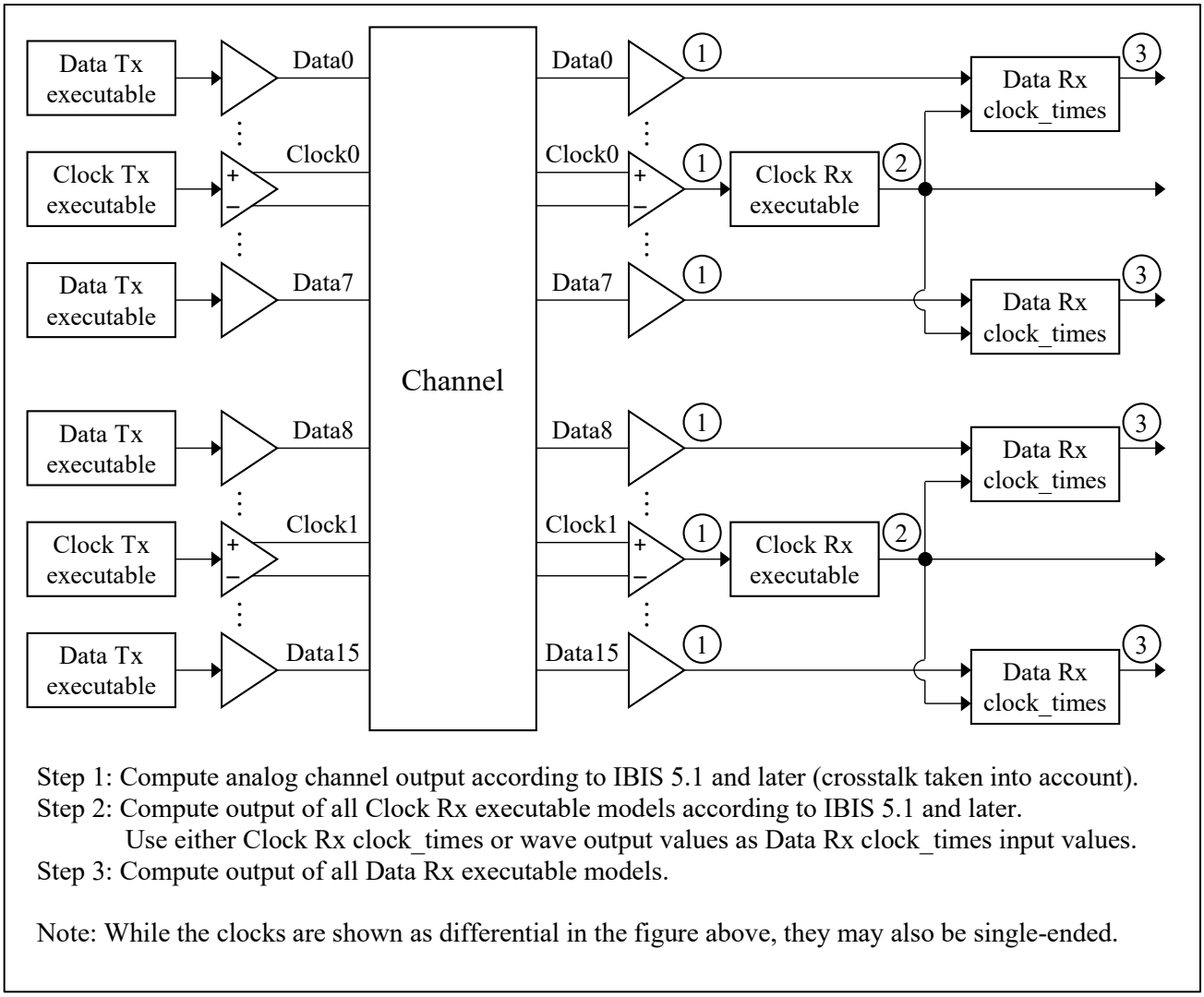


Figure 42 – Examples for Rx_Use_Clock_Input

Parameter: **Adaptation_Valid**
Required: No, and illegal before IBIS 8.0
Direction: Rx
Descriptors:
Usage: Out
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>
Definition: Indicates the EQ tracking status or locking status of an RX AMI, which becomes 1 when the simulation data of that block can be consumed for eye generation.

Usage Rules: This is an optional, Boolean output parameter that will indicate to the EDA tool that the block of bits in question should be ignored or discarded provided the tracking mode is False and consumed for eye accumulation when it becomes True. If the EDA tool chooses to implement this automatic ignoring mechanism, it will start accumulating bits for eye diagram starting with the next block because the model is allowed to report true provided the locking is successful anywhere in the relevant block. EDA tools have the option to ignore this parameter at the expense of potential runtime penalty for any reason including asynchronous locking ambiguities in case multiple RX AMIs are simulated together. Additionally, Ignore_Bits should be set large enough to handle the slowly locking channels. As aforementioned, Ignore_Bits serves as the maximum number of bits to ignore if this automatic ignoring mechanism is implemented.
Example:

```
(Adaptation_Valid (Usage Out) (Type Boolean) (Value False) (Description
"Control automatic ignoring of current block of simulation."))
```

10.4.1 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Table 19 – General Rules and Allowable Usage for General Reserved Parameters

Reserved Parameter	General Rules			Allowable Usage				
	Required	Default[2,3]	Place-holder[4]	Info	In	Out	Dep[1]	InOut
AMI_Version	Yes	--		X				
GetWave_Exists	Yes	--		X				
Tx_Impulse_Input	No	"Downstream"		X				
Ignore_Bits	No	0		X				
Init_Returns_Impulse	Yes	--		X				
Max_Init_Aggressors	No	0		X				
Use_Init_Output	No	True		X				
Resolve_Exists	No	False		X				
Model_Name	No	--	X		X			
Special_Param_Names	No	--		X				
Signal_Name	No[5]	--	X		X			
Component_Name	No[5]	--	X		X			
Rx_Decision_Time	No	--				X		
DC_Offset	No	--	X		X			
Rx_Use_Clock_Input	No	"None"			X			
Adaptation_Valid	No	--				X		

Reserved Parameter	General Rules			Allowable Usage				
	Required	Default[2,3]	Place-holder[4]	Info	In	Out	Dep[1]	InOut

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) "--" means that no default is assumed; an entry must be provided in the .ami file if the parameter is present.
4) In this context, "Placeholder X" means that the EDA tool will supply or calculate an entry for the parameter string to replace the entry found in the .ami file.
5) Either both Component_Name and Signal_Name, or neither, shall be present.

Table 20 – Allowable Data Types for General Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
AMI_Version				X	
GetWave_Exists					X
Tx_Impulse_Input				X	
Ignore_Bits			X		
Init_Returns_Impulse					X
Max_Init_Aggressors			X		
Use_Init_Output					X
Resolve_Exists					X
Model_Name				X	
Special_Param_Names				X	
Component_Name				X	
Signal_Name				X	
Rx_Decision_Time	X	X			
DC_Offset	X				
Rx_Use_Clock_Input				X	
Adaptation_Valid					X

Table 21 – Allowable Data Formats for General Reserved Parameters

Reserved Parameter	Data Format									
	Value	**Range**	**Corner**	**List**	**Increment**	**Steps**	**Gaussian**	**Dual-Dirac**	**DjRj**	**Table**
AMI_Version	X									
GetWave_Exists	X									
Tx_Impulse_Input	X									
Ignore_Bits	X									
Init_Returns_Impulse	X									
Max_Init_Aggressors	X									
Use_Init_Output	X									
Resolve_Exists	X									
Model_Name	X									
Special_Param_Names										X
Component_Name	X									
Signal_Name	X									
Rx_Decision_Time	X									
DC_Offset	X									
Rx_Use_Clock_Input	X			X						
Adaptation_Valid	X									

10.5 RESERVED PARAMETERS FOR DATA MANAGEMENT

Information for simulation involving algorithmic models may be contained in files other than the .ibs file, the AMI parameter definition file, or the executable model file. Parameters related to these other files, called supporting files, are described below.

Parameter: **Supporting_Files**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx, Tx

Descriptors:

Usage:	Info
Type:	String
Format:	Table
Default:	(Illegal)
Description:	<string>

Definition: Supporting_Files contains strings of file names and/or directory names to point to files and/or directories which are used by the AMI executable model directly or by the EDA tool (for example to generate the channel impulse response) to function properly. Supporting_Files is organized as a table containing a single column and one or more rows, in which each file name or directory name entry must be placed into a separate row. The file names or directory names may be written with or without a path, but in either case, they shall be expressed relative to the location of the .ami file in which the Supporting_Files parameter is found. Path separators in the entries of Supporting_Files must be forward slashes "/". Back slashes "\" are not allowed. The EDA tool is responsible for making any operating system-specific adjustments (for example, replacing forward slashes "/" with backslashes "\") if necessary. The last character of this string shall not be a forward slash "/". A Supporting_Files entry may not be an empty string "", or a string containing a period alone ".".

Usage Rules: The purpose of the Supporting_Files parameter is to enumerate all of the supporting files of an AMI model. This is important in situations when the EDA tool needs to know about the supporting files of an AMI model, for example to copy the original model files into its own simulation model library. For this reason, all supporting files of an AMI model must be listed in the Supporting_Files parameter, either using individual file names, or using directory names. When directory names are used in this parameter, it is implied that all of the files and subdirectories in that directory are needed by the AMI model. A file definition is legal but redundant if the directory in which it is located is also defined in a Supporting_Files entry.

Other Notes: The EDA tool is not expected to make wildcard expansions (globbing) for any characters in the string.

Example:

```
(Supporting_Files (Usage Info) (Type String)
   (Description
       "Additional files and directories required by this model")
   (Table
       ("my_stuff_dir")
       ("my_deeper_stuff_dir/here")
```

```
        ("m1.s4p")
        ("my_special_dir/m2.s4p")
    )
)
```

Parameter: **DLL_Path**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx, Tx

Descriptors:

Usage:	In
Type:	String
Format:	Value
Default:	<string literal>
Description:	<string>

Definition: The EDA tool is responsible for recognizing this parameter name and replacing the value declared in the .ami file with a string that contains the path to the directory where the executable model file (called "DLL" here and below) and .ami files reside. The Value specified in the .ami file shall be ignored by the EDA tool. The value of DLL_Path passed to the DLL can either be an absolute path, or a path relative to the current working directory of the process running the DLL. In this string, the path separator is the forward slash "/". Back slashes "\" are not allowed. The model is responsible for making any operating system-specific adjustments (for example, replacing forward slashes "/" with backslashes "\") if necessary.

The last character of the value passed to the DLL shall not be a forward slash "/". To access a supporting file, the DLL should create a file name by creating a string consisting of the value of the DLL path, convert forward slashes "/" to backslashes "\" on operating systems that require a backslash "\" as a path separator, append a forward slash "/" or backslash "\" as appropriate to the operating systems, and then append the name of the file. If the EDA tool chooses to pass a relative path and if the current working directory (CWD) is where the DLL resides then DLL_Path should be a period ".".

Usage Rules:

Other Notes: A DLL should not rely on the current working directory (CWD) set by the EDA tool to determine the locations of files. If DLL_Path is a relative path name then the DLL shall assume that it is a relative path from the CWD, and the EDA tool is responsible for setting the CWD to ensure that the relative DLL_Path is correct. The DLL shall not change the CWD. The EDA tool is not expected to make wildcard expansions (globbing) for any characters in the string.

Example:

```
(DLL_Path (Usage In) (Type String) (Value "placeholder")
          (Description "Path to where the DLL is located"))
```

Parameter: **DLL_ID**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx, Tx

Descriptors:

Usage: In
Type: String
Format: Value
Default: <string literal>
Description: <string>

Definition: The EDA tool is responsible for recognizing this parameter name and replacing the value declared in the .ami file with a string that shall conform to the rules in item 3 of Section 3.2, "SYNTAX RULES". The algorithmic model is responsible for using the DLL_ID string as part of the name for any data files that the model creates, either for use as temporary storage or for recording output data. The use of DLL_ID helps guarantee that multiple instances of the same model (or different models from the same vendor) do not mix up data as a result of collisions between temporary or permanent file names.

Usage Rules:

Example:

```
(DLL_ID (Usage In) (Type String) (Value "placeholder")
       (Description "Unique base name for each AMI model instance and run"))
```

10.5.1 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Tables summarizing the Reserved Parameters for supporting files are shown below.

Table 22 – General Rules and Allowable Usage for Supporting Files Reserved Parameters

Reserved Parameter	General Rules			Allowable Usage				
	Required	Default[2, 3]	Placeholder[4]	Info	In	Out	Dep[1]	InOut
DLL_ID	No	--	X		X			
DLL_Path	No	--	X		X			
Supporting_Files	No	--	X	X				

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) "--" means that no default is assumed; an entry must be provided in the .ami file if the parameter is present.
4) In this context, "Placeholder X" means that the EDA tool will supply or calculate an entry for the parameter string to replace the entry found in the .ami file.

Table 23 – Allowable Data Types for Supporting Files Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
DLL_ID				X	
DLL_Path				X	
Supporting_Files				X	

Table 24 – Allowable Data Formats for Supporting Files Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
DLL_ID	X									
DLL_Path	X									
Supporting_Files										X

10.6 JITTER AND NOISE RESERVED PARAMETERS

Jitter introduced by transmitter and receiver buffers, and the noise sensitivity of the receiver, may be described using AMI Reserved Parameters. These Jitter and Noise parameters are described below.

Note:
If the Jitter and Noise parameters are Usage Info, the EDA tool shall obtain their values from the AMI parameter (.ami) file, optionally through a user interface if user selections are available or needed.

If these parameters are Usage Out, the EDA tool shall use only the values returned by the AMI_Init function, unless otherwise noted. It is the model maker's responsibility to make sure that the AMI_Init function returns the appropriate value in these parameters to the EDA tool to achieve successful simulations.

The model's AMI_GetWave function may also return values in these parameters to the EDA tool, and these values are not required to be the same as the values previously returned by the AMI_Init function. The EDA tool may report the values returned by the AMI_GetWave function to the user, but these values shall not be used by the EDA tool to modify or calculate parameter values passed into simulation models in subsequent function calls or simulations, or to modify or calculate the simulation results in any way.

10.6.1 TX-ONLY RESERVED PARAMETERS

The following optional Reserved Parameters are used to specify impairments for the transmitter output. These budgets specify the impairment as measured at the Tx output (i.e., the transmitter output is expected to be directly modulated by these amounts). These data are used by the EDA tool to either modify the input stimulus presented to the algorithmic model or when post-processing the results from the model; the budget values specified by these parameters are not passed directly to the model itself.

If the parameters are not specified, their default behavior is as summarized in Table 25 and Table 42.

Parameter: **Tx_Jitter**

Required: No

Direction: Tx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Gaussian, Dual-Dirac, DjRj, Table
Default:	(Illegal)
Description:	<string>

Definition: Tells the EDA tool how much jitter exists at the input to the transmitter's analog output buffer.

Usage Rules: For formats Gaussian, Dual-Dirac and DjRj, entries are assumed to be in units of UI when declared as Type UI and in units of seconds when Type Float.

For the Table format, only three table columns are permitted, which shall be entered in the following order:

Row_number Time Probability, or
Row_number UI Probability

where each Row_number is an integer (positive or negative), each Time value is a floating-point number in seconds or a symbol time in units of UI, and each Probability is a unitless floating-point number. The Type for each column must be specified when Format Table is used, as in:

(Type Integer Float Float)
(Type Integer UI Float)

Other Notes: For compatibility with earlier versions, (Type Float) and (Type UI) are permitted for data using the Table format, with Type Float signifying that the three column data types are Integer, Float and Float, and Type UI signifying that the three column data types are Integer, UI and Float. However, these variations are discouraged.

Default is not shown in the examples below.

Examples:

```
(Tx_Jitter (Usage Info) (Type Float)
      (Gaussian 0.2e-12 0.03e-12)
```

```
)

(Tx_Jitter (Usage Info) (Type Float)
      (Dual-Dirac 3e-12 6e-12 0.5e-12)
)

(Tx_Jitter (Usage Info) (Type Float)
      (DjRj 0 6E-12 1.3E-12)
)

(Tx_Jitter (Usage Info) (Type Integer Float Float)
      (Table
      (Labels  "Row_No" "Time" "Probability")
                  (-5  -5e-12  1e-10)
                  (-4  -4e-12  3e-7)
                  (-3  -3e-12  1e-4)
                  (-2  -2e-12  1e-2)
                  (-1  -1e-12  0.29)
                  (0    0      0.4)
                  (1    1e-12  0.29)
                  (2    2e-12  1e-2)
                  (3    3e-12  1e-4)
                  (4    4e-12  3e-7)
                  (5    5e-12  1e-10)
      )
)
```

Parameter: **Tx_DCD**

Required: No

Direction: Tx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, Range, Corner, List, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Tx_DCD (Transmit Duty Cycle Distortion) defines half the peak-to-peak clock duty cycle distortion to be added to the behavior implemented by the EDA tool by modifying the stimulus input or by post-processing the simulation.

$$Time(n) = n * symbol_time + Tx_DCD * (-1.0)^n$$

where:

- n*symbol_time is the ideal time of the nth clock.
- Time(n) is the time of the nth clock modified when creating input waveforms for the Tx.

Entries are assumed to be in units of seconds when declared as Type Float. Note that all equations using jitter parameters that can be defined as UI shall be assumed to seconds in these formulae.

Usage Rules:

Other Notes:

Example:

```
(Tx_DCD (Usage Info) (Type Float)
      (Range 2e-12 1e-12 3e-12))
```

Parameter: **Tx_Rj**

Required: No, and illegal before AMI_Version 6.0

Direction: Tx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The standard deviation of a white Gaussian phase noise process at the transmitter which is to be added to the behavior implemented by the EDA tool by modifying the stimulus input or by post-processing the simulation results. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*Time(n) = n * symbol_time + Tx_Rj * gaussian_rand()*

where gaussian_rand() is a function that returns floating-point numbers between -inf and +inf. The distribution of these numbers shall be a white Gaussian distribution centered at 0.0 with a standard deviation of 1.0. The EDA tool can protect against abs(Tx_Rj*gaussian_rand()) > 0.5UI.

Example:

```
(Tx_Rj (Usage Info) (Corner 0.005 0.006 0.004) (Type UI)
         (Description "Tx Random Jitter in UI."))
```

Parameter: **Tx_Dj**

Required: No, and illegal before AMI_Version 6.0

Direction: Tx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the worst-case peak-to-peak variation at the transmitter implemented by the EDA tool and found by modifying the stimulus input or by post-processing the simulation results. Tx_Dj shall include all deterministic and uncorrelated bounded jitter that is not accounted for by Tx_DCD, and Tx_Sj. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*Time(n) = n * symbol_time + 2.0 * Tx_Dj * rand()*

where rand() is a function that returns floating-point numbers between -0.5 and +0.5 with white uniform distribution.

Example:

```
(Tx_Dj (Usage Info) (Value 0.1) (Type UI)
         (Description "Tx Bounded Jitter in UI."))
```

Parameter: **Tx_Sj**

Required: No, and illegal before AMI_Version 6.0

Direction: Tx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the peak-to-peak amplitude of a sinusoidal jitter which is to be added to the behavior implemented directly by the transmitter model.

Usage Rules: If Tx_Sj_Frequency is not assigned (either in the model or by the user), Tx_Sj should be ignored. Entries are assumed to be in units of seconds when declared as Type Float.

Other Notes: Time is calculated as follows:

*Time(n) = n * symbol_time + Tx_Sj * sin((n * symbol_time * 2.0 * Pi) * Tx_Sj_Frequency)*

Example:

```
(Tx_Sj (Usage Info) (Corner 0.005 0.006 0.004) (Type UI)
         (Description "Tx Sinusoidal Jitter in UI."))
```

Parameter: **Tx_Sj_Frequency**

Required: No, and illegal before AMI_Version 6.0

Direction: Tx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float

Format: Value, List, Range, Corner, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: The frequency, in hertz, of the sinusoidal jitter at the transmitter.

Usage Rules: If Tx_Sj_Frequency is not assigned (either in the model or by the user), Tx_Sj should be ignored.

Other Notes: Time is calculated as follows:

*Time(n) = n * symbol_time + Tx_Sj * sin((n * symbol_time * 2.0 * Pi) * Tx_Sj_Frequency)*

Example:

```
(Tx_Sj_Frequency (Usage Info) (Corner 6.5E7 6.5E7 6.5E7) (Type Float)
         (Description "Tx Sinusoidal Jitter Frequency in Hz."))
```

10.6.2 RX-ONLY RESERVED PARAMETERS

These Reserved Parameters only apply to Rx algorithmic models. These parameters are optional. If the parameters are not specified, their default behavior is as summarized in Table 25 and Table 42.

RECEIVER JITTER RESERVED PARAMETERS

The following optional Reserved Parameters are used to modify the statistics associated with receiver's recovered clock. These parameters are used to account for jitter that is not included in either the clock_times returned by Rx AMI_GetWave or the Rx_Clock_Recovery parameters. These data are used by the EDA tool when post-processing the results from the model; the budget values specified by these parameters are not passed directly to the model itself.

The "Rx Jitter Parameters" below (Rx_DCD, Rx_Rj, Rx_Dj, and Rx_Sj) should be used by the EDA tool when analyzing the output of either Rx AMI_Init (for statistical analysis) or Rx AMI_GetWave (for time-domain analysis).

Parameter: **Rx_DCD**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage: Info, Out, Dep
Type: Float, UI
Format: Value, List, Range, Corner, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: Half of the peak-to-peak variation of a clock duty cycle distortion. This phase noise is to be accounted for by the EDA tool in both Statistical and Time-Domain simulations. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

$$nominal_sample_time = time + Rx_DCD * (-1.0)^n$$

where:

- n is the nth clock.
- time = ideal_time in Statistical, and Time-Domain when clock_times(n) is not available.
- time = clock_times(n) in Time-Domain when clock_times(n) is returned by Rx AMI_GetWave.

Example:

```
(Rx_DCD (Usage Info) (Corner 0.008 0.016 0.005) (Type UI)
         (Description "Rx Duty Cycle Distortion in UI."))
```

Rx_Dj may be used as a repository of all deterministic jitter not included in clock_times. However, any combination of Rx_Dj, Rx_Sj and Rx_DCD is allowed, but the model maker should make sure that jitter components are not double-counted. Total clock recovery deterministic jitter that is not included in the clock_times vector returned by the AMI_GetWave function should be equal to the sum of Rx_Dj, Rx_Sj and Rx_DCD.

Total Clock Recovery Deterministic Jitter not accounted for in clock_times is calculated as follows:

$$\begin{aligned} nominal_sample_time = time &+ 2.0 * Rx_Dj * rand() \\ &+ Rx_Sj * sin(Pi * rand()) \\ &+ Rx_DCD * (-1.0)^n \end{aligned}$$

Parameter: **Rx_Rj**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The standard deviation of a Gaussian phase noise driven by impairments external to the receiver that are input to the Rx CDR, but are not included in the CDR clock_times output. This phase noise is to be accounted for by the EDA tool, in both Statistical and Time-Domain simulations. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*clock_times(n) = time + Rx_Rj * gaussian_rand()*

where:

- time = ideal_time in Statistical, and Time-Domain when clock_times(n) is not available.
- time = clock_times(n) in Time-Domain when clock_times(n) is returned by Rx AMI_GetWave.

Example:

```
(Rx_Rj (Usage Info) (Corner 0.005 0.006 0.004) (Type UI)
         (Description "Rx Random Jitter in UI."))
```

Parameter: **Rx_Dj**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the worst-case peak-to-peak variation of the recovered clock, not including the random jitter specified by Rx_Rj, Rx_Sj, or Rx_DCD. Rx_Dj shall include all deterministic and uncorrelated bounded jitter that is not accounted for by Rx clock_times, Rx_Rj, or Rx_Clock_Recovery parameters. This phase noise is to be accounted for by the EDA tool in both Statistical and Time-Domain simulations. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*nominal_sample_time = time + 2.0 * Rx_Dj * rand()*

where:

- time = ideal_time in Statistical, and Time-Domain when clock_times(n) is not available.
- time = clock_times(n) in Time-Domain when clock_times(n) is returned by Rx AMI_GetWave.

Example:

```
(Rx_Dj (Usage Info) (Value 0.1) (Type UI)
         (Description "Rx Bounded Jitter in UI."))
```

Parameter: **Rx_Sj**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage: Info, Out, Dep
Type: Float, UI
Format: Value, List, Range, Corner, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: Half of the peak-to-peak variation of a sinusoidal phase noise, but are not included in the CDR clock_times output. This phase noise is to be accounted for by the EDA tool in both Statistical and Time-Domain simulations. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*nominal_sample_time = time + Rx_Sj * sin(Pi * rand())*

where:

- time = ideal_time in Statistical, and Time-Domain when clock_times(n) is not available.
- time = clock_times(n) in Time-Domain when clock_times(n) is returned by Rx AMI_GetWave.

Example:

```
(Rx_Sj (Usage Info) (Corner 0.05 0.07 0.04) (Type UI)
         (Description "Rx Sinusoidal Jitter in UI."))
```

RECEIVER RECOVERED CLOCK RESERVED PARAMETERS

The following optional Reserved Parameters are used to specify characteristics of the receiver's recovered clock. These data are used by the EDA tool when post-processing the results from the model when the model does not return clock_times, or when Rx AMI_GetWave is not used; the budget values specified by these parameters are not passed directly to the model itself. For Rx models that do return clock_times by AMI_GetWave, these parameters represent the amount of jitter *that had already been implemented by Rx AMI_GetWave and already included in the returned* clock_times. For this reason, the EDA tool should *not* apply these jitter parameters again to the Rx clock_times. These parameters are provided by the model creator to the EDA tool and end users for the sole purpose that these jitters can be properly accounted for when Rx AMI_GetWave is *not* used or Rx clock_times was not returned, in which cases the EDA tool is responsible to apply these jitters to the Rx output.

Parameter: **Rx_Clock_PDF**

Required: No

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Gaussian, Dual-Dirac, DjRj, Table
Default:	(Illegal)
Description:	<string>

Definition: Tells the EDA tool the probability density function of the recovered clock.

Usage Rules: For formats Gaussian, Dual-Dirac and DjRj, entries are assumed to be in units of UI when declared as Type UI and in units of seconds when Type Float.

For the Table format, only three table columns are permitted, which shall be entered in the following order:

Row_number Time Probability, or
Row_number UI Probability

where each Row_number is an integer (positive or negative), each Time value is a floating-point number in seconds or a symbol time in units of UI, and each Probability is a unitless floating-point number. The Type for each column must be specified when Format Table is used, as in:

(Type Integer Float Float)
(Type Integer UI Float)

Other Notes: For compatibility with earlier versions, (Type Float) and (Type UI) are permitted for data using the Table format, with Type Float signifying that the three column data types are Integer, Float and Float, and Type UI signifying that the three column data types are Integer, UI and Float. However, these variations are discouraged.

Examples:

```
(Rx_Clock_PDF (Usage Info) (Type Float)
      (Gaussian 0.2e-12 0.03e-12)
)

(Rx_Clock_PDF (Usage Info) (Type Float)
      (Dual-Dirac 3e-12 6e-12 0.5e-12)
)

(Rx_Clock_PDF (Usage Info) (Type Float)
      (DjRj 0 6E-12 1.3E-12)
)


(Rx_Clock_PDF (Usage Info) (Type Integer Float Float)
      (Table
      (Labels  "Row_No" "Time" "Probability")
                  (-5  -5e-12  1e-10)
                  (-4  -4e-12  3e-7)
                  (-3  -3e-12  1e-4)
                  (-2  -2e-12  1e-2)
                  (-1  -1e-12  0.29)
                  (0    0      0.4)
                  (1    1e-12  0.29)
                  (2    2e-12  1e-2)
                  (3    3e-12  1e-4)
```

```
                    (4    4e-12  3e-7)
                    (5    5e-12  1e-10)
      )
)
```

Parameter: **Rx_Clock_Recovery_Mean**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: A static offset between the recovered clock and the point half way between the PDF medians of consecutive edge threshold crossing times. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

nominal_sample_time = ideal_time + Rx_Clock_Recovery_Mean

where ideal_time is halfway between the median of the threshold crossing times on both sides of the eye.

Example:

```
(Rx_Clock_Recovery_Mean (Usage Info) (Value 0.05)
         (Type UI) (Description "Recovered Clock offset in UI."))
```

Parameter: **Rx_Clock_Recovery_Rj**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The standard deviation of a Gaussian phase noise exhibited by the recovered clock and included in the clock_times vector returned by the AMI_GetWave function. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*nominal_sample_time = ideal_time + Rx_Clock_Recovery_Rj * gaussian_rand()*

Example:

```
(Rx_Clock_Recovery_Rj (Usage Info) (Corner 0.005 0.006 0.004)
         (Type UI) (Description "Rx Random Clock Jitter in UI."))
```

Parameter: **Rx_Clock_Recovery_Dj**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the worst-case peak-to-peak variation of the recovered clock. Rx_Clock_Recovery_Dj shall include all deterministic and uncorrelated bounded jitter that is included in the clock_times vector returned by the AMI_GetWave function and not accounted for by Rx_Clock_Recovery_DCD and Rx_Clock_Recovery_Sj. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

*nominal_sample_time = ideal_time + 2.0 * Rx_Clock_Recovery_Dj * rand()*

Example:

```
(Rx_Clock_Recovery_Dj (Usage Info) (Value 0.1) (Type UI)
         (Description "Rx Bounded Jitter in UI."))
```

Parameter: **Rx_Clock_Recovery_Sj**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the peak-to-peak variation of a sinusoidal phase noise exhibited by the recovered clock and included in the clock_times vector returned by the AMI_GetWave function. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

$$nominal_sample_time = ideal_time + Rx_Clock_Recovery_Sj * sin(Pi * rand())$$

Example:

```
(Rx_Clock_Recovery_Sj (Usage Info) (Corner 0.05 0.07 0.4) (Type UI)
         (Description "Rx Sinusoidal Jitter in UI."))
```

Parameter: **Rx_Clock_Recovery_DCD**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the peak-to-peak variation of a clock duty cycle distortion exhibited by the recovered clock and included in the clock_times vector returned by the AMI_GetWave function. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

$$nominal_sample_time = ideal_time + Rx_Clock_Recovery_DCD * (-1.0)^n$$

Example:

```
 (Rx_Clock_Recovery_DCD (Usage Info) (Corner 0.008 0.016 0.005)
   (Type UI) (Description "Rx Duty Cycle Distortion in UI."))
```

Rx_Clock_Recovery_Dj may be used as a repository of all deterministic jitter. However, any combination of Rx_Clock_PDF, Rx_Clock_Recovery_Dj, Rx_Clock_Recovery_Sj and Rx_Clock_Recovery_DCD is allowed, but the model maker should make sure that jitter components are not double counted. Total clock recovery deterministic jitter that is included in the clock_times vector returned by the AMI_GetWave function should be equal to the sum of Rx_Clock_PDF, Rx_Clock_Recovery_Dj, Rx_Clock_Recovery_Sj and Rx_Clock_Recovery_DCD.

Total Clock Recovery Deterministic Jitter accounted for in clock_times:

$$\begin{aligned} nominal_sample_time = ideal_time &+ 2.0 * Rx_Clock_Recovery_Dj * rand() \\ &+ Rx_Clock_Recovery_Sj * sin(Pi * rand()) \\ &+ Rx_Clock_Recovery_DCD * (-1.0)^n \\ &+ <deterministic\ contribution\ from\ Rx_Clock_PDF> \end{aligned}$$

RECEIVER NOISE RESERVED PARAMETERS

The following optional Reserved Parameters are used to modify the statistics associated with the data input to the receiver's sampling latch (a.k.a. 'slicer'). These data are used by the EDA tool when post-processing the results from the model; the budget values specified by these parameters are not passed directly to the model itself.

Parameter: **Rx_Receiver_Sensitivity**

Required: No

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float
Format:	Value, Range, Corner, List, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Description: Tells the EDA tool the voltage needed at the receiver data decision point above and below the reference voltage to ensure proper sampling of the equalized signal. The reference voltage is 0 V by default, unless defined by the PAM4_LowerThreshold, PAM4_CenterThreshold, PAM4_UpperThreshold parameters, or the PAM_Thresholds parameter.

Usage Rules: Entries are assumed to be in units of volts. If not present, the value of this parameter is 0 V.

Other Notes: For all formats, the resulting values of this parameter shall be non-negative floating-point numbers.

Examples:

In the first example below, the waveform must rise at least 100 mV above or at least 100 mV below the reference voltage to ensure that the signal is sampled as logic ‘1’ or logic ‘0’, respectively. The remaining examples illustrate Rx_Receiver_Sensitivity with user settable values using various formats.

```
(Rx_Receiver_Sensitivity (Usage Info) (Type Float)
      (Value 0.1))

(Rx_Receiver_Sensitivity (Usage Info) (Type Float)
      (List 0.1 0.05 0.06 0.07 0.08 0.09 0.11))

(Rx_Receiver_Sensitivity (Usage Info) (Type Float)
      (Range 0.2 0.1 0.3))

(Rx_Receiver_Sensitivity (Usage Info) (Type Float)
      (Corner 0.2 0.1 0.3))
```

Parameter: **Rx_Noise, Rx_GaussianNoise**

Required: No, and Rx_Noise is illegal before AMI_Version 6.0; Rx_GaussianNoise is illegal before AMI_Version 7.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The standard deviation, in volts, of an unbounded white Gaussian random process, which is to be added by the EDA tool to the signal measured at the sampling latch of a receiver.

Usage Rules: If Rx_Noise is Usage Out, then the EDA tool shall use the value returned by Rx AMI_Init if Rx AMI_GetWave is not used. If Rx AMI_GetWave is used, then the EDA tool may apply the value returned by each AMI_GetWave call to the waveform returned by that call to AMI_GetWave, or use the average value of Rx_Noise returned by all calls to AMI_GetWave (after Ignore_Bits), or the value of Rx_Noise returned by the last call to AMI_GetWave.

Other Notes: The output voltage waveform is calculated as follows:

$$Output_wave(t) = wave(t) + Rx_Noise * gaussian_rand()$$

where wave(t) is the waveform returned by Rx AMI_GetWave and gaussian_rand() is a function that returns floating-point numbers between -inf and +inf. The distribution of these numbers shall be a white Gaussian distribution centered at 0.0 with a standard deviation of 1.0.

Rx_GaussianNoise is permitted and recommended as an equivalent name for Rx_Noise in AMI_Version 7.0 and higher.

Example:

```
(Rx_Noise (Usage Info) (Value 0.010) (Type Float)
         (Description "Rx amplitude noise at sampling latch in volts."))
```

Parameter: **Rx_UniformNoise**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx

Descriptors:

Usage:	Info, Out, Dep
Type:	Float
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half of the worst-case peak-to-peak variation, in volts, of a bounded uniform random process, which is to be added by the EDA tool to the signal measured at the sampling latch of a receiver.

Usage Rules: If Rx_UniformNoise is Usage Out, then the EDA tool shall use the value returned by Rx AMI_Init if Rx AMI_GetWave is not used. If Rx AMI_GetWave is used, then the EDA tool may apply the value returned by each AMI_GetWave call to the waveform returned by that call to AMI_GetWave, or use the average value of Rx_UniformNoise returned by all calls to AMI_GetWave (after Ignore_Bits), or the value of Rx_UniformNoise returned by the last call to AMI_GetWave.

Other Notes: The output voltage waveform is calculated as follows:

*Output_wave(t) = wave(t) + 2 * Rx_UniformNoise * rand()*

where wave(t) is the waveform returned by Rx AMI_GetWave and rand() is a function that returns floating-point numbers between -0.5 and +0.5 with white uniform distribution.

Example:

```
(Rx_UniformNoise (Usage Info) (Value 0.010) (Type Float)
         (Description "Rx uniform amplitude noise at sampling latch in
         volts."))
```

10.6.3 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Tables summarizing the rules for the jitter, noise, and sensitivity Reserved Parameters are shown below.

Table 25 – General Rules and Allowable Usage for Jitter and Noise Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage				
	Required	Default[2]	Info	In	Out	Dep[1]	InOut
Rx_Clock_PDF	No	0	X		X	X	
Rx_Clock_Recovery_DCD	No	0	X		X	X	
Rx_Clock_Recovery_Dj	No	0	X		X	X	
Rx_Clock_Recovery_Mean	No	0	X		X	X	
Rx_Clock_Recovery_Rj	No	0	X		X	X	
Rx_Clock_Recovery_Sj	No	0	X		X	X	
Rx_DCD	No	0	X		X	X	
Rx_Dj	No	0	X		X	X	
Rx_Noise, Rx_GaussianNoise	No	0	X		X	X	
Rx_UniformNoise	No	0	X		X	X	
Rx_Receiver_Sensitivity	No	0	X		X	X	
Rx_Rj	No	0	X		X	X	

Reserved Parameter	General Rules		Allowable Usage				
	Required	Default[2]	Info	In	Out	Dep[1]	InOut
Rx_Sj	No	0	X		X	X	
Tx_DCD	No	0	X		X	X	
Tx_Dj	No	0	X		X	X	
Tx_Jitter	No	0	X		X	X	
Tx_Rj	No	0	X		X	X	
Tx_Sj	No	0	X		X	X	
Tx_Sj_Frequency	No	(see Usage Rules)	X		X	X	

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".

Table 26 – Allowable Data Types for Jitter and Noise Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Rx_Clock_PDF	X	X			
Rx_Clock_Recovery_DCD	X	X			
Rx_Clock_Recovery_Dj	X	X			
Rx_Clock_Recovery_Mean	X	X			
Rx_Clock_Recovery_Rj	X	X			
Rx_Clock_Recovery_Sj	X	X			
Rx_DCD	X	X			
Rx_Dj	X	X			
Rx_Noise, Rx_GaussianNoise	X				
Rx_UniformNoise	X				
Rx_Receiver_Sensitivity	X				
Rx_Rj	X	X			
Rx_Sj	X	X			
Tx_DCD	X	X			
Tx_Dj	X	X			
Tx_Jitter	X	X			
Tx_Rj	X	X			
Tx_Sj	X	X			

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Tx_Sj_Frequency	X				

Table 27 – Allowable Data Formats for Jitter and Noise Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
Rx_Clock_PDF							X	X	X	X
Rx_Clock_Recovery_DCD	X	X	X	X	X	X				
Rx_Clock_Recovery_Dj	X	X	X	X	X	X				
Rx_Clock_Recovery_Mean	X	X	X	X	X	X				
Rx_Clock_Recovery_Rj	X	X	X	X	X	X				
Rx_Clock_Recovery_Sj	X	X	X	X	X	X				
Rx_DCD	X	X	X	X	X	X				
Rx_Dj	X	X	X	X	X	X				
Rx_Noise, Rx_GaussianNoise	X	X	X	X	X	X				
Rx_UniformNoise	X	X	X	X	X	X				
Rx_Receiver_Sensitivity	X	X	X	X	X	X				
Rx_Rj	X	X	X	X	X	X				
Rx_Sj	X	X	X	X	X	X				
Tx_DCD	X	X	X	X	X	X				
Tx_Dj	X	X	X	X	X	X				
Tx_Jitter							X	X	X	X
Tx_Rj	X	X	X	X	X	X				
Tx_Sj	X	X	X	X	X	X				
Tx_Sj_Frequency	X	X	X	X	X	X				

With the exception of the "Table" format, the Tx_Jitter parameter has been essentially superseded by the Reserved_Parameters Tx_Rj, Tx_Dj, Tx_Sj, Tx_Sj_Frequency, and Tx_DCD, which enable SerDes transmitter jitter to be specified in greater detail. It is recommended for AMI model developers to use these preferred jitter parameters when possible instead of Tx_Jitter. With the exception of the "Table" format, the Rx_Clock_PDF parameter has been essentially superseded by the Reserved_Parameters Rx_Clock_Recovery_Rj, Rx_Clock_Recovery_Dj,

Rx_Clock_Recovery_Sj, and Rx_Clock_Recovery_DCD, which enable SerDes receiver jitter to be specified in greater detail. It is recommended for AMI model developers to use these preferred jitter parameters when possible instead of Rx_Clock_PDF.

10.7 MODULATION RESERVED PARAMETERS

Prior to AMI_Version 6.1, AMI modeling supported only NRZ SerDes signaling. AMI_Version 6.1 introduced support for four-level Pulse Amplitude Modulation (PAM4) SerDes signaling. In addition, AMI_Version 7.2 introduced Modulation_Levels, PAM_Thresholds, and PAM_Offsets to support PAMn SerDes signaling. Since PAMn is a superset of PAM4, the new PAMn parameters are intended to replace the older PAM4 parameters. Therefore, it is highly recommended to use the PAMn parameters for PAM4 signaling instead of the older PAM4 equivalents. Note that no parameter analogous to PAM4_Mapping has been defined for PAMn. The intent is to leave mapping entirely to the EDA tool.

Parameter: **Modulation**

Required: No, and illegal before AMI_Version 6.1

Direction: Rx, Tx

Descriptors:

Usage:	Info, In
Type:	String
Format:	Value, List
Default:	<string_literal>
Description:	<string>

Definition: Tells the EDA tool whether NRZ or PAM4 analysis is to be performed.

Usage Rules: Modulation and Modulation_Levels shall not both be present. This Reserved Parameter tells the EDA tool (and optionally, the algorithmic model) of the modulation scheme to be used for analysis. It is declared as Type String with two pre-defined values of "NRZ" and "PAM4". Valid entries for this parameter are "NRZ" and "PAM4". The default "NRZ" applies if the Modulation parameter is not included in the .ami file.

The Modulation parameter controls how the EDA tool prepares the Stimulus waveform for AMI_GetWave-based analysis and how it post-processes simulation results:

- When Modulation is set to "NRZ", the EDA tool prepares the input stimulus using -0.5V to represent a logic 0 and 0.5V to represent a logic 1. The Rx Parameter Rx_Receiver_Sensitivity is used to post-process Rx model data.
- When Modulation is set to "PAM4", the EDA tool prepares the input stimulus using voltage levels of -0.5, -0.166, 0.166 and 0.5 volts to represent PAM4 symbols 0, 1, 2 and 3 respectively. The conversion between binary bits and PAM4 symbols, and the voltage and timing offsets used for simulation waveform processing are specified by the Parameters PAM4_Mapping, PAM4_UpperThreshold, PAM4_CenterThreshold, PAM4_LowerThreshold, PAM4_UpperEyeOffset, PAM4_CenterEyeOffset and PAM4_LowerEyeOffset.

Other Notes: When Usage is declared as In, this Parameter is also passed to the algorithmic model. The EDA tool continues to behave as described above. The use of a single Parameter to control both EDA tool and model behavior is intended to streamline the experience for the end-user.

Examples:

```
(Modulation (Usage Info) (Value "PAM4") (Type String)
         (Description "This is a PAM4 model.")
)

(Modulation (Usage In) (List "NRZ" "PAM4") (Type String)
         (Description "This model can be used either for NRZ or PAM4
         analysis.")
)
```

Parameter: **PAM4_Mapping**

Required: No, and illegal before AMI_Version 6.1

Direction: Rx, Tx

Descriptors:

Usage:	Info, In
Type:	String
Format:	Value, List
Default:	<string_literal>
Description:	<string>

Definition: Tells the EDA tool how to map voltage levels to two-bit PAM4 symbols

Usage Rules: Different devices may translate between voltage levels and two-bit symbols differently, and this parameter defines the mapping to be used for a specific model. There are two different pieces of information to be mapped:

- The four voltage levels in the signal (for example -0.5V, -0.166V, 0.166V, 0.5V in the transmitter's waveform stimulus)
- The four two-bit PAM4 symbols (00, 01, 10, 11)

The PAM4_Mapping parameter declares a four-character string that declares how the EDA tool should map between voltage levels and bit sequences. The *positions* in the string (1st, 2nd, 3rd, 4th) correspond to signal voltage *levels*, beginning with the most negative voltage and becoming incrementally more positive. The *values* of the characters in the string correspond to two-bit binary sequences, with "0" = binary 00, "1" = binary 01, "2" = binary 10, and "3" = binary 11.

Other Notes: A PAM4_Mapping value string of "0132" (the default) tells the EDA tool:

- The most negative voltage should be interpreted as binary 00
- The next higher voltage should be interpreted as binary 01
- The next higher voltage should be interpreted as binary 11
- The most positive voltage should be interpreted as binary 10

A PAM4_Mapping value string of "0123" tells the EDA tool:

- The most negative voltage should be interpreted as binary 00
- The next higher voltage should be interpreted as binary 01

- The next higher voltage should be interpreted as binary 10
- The most positive voltage should be interpreted as binary 11

If the AMI Reserved Parameter Modulation is set to "PAM4" and PAM4_Mapping is *not* declared, the EDA tool should assume a default "Gray code" value of "0132" for PAM4_Mapping. The PAM4_Mapping parameter is ignored when the AMI Reserved Parameter Modulation is not declared or is declared and set to "NRZ". The PAM4_Mapping parameter is illegal when the AMI Reserved Parameter Modulation_Levels is present. The PAM4_Mapping parameter must contain four characters and each of the four characters "0", "1", "2" and "3" must occur once.

There are two reasons why PAM4_Mapping might be used:

1. The EDA tool needs to convert a symbol error rate into a bit error rate. For PAM4, each symbol carries two bits of information. So, when an incorrect symbol is received, there can be either one or two bit errors involved. The EDA tool needs to know how many bits were received in error to accurately calculate a BER.
2. SerDes designers may choose other mappings for reasons of their own. The choice of a mapping may affect the bit error rate, but, for example, might produce error patterns that fall more often into the correctable space of a particular choice of Forward Error Correction (FEC) code. The mapping enables SerDes designers to communicate these choices, and for system developers to evaluate these choices.

Examples:

```
(PAM4_Mapping (Usage Info) (Value "0123") (Type String)
          (Description "Simple mapping from voltages to symbols.")
)

(PAM4_Mapping (Usage Info) (Value "0132") (Type String)
          (Description "Gray code is being used.")
)

(PAM4_Mapping (Usage In) (List "0132" "0123") (Type String)
          (Description "Two PAM4 Mappings are allowed for this model.")
)
```

Parameters: **PAM4_UpperThreshold, PAM4_CenterThreshold, PAM4_LowerThreshold**

Required: No, and illegal before AMI_Version 6.1

Direction: Rx

Descriptors:

Usage:	Info, InOut, Out, Dep
Type:	Float
Format:	Value
Defaults:	<numeric_literal> …
Description:	<string>

Definition: Voltages used by EDA tools for PAM4 waveform and eye processing.

Usage Rules: The EDA tool uses these voltages in conjunction with Rx clock information to detect which of the four PAM4 symbols a waveform represents when the zero voltage-centered signal is sampled:

- Voltages *lower* than **PAM4_LowerThreshold - Rx_Receiver_Sensitivity** are detected as voltage level 0
- Voltages *lower* than **PAM4_CenterThreshold – Rx_Receiver_Sensitivity** and *greater* than **PAM4_LowerThreshold + Rx_Receiver_Sensitivity** are detected as voltage level 1
- Voltages *lower* than **PAM4_UpperThreshold – Rx_Receiver_Sensitivity** and *greater* than **PAM4_CenterThreshold + Rx_Receiver_Sensitivity** are detected as voltage level 2
- Voltages *greater* than **PAM4_UpperThreshold + Rx_Receiver_Sensitivity** are detected as voltage level 3

Voltages that do *not* fall into one of these regions are considered a symbol error.

If these parameters are declared as Usage InOut or Out, the algorithmic model is expected to output values from the AMI_Init and AMI_GetWave call for the EDA tool to use during waveform and eye processing.

If the AMI Reserved Parameter Modulation lists “PAM4” (either as a Value or as a List selection), PAM4_UpperThreshold and PAM4_LowerThreshold are required for Rx AMI parameter definition files. If PAM4_CenterThreshold is *not* declared, the value of PAM4_CenterThreshold shall default to 0.0 volts.

The PAM4_UpperThreshold, PAM4_CenterThreshold and PAM4_LowerThreshold parameters are ignored when the AMI Reserved Parameter Modulation is not declared or is declared and set to “NRZ”, or when AMI Reserved Parameter Modulation_Levels is declared.

Other Notes:

Examples:

```
(PAM4_LowerThreshold (Usage Info) (Value -0.333) (Type Float)
         (Description "Lower eye voltage threshold for waveform and eye
         processing.")
)
(PAM4_CenterThreshold (Usage Info) (Value 0.0) (Type Float)
         (Description "Center eye voltage threshold for waveform and eye
         processing.")
)
(PAM4_UpperThreshold (Usage Info) (Value 0.333) (Type Float)
         (Description "Upper eye voltage threshold for waveform and eye
         processing.")
)

(PAM4_LowerThreshold (Usage Out) (Value -0.333) (Type Float)
         (Description "Lower eye voltage threshold returned by AMI_Init.")
)
(PAM4_CenterThreshold (Usage Out) (Value 0.0) (Type Float)
         (Description "Center eye voltage threshold returned by AMI_Init.")
)
```

```
(PAM4_UpperThreshold (Usage Out) (Value 0.333) (Type Float)
         (Description "Upper eye voltage threshold returned by AMI_Init.")
)
```

Parameters: **PAM4_UpperEyeOffset, PAM4_CenterEyeOffset, PAM4_LowerEyeOffset**

Required: No, and illegal before AMI_Version 6.1

Direction: Rx

Descriptors:

Usage:	Info, InOut, Out, Dep
Type:	Float, UI
Format:	Value
Default:	<numeric_literal>
Description:	<string>

Definition: Sampling clock offsets for Upper, Center and Lower PAM4 eyes

Usage Rules: Rx models provide a single set of sampling information returned that pertains to a nominal eye centered between consecutive edge threshold crossing times during PAM4 analysis. When the PAM4 Upper, Center and Lower eyes have a time shift with respect to the nominal eye, these parameters are used to define a sampling offset from the nominal eye.

When a positive value is declared, the latch in question will sample the waveform *after* the sample time for the nominal eye. When a negative value is declared, the latch in question will sample the waveform *before* the sample time for the nominal eye.

If these parameters are declared as Usage InOut or Out, the algorithmic model is expected to output values from the AMI_Init and AMI_GetWave call for the EDA tool to use during waveform and eye processing.

If the AMI Reserved Parameter Modulation is set to "PAM4" and these offset values are *not* declared, the EDA tool is expected to use a default value of 0.0 for each offset parameter not declared. The PAM4_UpperEyeOffset, PAM4_CenterEyeOffset and PAM4_LowerEyeOffset parameters are ignored when the AMI Reserved Parameter Modulation is not declared or is declared and set to "NRZ", or when AMI Reserved Parameter Modulation_Levels is declared.

Other Notes: In statistical analysis, offset from the center of the nominal eye shall include Rx_Clock_Recovery_Mean and either the PAM4_UpperEyeOffset, PAM4_CenterEyeOffset and PAM4_LowerEyeOffset. In time-domain analysis, PAM4_UpperEyeOffset, PAM4_CenterEyeOffset and PAM4_LowerEyeOffset shall be three independent corrections to the clock times. Specifically, the PAM4_UpperEyeOffset and PAM4_LowerEyeOffset are offsets from the nominal eye and not the PAM4_CenterEyeOffset.

Examples:

```
(PAM4_UpperEyeOffset (Usage Out) (Value 2.5e-12) (Type Float)
         (Description "The upper eye sampling offset.")
)

(PAM4_CenterEyeOffset (Usage Out) (Value 0.0) (Type Float)
         (Description "The center eye sampling offset.")
)
```

```
(PAM4_LowerEyeOffset (Usage Out) (Value 2.5e-12) (Type Float)
         (Description "The lower eye sampling offset.")
)
```

Parameter: **Modulation_Levels**

Required: No, and illegal before AMI_Version 7.2

Direction: Rx, Tx

Descriptors:

Usage:	In
Type:	Integer
Format:	Value or List
Default:	<numeric_literal>
Description:	<string>

Definition: Tells the EDA tool (and optionally, the algorithmic model) whether NRZ or PAMn modulation is to be used for analysis.

Usage Rules: Modulation and Modulation_Levels shall not both be present. If the format is Value, then the value shall be greater than 1. If the format is List then all values shall be greater than 1. If neither Modulation nor Modulation_Levels are defined, then the modulation scheme used by the EDA tool must be NRZ. The following table maps typical Modulation_Levels to common modulation names.

Modulation_Levels	Common Name
2	NRZ, PAM2
3	PAM3, Duobinary
4	PAM4
5	PAM5
…	
8	PAM8
…	

The Modulation_Levels parameter controls how the EDA tool prepares the stimulus waveform for AMI_GetWave-based analysis and post-processes simulation results:

- When Modulation_Levels is set to 2, the simulator prepares the input stimulus using -0.5V to represent a logic 0 and 0.5V to represent a logic 1. The Rx parameter Rx_Receiver_Sensitivity is used to post-process Rx model data.
- When Modulation_Levels is set to "n", the simulator prepares the input stimulus using voltage levels between -0.5 and 0.5 volts in uniform increments of 1.0/(n-1) volts. There are n voltage levels corresponding to n symbol levels between 0 and n-1. The voltage and timing offsets used for simulation waveform processing are specified by the parameters PAM_Thresholds and PAM_Offsets.

Example:

```
(Modulation_Levels (Usage In) (List 2 3) (Type Integer)
```

```
        (Description "This model can be used either for NRZ or PAM3
        analysis.")
)
```

Parameter: **PAM_Thresholds**

Required: Yes, if Modulation_Levels is specified, illegal if Modulation_Levels is not specified

Direction: Rx

Descriptors:

Usage:	Out
Type:	String
Format:	Value
Defaults:	<string_literal>
Description:	<string>

Definition: Voltages used by EDA tools for PAMn waveform and eye processing. The string returned must contain n-1 float values of the threshold (volts) separated by white spaces.

Usage Rules: The EDA tool uses the voltages returned by the executable model through this parameter in conjunction with Rx clock information to detect which of the n PAMn symbols a waveform represents when the signal is sampled.

A PAMn eye has n-1 "eyes" and n symbol levels. The first float value in the string (**Value 1**) returned for PAM_Thresholds is the voltage threshold of eye number 1, the second value (**Value 2**) is the voltage threshold of eye number 2 and so on. The threshold for each eye is typically at the "vertical center" of that eye.

- Voltages *lower* than **Value 1 PAM_Thresholds – Rx_Receiver_Sensitivity** are detected as symbol level **0**
- Voltages *lower* than **Value 2 PAM_Thresholds – Rx_Receiver_Sensitivity** and *greater* than **Value 1 PAM_Thresholds + Rx_Receiver_Sensitivity** are detected as symbol level **1**
- **…**
- Voltages *lower* than **Value n-1 PAM_Thresholds – Rx_Receiver_Sensitivity** and *greater* than **Value n-2 PAM_Thresholds + Rx_Receiver_Sensitivity** are detected as symbol level **n-2**
- Voltages *greater* than **Value n-1 PAM_Thresholds + Rx_Receiver_Sensitivity** are detected as symbol level **n-1**

Example:

```
(PAM_Thresholds (Usage Out) (Type String)
      (Description "Output values from executable model for numerical
      thresholds.")
      (Value "placeholder")
)
```

Parameters: **PAM_Offsets**

Required: No, and illegal if Modulation_Levels is not specified

Direction: Rx

Descriptors:

Usage:	Out
Type:	String
Format:	Value
Defaults:	<string_literal>
Description:	<string>

Definition: Sampling clock offsets for PAMn eyes. The string returned must contain n-1 float values of the clock offsets separated by white spaces.

Usage Rules: A PAMn receiver has n-1 latches. PAM_Offsets is used to allow different sampling times at each latch (eye). There are existing ways to determine the nominal_sample_time that the latches are sampled. The values of PAM_Offsets are added to the nominal_sample_time. The sampling time of the k^{th} eye = nominal_sample_time + k^{th} value of PAM_Offsets, where nominal_sample_time is defined as follows.

Case 1: Statistical simulation, Rx_Decision_Time is present
nominal_sample_time = Rx_Decision_Time

Case 2: Statistical simulation, Rx_Clock_Recovery_Mean is present, Rx_Decision_Time is not present
nominal_sample_time = ideal_time + Rx_Clock_Recovery_Mean

Case 3: Statistical simulation, Rx_Clock_Recovery_Mean and Rx_Decision_Time are not present
nominal_sample_time = ideal_time

Case 4: Time domain simulation, Rx AMI_GetWave outputs clock_times
nominal_sample_time = clock_times + ½ UI*

Case 5: Time domain simulation, Rx AMI_GetWave does not output clock_times
nominal_sample_time = ideal_time + Rx_Clock_Recovery_Mean

where ideal_time is halfway between the median of the threshold crossing times on both sides of the eye. If the AMI Reserved Parameter Modulation_Levels is defined and these offset values are *not* declared, the EDA tool is expected to use a default value of 0.0 for each offset parameter.

Other Notes:

Example:

```
(PAM_Offsets (Usage Out) (Type String)
      (Description "Output values from executable model for clock timing
      offsets.")
      (Value "placeholder")
)
```

10.7.1 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Tables summarizing the rules for the Modulation Reserved Parameters are shown below.

Table 28 – General Rules and Allowable Usage for Modulation Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage				
	Required	Default[2]	Info	In	Out	Dep[1]	InOut
Modulation	No	"NRZ"	X	X			
Modulation_Levels	No	(see Usage Rules)		X			
PAM_Thresholds	No[3]	(see Usage Rules)			X		
PAM_Offsets	No	(see Usage Rules)			X		
PAM4_Mapping	No	"0132"	X	X			
PAM4_UpperThreshold	No	(see Usage Rules)	X		X	X	X
PAM4_CenterThreshold	No	0	X		X	X	X
PAM4_LowerThreshold	No	(see Usage Rules)	X		X	X	X
PAM4_UpperEyeOffset	No	0	X		X	X	X
PAM4_CenterEyeOffset	No	0	X		X	X	X
PAM4_LowerEyeOffset	No	0	X		X	X	X

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) Required if Modulation_Levels is present.

Table 29 – Allowable Data Types for Modulation Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Modulation				X	
Modulation_Levels			X		

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
PAM_Offsets				X	
PAM_Thresholds				X	
PAM4_Mapping				X	
PAM4_UpperThreshold	X				
PAM4_CenterThreshold	X				
PAM4_LowerThreshold	X				
PAM4_UpperEyeOffset	X	X			
PAM4_CenterEyeOffset	X	X			
PAM4_LowerEyeOffset	X	X			

Table 30 – Allowable Data Formats for Modulation Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
Modulation	X			X						
Modulation_Levels	X			X						
PAM_Offsets	X									
PAM_Thresholds	X									
PAM4_Mapping	X			X						
PAM4_UpperThreshold	X									
PAM4_CenterThreshold	X									
PAM4_LowerThreshold	X									
PAM4_UpperEyeOffset	X									
PAM4_CenterEyeOffset	X									
PAM4_LowerEyeOffset	X									

10.8 REPEATERS

A Repeater is a type of device that is placed in the middle of the channel to compensate for channel loss. Repeaters consist of two categories, Redrivers and Retimers. A Redriver equalizes the upstream channel signal and retransmits it to the downstream channel. The output signal is continuously driven by the input signal. A Redriver does not have a clock-data recovery circuit (CDR), and no retiming is performed when the Redriver retransmits the signal. A Retimer equalizes the upstream channel signal, recovers the clock using a CDR and generates a digital stimulus that is transmitted to the downstream channel.

A Repeater is modeled by two back-to-back input-output IBIS-AMI models as shown in Figure 43.

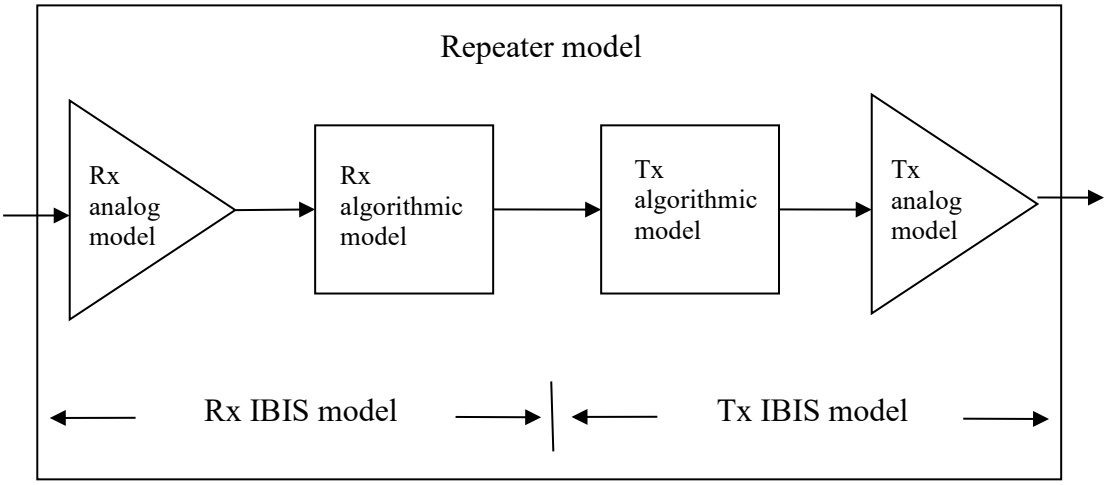


Figure 43 – Repeater Model

The analog part of the Rx model represents the input termination at the device input. The analog part of the Tx model represents the output impedance at the device output. The two algorithmic models represent equalizers, clock data recovery or CDR circuits (if they exist) and/or pre-emphasis inside the devices. In a Redriver, both algorithmic models can optionally implement the AMI_GetWave function. In a Retimer, the Rx algorithmic model must implement AMI_GetWave and the function must return clock times. The Retimer Tx algorithmic model can optionally implement AMI_GetWave. The order of signal flow in a Repeater model is from Rx analog to Rx algorithmic to Tx algorithmic to Tx analog. Looking from the Rx analog portion, the Rx algorithmic block is assumed to have infinite input impedance. Looking from the Tx analog portion, the Tx algorithmic block is assumed to have an output of an ideal voltage source.

A Repeater model is specified in a single .ibs file that includes both input and output models.

While the text and figures herein describe links with only one Repeater, multiple Repeaters, or a mixture of Redrivers and Retimers, cascaded in a link, are permitted.

Keyword: **[Repeater Pin]**

Required: No

Description: Associates a differential Rx non-inv pin with a Tx non-inv pin to form a Repeater.

Sub-Params: tx_non_inv_pin

Usage Rules: Enter only Repeater pin pairs. The first column, [Repeater Pin] contains a non-inv pin name of an entry in the [Diff Pin] section that represents an Input or Input_diff model corresponding to the Rx part of the Repeater model. The second column, tx_non_inv_pin contains a non-inv pin name of an entry in the [Diff Pin] section that represents an Output or Output_diff model corresponding to the Tx part of the Repeater model.

If [Repeater Pin] is present, the [Model]s associated with the pins listed under [Repeater Pin] shall contain [Algorithmic Model] sections. The AMI parameter definition files for the [Algorithmic Model]s associated with the receiver (Model_type Input or Input_diff) shall contain the Repeater_Type parameter.

Other Notes: Each line must contain two columns. A pin name may appear in only one [Repeater Pin] record.

The column length limits are:

[Repeater Pin]	20 characters maximum
tx_non_inv_pin	20 characters maximum

Example:

```
[Repeater Pin]  tx_non_inv_pin
3               11
```

AMI Reserved Parameters:

Parameter: **Repeater_Type**

Required: No, and illegal before AMI_Version 6.0

Direction: Rx

Descriptors:

Usage:	Info
Type:	String
Format:	Value
Default:	<string_literal>
Description:	<string>

Definition: This Reserved Parameter identifies the type of Repeater associated with a Repeater Rx model. Allowed values are “Redriver” and “Retimer”.

Usage Rules: This parameter is required if the Rx model is part of a Repeater Rx/Tx pair. A Retimer Rx model shall contain AMI_GetWave (GetWave_Exists is True) and the AMI_GetWave function shall return clock times. The [Model] associated with the AMI parameter definition file containing Repeater_Type shall be of Model_type Input or Input_diff. Further, the [Model] shall be associated with a [Pin] listed under the [Repeater Pin] keyword, or with a differential pair that has its non-inverting [Pin] listed under the [Repeater Pin] keyword.

Other Notes:

Example:

```
(Repeater_Type (Usage Info) (Type String) (Value "Redriver"))
```

10.8.1 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Tables summarizing the Reserved Parameters for Repeaters are shown below.

Table 31 – General Rules and Allowable Usage for Repeater Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage				
	Required	Default[2,4]	Info	In	Out	Dep[1]	InOut
Repeater_Type	No[3]	--	X				

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) Required if [Repeater Pin] is present.
4) "--" means that an entry must be provided if the parameter is present; no default is assumed or permitted.

Table 32 – Allowable Data Types for Repeater Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Repeater_Type				X	

Table 33 – Allowable Data Formats for Repeater Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
Repeater_Type	X									

As mentioned above, a Retimer Rx shall contain AMI_GetWave (GetWave_Exists is True) and the AMI_GetWave function must return clock times. The EDA tool shall generate a digital input to the Retimer Tx by sampling the Rx AMI_GetWave output waveform appropriate to its Modulation. If Modulation is NRZ, the digital stimulus shall have values of -½ and +½. For other Modulation values, see MODULATION RESERVED PARAMETERS and the clock_times definition for the AMI_GetWave function.

In Repeater AMI simulations, both Repeater analog models are treated as if they are linear and time-invariant. The incoming (upstream) analog channel of the Redriver, including the upstream Tx analog model, the physical channel and the Repeater Rx analog model, is represented by an

impulse response. The outgoing (downstream) analog channel of the Repeater, including the Repeater Tx analog model, the physical channel and the downstream Rx analog model, is represented by another impulse response.

10.8.2 REPEATER TIME-DOMAIN SIMULATION FLOW FOR A REPEATER LINK

The physical layout of the Repeater (Retimer or Redriver) link discussed below is illustrated in Figure 44.

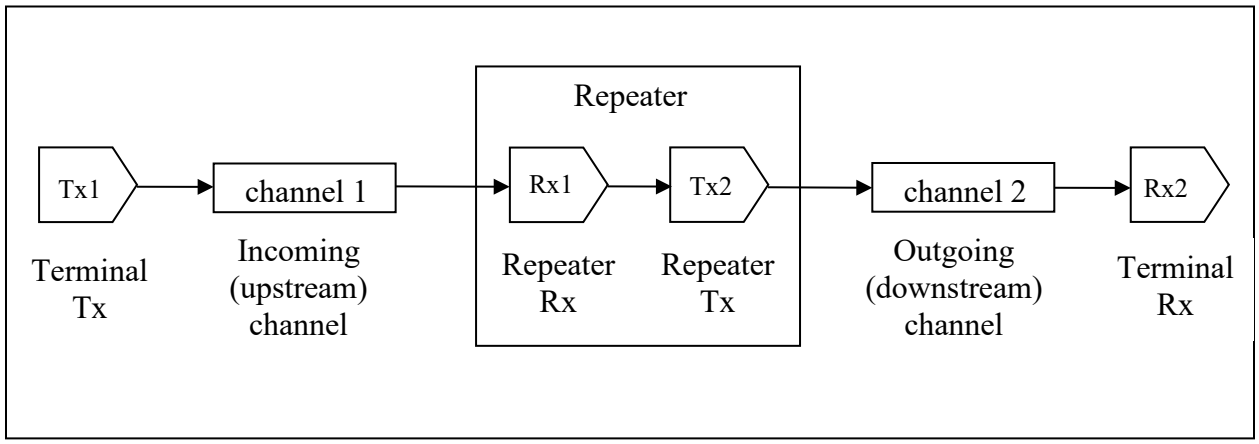


Figure 44 – Repeater Link

Here Tx1 denotes the Repeater upstream channel (channel 1) Tx AMI model (including analog and algorithmic models), Rx1 denotes the Repeater Rx AMI model (including analog and algorithmic models), Tx2 denotes the Repeater Tx AMI model (including analog and algorithmic models), and Rx2 denotes the Repeater Downstream channel (channel 2) Rx AMI model (including analog and algorithmic models).

Retimer Statistical Simulation Flow

Step 1. The EDA tool obtains the impulse response of the upstream analog channel, which represents the combined impulse response of Tx1's analog model, physical channel 1, and Rx1's analog model.

Step 2. The output of step 1 is presented to the Tx1's AMI_Init function, and Tx1's AMI_Init function is executed.

Step 3. The output of step 2 is presented to the Rx1's AMI_Init function, and the Rx1's AMI_Init function is executed.

Step 4. The EDA tool obtains the impulse response of the downstream analog channel, which represents the combined impulse response of Tx2's analog model, physical channel 2, and Rx2's analog model.

Step 5. The output of step 4 is presented to Tx2's AMI_Init function, and Tx2's AMI_Init function is executed.

Step 6. The output of step 5 is presented to Rx2's AMI_Init function, and Rx2's AMI_Init function is executed.

Step 7. The EDA tool completes the rest of the statistical simulation and analysis using the impulse response returned in step 3 by the Rx1's AMI_Init function, which is a complete representation of the behavior of Tx1 and Rx1 algorithmic models combined with the upstream channel 1, and the impulse response returned in step 6 by the Rx2's AMI_Init function, which is a complete representation of the behavior of Tx2 and Rx2 algorithmic models combined with the downstream channel 2.

Retimer Time Domain Simulation Flow

Step 1. The EDA tool obtains the impulse response of the upstream analog channel, which represents the combined impulse response of Tx1's analog model, physical channel 1, and Rx1's analog model.

Step 2. The output of step 1 is presented to Tx1's AMI_Init function, and Tx1's AMI_Init function is executed.

Step 3. The output of step 2 is presented to Rx1's AMI_Init function, and Rx1's AMI_Init function is executed.

Step 4. The EDA tool obtains the impulse response of the downstream analog channel, which represents the combined impulse response of Tx2's analog model, physical channel 2, and Rx2's analog model.

Step 5. The output of step 4 is presented to Tx2's AMI_Init function, and Tx2's AMI_Init function is executed.

Step 6. The output of step 5 is presented to Rx2's AMI_Init function, and Rx2's AMI_Init function is executed.

Step 7. The EDA tool performs the time domain simulation on the upstream channel, which consists of Tx1, physical channel 1, and Rx1, according to the AMI flow defined in the specification for channels without Repeaters.

Step 8. The EDA tool samples the output waveform of Retimer Rx1 AMI_GetWave at ½ UI after each clock time returned by the function, generates a digital stimulus as the input to Tx2's algorithmic model, regardless of whether Tx2's AMI_GetWave exists or not, and performs the simulation on the downstream channel, which consists of Tx2, physical channel 2, and Rx2, according to the AMI flow defined in the specification for channels without Repeater. For NRZ signaling where Modulation or Modulation_Levels is not used, the logic level of the digital stimulus is 1 if sampled value >= Rx1's Rx_Receiver_Sensitivity and 0 if sampled value <= –Rx1's Rx_Receiver_Sensitivity. If –Rx1's Rx_Receiver_Sensitivity < sampled value < Rx1's Rx_Receiver_Sensitivity, the logic level is unchanged from the previous bit. The digital stimulus shall have values of -½ volt for logic 0 and +½ volt for logic 1. When the PAM4 parameters introduced in IBIS 6.1 are used, the digital state of the receiver is determined by the parameters PAM4_UpperThreshold, PAM4_CenterThreshold, PAM4_LowerThreshold, PAM4_UpperEyeOffset, PAM4_CenterEyeOffset, and PAM4_LowerEyeOffset, and the stimulus is determined by the Modulation parameter. When the PAMn parameters introduced in IBIS 7.2

are used, the digital state of the receiver is determined by the parameters PAM_Thresholds and PAM_Offsets. The simulator prepares the input stimulus for the attached Tx according to rules described for the Modulation_Levels parameter.

Steps 7 through 8 can be called once or can be called multiple times to process the full analog waveform. Processing the full analog waveform using multiple calls, where each call passes the model a block of samples, reduces the memory requirements when performing long simulations. This also allows AMI_GetWave to return model status after processing each block (note that the size of a block is not limited to the number of samples needed to define an integer number of complete symbols). Once all blocks of the input waveform have been processed, the EDA tool calls the AMI_Close function of each algorithmic model in Tx1, Rx1, Tx2, and Rx2.

Since the Retimer output signal is driven by a digital stimulus as described above in step 8, jitter and noise parameters specified in Retimer .ami files are applied according to the specification for channels without Repeaters.

Redriver Flows

Both statistical and time domain Redriver simulations require that AMI_Init functions of Tx1, Rx1, Tx2, and Rx2 are executed first. The following figure shows flows of executing AMI_Init functions in Redriver statistical and time domain simulations when the Tx2 Tx_Impulse_Input is set to “Downstream”, “Combined”, “Separate”, and “Upstream”, respectively. By setting Tx_Impulse_Input to “Upstream”, the Tx2 model maker is declaring that the Tx2 initialization (AMI_Init) function does not have the ability to adapt itself based on the downstream channel.

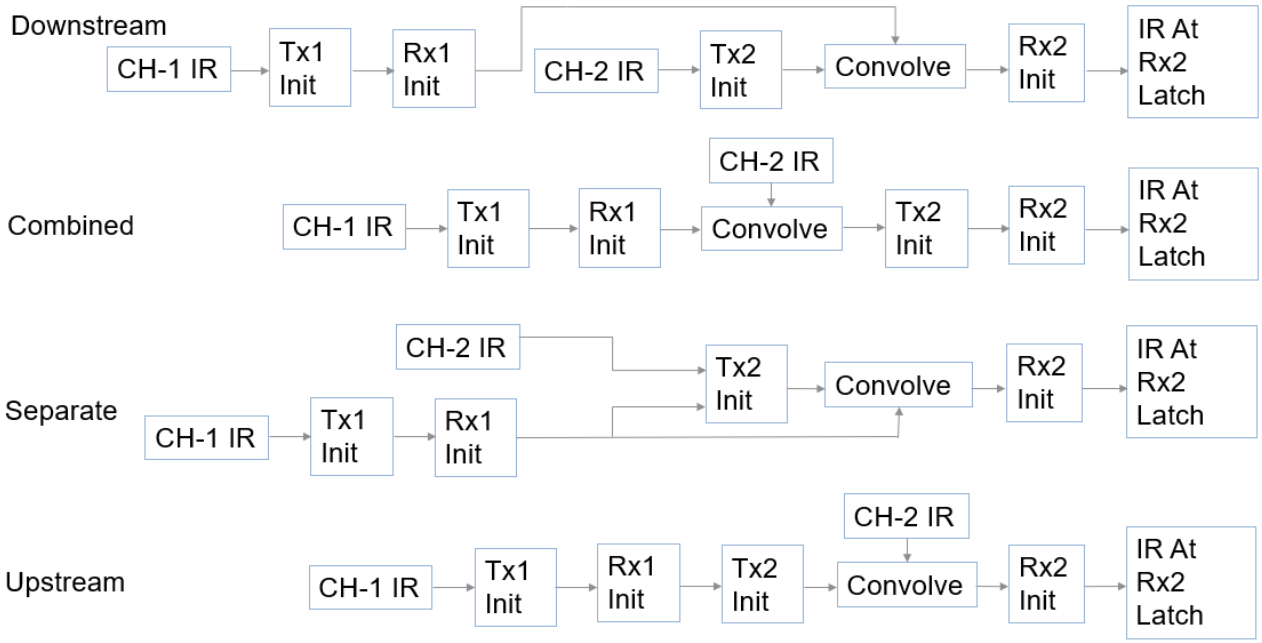


Figure 45 – Redriver AMI_Init Execution for Various Tx_Impulse_Input Settings

If Tx_Impulse_Input is not present, the AMI_Init functions shall be executed in the same manner as when Tx_Impulse_Input is set to Downstream. This rule shall be applied to all AMI models.

After completing all steps of executing Tx1, Rx1, Tx2 and Rx2 AMI_Init functions, the EDA tool may use results generated in these steps to perform the rest of the statistical or time domain simulation as described below.

Redriver Statistical Simulation Flow

To perform statistical simulations, all models, including the terminal Tx, Redriver Rx, Redriver Tx, and terminal Rx shall set Init_Returns_Impulse to True.

Step 1. The EDA tool obtains the impulse response of the analog channel 1, which represents the combined impulse response of Tx1's analog model, physical channel 1, and Rx1's analog model.

Step 2. The output of step 1 is presented to Tx1's AMI_Init function, and Tx1's AMI_Init function is executed.

Step 3. The output of step 2 is presented to Rx1's AMI_Init function, and Rx1's AMI_Init function is executed.

Step 4. The EDA tool obtains the impulse response of the analog channel 2, which represents the combined impulse response of Tx2's analog model, physical channel 2, and Rx2's analog model.

Step 5a. If Tx2's Tx_Impulse_Input is not present or is "Downstream" then column 1 of impulse_matrix shall contain the output of step 4. This impulse_matrix is presented to Tx2's AMI_Init function, and Tx2's AMI_Init function is executed.

Step 5b. If Tx2's Tx_Impulse_Input is "Combined" then column 1 of impulse_matrix shall contain the output of step 3 convolved with the output of step 4. This impulse_matrix is presented to Tx2's AMI_Init function, and Tx2's AMI_Init function is executed.

Step 5c. If Tx2's Tx_Impulse_Input is "Separate" then column 1 of impulse_matrix shall contain the output of step 4 and column "aggressors+2" shall contain the output of step 3. This impulse_matrix is presented to Tx2's AMI_Init function, and Tx2's AMI_Init function is executed.

Step 5d. If Tx2's Tx_Impulse_Input is "Upstream" then column 1 of impulse_matrix shall contain the output of step 3. This impulse_matrix is presented to Tx2's AMI_Init function, and Tx2's AMI_Init function is executed.

Step 6a. If Tx2's Tx_Impulse_Input is not present or is "Downstream" then the output of column 1 of step 5 is convolved with the output of step 3, the result is presented to Rx2's AMI_Init function, and Rx2's AMI_Init function is executed.

Step 6b. If Tx2's Tx_Impulse_Input is "Combined" then the output of column 1 of step 5 is presented to Rx2's AMI_Init function, and Rx2's AMI_Init function is executed.

Step 6c. If Tx2 Tx_Impulse_Input is “Separate” then the output of column 1 of step 5 is convolved with the output of step 3, the result is presented to Rx2’s AMI_Init function, and Rx2’s AMI_Init function is executed.

Step 6d. If Tx2 Tx_Impulse_Input is “Upstream” then the output of column 1 of step 5 is convolved with the output of step 4, the result is presented to Rx2’s AMI_Init function, and Rx2 AMI_Init function is executed.

Step 7. The EDA tool completes the rest of the simulation and analysis using the impulse response returned in step 6 by the Rx2’s AMI_Init function.

Redriver Time Domain Simulation Flow

Step 1. The EDA tool obtains the impulse response of the analog channel 1, which represents the combined impulse response of Tx1’s analog model, physical channel 1, and Rx1’s analog model.

Step 2. The output of step 1 is presented to Tx1’s AMI_Init function, and Tx1’s AMI_Init function is executed.

Step 3. The output of step 2 is presented to Rx1’s AMI_Init function, and Rx1’s AMI_Init function is executed.

Step 4. The EDA tool obtains the impulse response of the analog channel 2, which represents the combined impulse response of Tx2’s analog model, physical channel 2, and Rx2’s analog model.

Step 5a. If Tx2’s Tx_Impulse_Input is not present or is “Downstream” then column 1 of impulse_matrix shall contain the output of step 4. This impulse_matrix is presented to Tx2’s AMI_Init function, and Tx2’s AMI_Init function is executed.

Step 5b. If Tx2’s Tx_Impulse_Input is “Combined” then column 1 of impulse_matrix shall contain the output of step 3 convolved with the output of step 4. This impulse_matrix is presented to Tx2’s AMI_Init function, and Tx2’s AMI_Init function is executed.

Step 5c. If Tx2’s Tx_Impulse_Input is “Separate” then column 1 of impulse_matrix shall contain the output of step 4 and column “aggressors+2” shall contain the output of step 3. This impulse_matrix is presented to Tx2’s AMI_Init function, and Tx2’s AMI_Init function is executed.

Step 5d. If Tx2’s Tx_Impulse_Input is “Upstream” then column 1 of impulse_matrix shall contain the output of step 3. This impulse_matrix is presented to Tx2’s AMI_Init function, and Tx2’s AMI_Init function is executed.

Step 6a. If Tx2’s Tx_Impulse_Input is not present or is “Downstream” then the output of column 1 of step 5 is convolved with the output of step 3, the result is presented to Rx2’s AMI_Init function, and Rx2’s AMI_Init function is executed.

Step 6b. If Tx2’s Tx_Impulse_Input is “Combined” then the output of column 1 of step 5 is presented to Rx2’s AMI_Init function, and Rx2’s AMI_Init function is executed.

Step 6c. If Tx2 Tx_Impulse_Input is “Separate” then the output of column 1 of step 5 is convolved with the output of step 3, the result is presented to Rx2’s AMI_Init function, and Rx2’s AMI_Init function is executed.

Step 6d. If Tx2 Tx_Impulse_Input is “Upstream” then the output of column 1 of step 5 is convolved with the output of step 4, the result is presented to Rx2’s AMI_Init function, and Rx2’s AMI_Init function is executed.

Step 7. The EDA tool performs the simulation on the upstream channel, which consists of Tx1, physical channel 1, and Rx1, according to the AMI flow defined in the specification for channels without Repeaters.

Step 8. The EDA tool uses the signal waveform at the output of Rx1’s algorithmic model in step 7 as the stimulus of Tx2’s algorithmic model and performs the simulation on the downstream channel, which consists of Tx2, physical channel 2, and Rx2, according to the AMI flow defined in the specification for channels without Repeaters.

Steps 7 through 8 can be called once or can be called multiple times to process the full analog waveform. Processing the full analog waveform using multiple calls, where each call passes the model a block of samples, reduces the memory requirements when performing long simulations. This also allows AMI_GetWave to return model status after processing each block (note that the size of a block is not limited to the number of samples needed to define an integer number of complete symbols). Once all blocks of the input waveform have been processed, the EDA tool calls the AMI_Close function of each algorithmic model in Tx1, Rx1, Tx2 and Rx2.

Since the Redriver output signal is driven continuously by the input analog signal and does not have a sampling latch, clock times, if returned by the Rx1 AMI_GetWave function, jitter parameters, and the Rx_Noise parameter specified in Redriver .ami files are ignored by the EDA tool.

10.9 AMI RESERVED PARAMETER DEFINITIONS FOR LINK TRAINING COMMUNICATIONS

With the information provided in this section, IC vendors and EDA tool vendors should be able to develop models that support Back Channel Training and develop enhancements EDA tools will need to support these models. The following Reserved Parameters are in the AMI file and positioned under the Reserved_Parameters branch.

Parameter: **BCI_Protocol**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx, Tx

Descriptors:

Usage:	In
Type:	String
Format:	Value, List
Default:	<string literal>
Description:	<string>

Definition: This parameter contains the name(s) of Back-Channel Interface Protocol(s) that the model supports. This parameter tells the model which Back-Channel Interface Protocol is being used for the training process. The BCI_Protocol defines the back-channel message files and BCI data contained therein that is read and/or generated by each call to each executable model.

Usage Rules: Both the transmitter and receiver for a given channel must have identical settings for the BCI_Protocol parameter for link training to be enabled.

BCI_Protocol must be present if the model supports any BCI protocol. If BCI_Protocol is not present, the use of any other BCI parameter is illegal.

Other Notes: A BCI_Protocol may be private or approved by the IBIS Open Forum. Protocol names beginning with the prefix "IBIS" are reserved for protocols approved by the IBIS Open Forum. Names for private and independently-specified published protocols should contain character strings sufficiently unique to avoid conflicts with other independently-named protocols.

Example:

```
(BCI_Protocol (Usage In)(Type String)(Value "Company_xyz")
      (Description "This Device supports Back-Channel Interface Protocol
      Company_xyz.  For private protocols, we suggest that the name should
      begin with a company name to help keep private protocol names unique.
      Protocols officially adopted by the IBIS Open Forum would begin with
      IBIS."))
```

Parameter: **BCI_ID**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx, Tx

Descriptors:

Usage: In
Type: String
Format: Value
Default: <string literal>
Description: <string>

Definition: The EDA tool is responsible for recognizing this parameter name and replacing the value in the .ami file with a partial file name that itself must conform to the rules for a "file name" in Section 3.1, "File Naming Definitions", but not including a "file name extension" as defined therein. The algorithmic model is responsible for using BCI_ID as the base name string for any data files that the model creates, either for use as temporary storage or for recording output data in accordance with the BCI_Protocol. File names created by the algorithmic model from BCI_ID shall also conform to the rules in Section 3.

The use of BCI_ID can be used to help guarantee that data from multiple channels are not overwritten as a result of collisions between temporary or permanent file names. It is the EDA tool's responsibility to ensure that BCI_ID represents a valid "namespace", that is any conforming file name that can be created by the algorithmic model from BCI_ID will not unintentionally match a file name already reserved for other use. All model instances in a channel, including buffers in channels with Repeaters, shall share a unique BCI_ID set which directs them to the same namespace in the same directory. Each channel analyzed concurrently (as in a crosstalk simulation) has its own BCI_ID set.

Usage Rules: To access a file within the namespace using BCI_ID, the executable model should create a file name by creating a string consisting of the value of BCI_ID appended with additional characters as specified in BCI_Protocol to create the complete name of the file. If the EDA tool uses BCI_ID to specify a namespace in a directory other than the current working directory, the directory must exist and be read/write accessible to the executable models. If the executable models in a channel do not share the same working directory, this may require the EDA tool to provide different paths in each model's BCI_ID to direct them to the same namespace.

BCI_ID must be present if BCI_Protocol is present. BCI_ID must be absent if BCI_Protocol is absent.

Other Notes: A BCI_Protocol may define one, two (e.g., one per direction) or any number of BCI message files with the same BCI_ID prefix to be used by the Tx and Rx executable models of the analyzed channel to support the required back-channel optimization.

Example:

```
(BCI_ID (Usage In) (Type String) (Value "dll_scratch_dir/channel1")
      (Description "Models may create/read/write/delete files in
      'dll_scratch_dir' with names beginning with 'channel1'"))
```

Parameter: **BCI_State**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx, Tx

Descriptors:

Usage: InOut

Type:	String
Format:	List (“Off” “Training” “Converged” “Failed” “Error”)
Default:	<string_literal>
Description:	<string>

Definition: The user, through the EDA tool, sets the initial value of BCI_State to either “Off” or “Training” on the calls to the Tx and Rx AMI_Init functions. The values of BCI_State sent to the Tx and Rx executable models shall be the same for both the Tx and Rx AMI_Init functions.

Usage Rules: If the BCI_State is “Off” on the calls to Tx and Rx AMI_Init, neither the Tx nor the Rx executable models will read or generate files in the BCI_ID namespace. The values of BCI_Protocol, BCI_Message_Interval_UI or BCI_Training_UI shall be ignored by the executable models. Executable models receiving BCI_State “Off” and subsequently returning BCI_State shall return BCI_State “Off”.

If BCI_State is present, the values of BCI_Protocol and BCI_ID are required; additionally, the values of BCI_Message_Interval_UI and BCI_Training_UI are required for Rx models. If the BCI_State is “Training” on the calls to Tx and Rx AMI_Init, both the Tx and Rx executable models will read and/or write files in the BCI_ID namespace per the BCI_Protocol. The Rx AMI_GetWave calls shall return a value in BCI_State of either “Training”, “Converged”, “Failed” or “Error”. If the Tx AMI_GetWave returns a value in BCI_State, it shall also be either “Training”, “Converged”, “Failed” or “Error”; “Training”, “Converged”, and “Failed” should reflect the Rx state per the BCI_Protocol.

The EDA tool shall consider the value of BCI_State returned by the terminating Rx executable model to be the definitive BCI_Protocol training state. However, any executable model in the channel, upon returning a BCI_State value of “Error”, may thereby signal that a BCI_Protocol has failed due to a mis-communication under the BCI_Protocol.

If the returned value is “Training”, then the Tx and Rx AMI_GetWave will continue to read and/or modify BCI_ID files per the BCI_Protocol.

If the returned value is “Converged”, then the Tx and Rx AMI_GetWave may continue to read and/or modify the BCI_ID files per the BCI_Protocol. However, it is implied that no further adaptation is performed under the BCI_Protocol and the EDA tool may complete the simulation starting with the settings determined through training.

If the returned value is “Failed” the Rx AMI_GetWave function indicates a condition that it was not able to converge in its search algorithm. Then the Tx and Rx AMI_GetWave may continue to read and/or modify the BCI_ID files per the BCI_Protocol. However, it is implied that no further adaptation is performed under the BCI_Protocol and the EDA tool may complete the simulation starting with the settings determined through training.

If the returned Tx or Rx value is “Error”, the executable model indicating “Error” is unable to understand the messages according to the BCI_Protocol. The Tx and/or Rx AMI_GetWave will stop reading and/or modifying the BCI_ID files. The EDA tool may communicate a protocol error to the user and complete the simulation starting with the settings determined through training. BCI_State must be present if BCI_Protocol is present. BCI_State must be absent if BCI_Protocol is absent.

Other Notes: Training and co-optimization is done by Rx models using one or more Tx equalization exploration algorithms. The Rx model may have Model Specific parameters that allow the user to choose which exploration algorithm to use.

During "Training", the EDA tool may supply a "training" stimulus pattern defined by the user. While not required, the Back-Channel Interface Protocol will likely specify the pattern that should be used.

Example:

```
(BCI_State (Usage InOut)(Type String)
    (List "Off" "Training" "Converged" "Failed" "Error"))
```

Parameter: **BCI_Message_Interval_UI**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx

Descriptors:

Usage:	Info
Type:	Integer
Format:	Value
Default:	<numeric_literal>
Description:	<string >

Definition: This Rx parameter tells the EDA tool the ideal number of UI the model and protocol desire between messaging opportunities.

Usage Rules: BCI_Message_Interval_UI may be used by the EDA tool to manage the size in UI of the block of waveform samples passed to the AMI_GetWave function to provide better synchronization between the times a model has a message to send and the actual timing of the AMI_GetWave block boundaries when messaging may occur. Note that a block is assumed to contain waveform samples for at least one complete UI.

BCI_Message_Interval_UI shall be present for any Rx model where BCI_Protocol is present. BCI_Message_Interval_UI shall be absent if BCI_Protocol is absent.

Other Notes: This parameter allows a BCI_Protocol to define the number of training UI between BCI messages, which necessarily must occur at most once per AMI_GetWave call. Protocols and models implementing them should not expect AMI_GetWave boundaries to occur precisely when a message (e.g., for a Tx adaptation) is ready to be sent. Adaptation engines within the models must therefore be capable of performing correctly without regard to the actual AMI_GetWave block size the EDA tool chooses.

Note that if an adaptation message is ready early in an AMI_GetWave block the adaptation engine must wait for the message to be sent and have an effect on AMI_GetWave results before it can begin to acquire information associated with performance at the new settings to determine the next adaptation. This means the adaptation process is interrupted for the remainder of the AMI_GetWave block, adding to the overall number of UI that must be processed in the time-domain simulation to complete the adaptation. The model maker/protocol designer should choose a value of BCI_Message_Interval_UI that is slightly larger than the smallest number of training UI required per adaptation.

To ensure good messaging efficiency, the EDA tool should consider choosing an AMI_GetWave block size such that either a single AMI_GetWave block or some number of concatenated AMI_GetWave blocks spans a number of UI equal to or slightly larger than BCI_Message_Interval_UI.

Example:

```
(BCI_Message_Interval_UI(Usage Info) (Type Integer) (Value 2048)
      (Description "Training requires at least 2000 UI per adaptation
      message"))
```

Parameter: **BCI_Training_UI**

Required: No, and illegal before AMI_Version 7.0

Direction: Rx

Descriptors:

Usage:	In
Type:	Integer
Format:	Value
Default:	<numeric_literal>
Description:	<string>

Definition: Tells the EDA tool how long the model is likely to take to complete training.

Usage Rules: This parameter is meant for Rx models that support BCI Training. The value in this field tells the EDA tool and the Rx AMI_GetWave function how many UI of the AMI_GetWave output should be expected to complete training.

BCI_Training_UI should be at least twice the value of BCI_Message_Interval_UI to ensure at least one adaptation message can be prepared and delivered.

BCI_Training_UI shall be present for any Rx model where BCI_Protocol is present. BCI_Training_UI must be absent if BCI_Protocol is absent.

Other Notes: The EDA tool may use BCI_Training_UI to terminate an AMI_GetWave simulation due to apparent lack of completion of adaptation.

Adaptation messages must occur at AMI_GetWave block boundaries. Inefficiencies due to mismatch between the time an adaptation is available and the AMI_Getwave boundary (when the change can actually be communicated and effected) will increase the number of UI that adaptation will require. To ensure the EDA tool does not prematurely "time out" an adaptation due to this inefficiency, the value of BCI_Training_UI should be large enough to complete the adaptation. A factor of 2 will generally ensure that any EDA-tool-determined AMI_GetWave block size less than BCI_Message_Interval_UI will still allow adaptation to complete before the simulation time reaches BCI_Training_UI.

Example:

```
(BCI_Training_UI (Usage In) (Type Integer) (Value 100000)
      (Description "BCI training may require 100000 UI"))
```

Parameter: **BCI_Training_Mode**

Required: No, and illegal before AMI_Version 7.1

Direction: Rx, Tx

Descriptors:

Usage:	In
Type:	String
Format:	Value, List
Default:	<string literal>
Description:	<string>

Definition: This parameter tells the EDA tool whether the model supports statistical (AMI_Init-based, using the "Impulse" argument) link training only, time-domain (AMI_GetWave-based, using the "GetWave" argument) link training only, or "Both" (both statistical link training followed by time-domain link training). The only allowed values of BCI_Training_Mode are "Impulse", "GetWave", or "Both". If "Both" is present, then "Impulse" and "GetWave" shall also be present.

Allowed Formats:

(Value "Impulse")

(Value "GetWave")

(List "Impulse")

(List "GetWave")

(List "Impulse" "GetWave")

(List "Impulse" "GetWave" "Both")

Illegal Formats:

(Value "Both")

(List "Both" "Impulse")

(List "Both" "GetWave")

Usage Rules: The user or EDA tool can only choose a BCI_Training_Mode value if it is available on both the Tx and the Rx (and must be set the same for Tx and Rx). If BCI_Protocol is present but BCI_Training_Mode is not present, then the EDA tool shall use a BCI training mode of "GetWave".

To run a BCI statistical training simulation, the Tx and Rx model (or the Terminal Tx, Terminal Rx, and all Repeater Rx and Tx models) must have BCI_Training_Mode as either "Impulse" or "Both".

To run a BCI time-domain training simulation, the Tx and Rx model (or the Terminal Tx, Terminal Rx, and all Repeater Rx and Tx models) must have BCI_Training_Mode as either "GetWave" or "Both".

Training and channel simulation need not use the same mode (e.g., training may be performed statistically, but channel simulation using those training results may be performed in the time domain).

Example:

```
(BCI_Training_Mode (Usage In)(Type String)(List "Both" "Impulse" "GetWave")
      (Description "This Device supports back-channel statistical and time-
       domain optimization"))
```

10.9.1 TRAINING FLOWS

A UNIFIED DESCRIPTION OF ALL AMI TRAINING AND SIMULATION FLOWS

An IBIS-AMI channel consists of:

- A Terminal Tx
- Zero or more Repeaters with each Repeater consisting of:
  - Repeater Rx
  - Repeater Tx
- A Terminal Rx

This forms a daisy chain list of Tx and Rx models.

Every simulation will sequentially execute the AMI_Init function of each model in this daisy chain list.

- The input to each Tx AMI_Init function shall be in accordance with the value of its Tx_Impulse_Input
- The input to each Rx AMI_Init function shall be the cumulative upstream impulse response

If BCI_Training_Mode is set to “Impulse” or “Both” then the simulator will sequentially execute the AMI_Impulse function of each model in this daisy chain list repeatedly.

- The input to each Tx AMI_Impulse function shall be in accordance with the value of its Tx_Impulse_Input and whether the Tx is a Terminal Tx or a Repeater Tx
- The input to each Rx AMI_Impulse function shall be the cumulative upstream impulse response
- The BCI training terminates when BCI_State returns “Converged”, “Fail”, or “Error” in the AMI_parameters_out of the Terminal Rx AMI_Impulse function

If doing time-domain simulation, the simulator will sequentially execute the AMI_GetWave function of each model in this daisy chain list repeatedly. If BCI_Training_Mode is set to “Both” or “GetWave” then the training will continue until the conditions described below have been reached.

- If BCI_Training_Mode is set to “Both” or “GetWave” then the training will finish when BCI_State returns “Converged”, “Fail”, or “Error” in the AMI_parameters_out parameter string of the Terminal Rx AMI_GetWave function. Then the EDA tool can start to accumulate waveform statistics.

- If BCI_Training_Mode is not set to “Both” then the EDA tool can start to accumulate waveform statistics after the EDA tool has completed simulation of a number of UI equal to Ignore_Bits.

STATISTICAL ONLY TRAINING FLOW FOR CHANNELS WITH NO REPEATERS (BCI_TRAINING_MODE SET TO “IMPULSE”)

The EDA tool shall make calls as described below to the Tx AMI_Init and Rx AMI_Init functions and Tx AMI_Impulse and Rx AMI_Impulse functions.

1. Tx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State “Training”) (BCI_Protocol “<name>”) (BCI_ID “<my_ ID>”) (BCI_Training_Mode “Impulse”)

   If BCI_Training_Mode is “GetWave”, follow the flow in Section 10.9.1. The impulse_matrix argument contains the impulse response of the channel. If the Tx executable model does not implement the BCI_Protocol and BCI_Training_Mode, it returns “Error” in BCI_State.

2. Rx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State “Training”) (BCI_Protocol “<name>”) (BCI_ID “<my_ID>”) (BCI_Training_Mode “Impulse”)

   The impulse_matrix argument contains the impulse response output of Tx AMI_Init. If the Rx executable model does not implement the BCI_Protocol and BCI_Training_Mode, it returns “Error” in BCI_State. The EDA tool may analyze the results of Rx AMI_Init.

3. Tx AMI_Impulse is called with the same impulse_matrix used in the call to Tx AMI_Init. The content of BCI_parameters_in shall be the empty string (“”) on the first call to Tx AMI_Impulse. On subsequent calls to Tx AMI_Impulse, the value of BCI_parameters_in shall be the value returned in BCI_parameters_out by the previous call to Rx AMI_Impulse.
4. Rx AMI_Impulse is called using the impulse_matrix output of Tx AMI_Impulse. The value of BCI_parameters_in shall be set to the value of BCI_parameters_out of the previous call to Tx AMI_Impulse.
5. Steps 3 and 4 are repeated until the Rx AMI_Impulse returns AMI Reserved Parameter BCI_State “Converged”, “Failed”, or “Error”
   a. “Converged”: The impulse_matrix and AMI_parameters_out returned by the receiver AMI_Impulse function are used by the EDA tool to complete the simulation.
   b. “Fail”: This tells the EDA tool that the training failed to converge. The EDA tool can terminate the simulation or proceed using the outputs of AMI_Impulse or AMI_Init to complete the simulation at its own risk.

c. "Error": Tells the EDA tool that an error was detected that prevented optimization to continue. The EDA tool can terminate the simulation or proceed using the outputs of AMI_Impulse or AMI_Init to complete the simulation at its own risk.

6. Even if the user selected time-domain simulation after statistical (AMI_Impulse) training, since BCI_Training_Mode is "Impulse", the Tx and Rx AMI_GetWave functions shall not perform training and shall return BCI_State "Off".
7. Tx AMI_Close and Rx AMI_Close are called.

STATISTICAL ONLY TRAINING FLOW FOR CHANNELS WITH REPEATERS (BCI_TRAINING_MODE SET TO "IMPULSE")

1. The AMI_Init flow is identical to the flow defined for the statistical simulation flow for a link containing a Repeater as shown in Figure 45 and documented in Section 10.8.2.
2. This same flow is repeated with the calls to AMI_Init replaced by calls to AMI_Impulse. Note that the EDA tool shall set the value of BCI_parameters_in to the value of BCI_parameters_out of the previous call to an AMI_Impulse function in the channel.
3. The BCI training terminates when BCI_State returns "Converged", "Fail", or "Error" in the AMI_parameters_out of the Terminal Rx AMI_Impulse function.

TIME-DOMAIN ONLY TRAINING FLOW FOR CHANNELS WITH NO REPEATERS (BCI_TRAINING_MODE SET TO "GETWAVE")

The EDA tool shall make calls as described below to the Tx AMI_Init and Rx AMI_Init functions and Tx AMI_GetWave and Rx AMI_GetWave functions.

1. Tx AMI_Init is called by the EDA tool using the following settings in the AMI parameter string:

   (BCI_State "Training") (BCI_Protocol "<name>") (BCI_ID "<my_ ID>")

   If the Tx executable model does not implement the BCI_Protocol, it returns "Error" in BCI_State. The Tx executable model may write a message file in the BCI_ID namespace under BCI_Protocol.
2. Rx AMI_Init is called by the EDA tool using the following settings in the AMI parameter string:

   (BCI_State "Training") (BCI_Protocol "<name>") (BCI_ID "<my_ID>")
   (BCI_Training_UI <# Training Bits>)

   If the Rx executable model does not implement BCI_Protocol, it returns "Error" in BCI_State. The Rx executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.

3. Tx AMI_GetWave is called by the EDA tool with the stimulus pattern passed in. The Tx executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
4. Rx AMI_GetWave is called by the EDA tool with a waveform that is the convolution of the output of Tx AMI_GetWave and the impulse response of the channel. The Rx executable model may read, write, modify and/or delete message files under BCI_Protocol.
5. Steps 3 and 4 are repeated until the EDA tool stops the simulation. The EDA tool should start processing the output of Rx AMI_GetWave after Ignore_Bits and either:

   after BCI_Training_UI, or

   when the Rx AMI_GetWave function returns BCI_State "Converged" or "Failed" or either the Tx or Rx executable model returns "Error".
6. Tx and Rx AMI_Close are called.

Note that the EDA tool does not need to perform any operations specifically assisting the BCI communication between the Tx and the Rx executable models beyond passing the BCI parameters to both executable models on AMI_Init.

TIME-DOMAIN ONLY TRAINING FLOW FOR CHANNELS WITH REPEATERS (BCI_TRAINING_MODE SET TO "GETWAVE")

The EDA tool shall make the following calls to the Upstream Tx, Repeater Rx, Repeater Tx, Downstream Rx AMI_Init and AMI_GetWave functions.

1. Upstream Tx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State "Training") (BCI_Protocol "<name>") (BCI_ "<my_ID>")

   If the executable model does not implement the BCI_Protocol, it returns "Error" in BCI_State. The executable model may write a message file in the BCI_ID namespace under BCI_Protocol.
2. Repeater Rx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State "Training") (BCI_Protocol "<name>") (BCI_ID "<my_ID>")

   If the executable model does not implement the BCI_Protocol, it returns "Error" in BCI_State. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
3. Repeater Tx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

(BCI_State “Training”) (BCI_Protocol “<name>”) (BCI_ID “<my_ID>”)

If the executable model does not implement the BCI_Protocol, it returns “Error” in BCI_State. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.

4. Downstream Rx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State “Training”) (BCI_Protocol “<name>”) (BCI_ID “<my_ID>”) (BCI_Training_UI <# Training Bits>)

   If the executable model does not implement the BCI_Protocol, it returns “Error” in BCI_State. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
5. Upstream Tx AMI_GetWave is called with the stimulus pattern. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
6. Repeater Rx AMI_GetWave is called with the Upstream Tx AMI_GetWave function’s output waveform combined with the Upstream Channel Impulse Response. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
7. Repeater Tx AMI_GetWave is called with the waveform output of the Repeater Rx AMI_GetWave. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
8. Downstream Rx AMI_GetWave is called with the Repeater Tx AMI_GetWave function’s output waveform combined with the Downstream Channel Impulse Response. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol
9. Steps 5 through 8 are repeated until the EDA tool stops the simulation.
   a. The EDA tool should start processing the output of Rx AMI_GetWave after Ignore_Bits and either:

      after BCI_Training_UI, or

      when the downstream Rx AMI_GetWave function returns BCI_State “Converged” or “Failed” or any executable model in the channel returns “Error”.

Note that it is the responsibility of the BCI _Protocol to define the BCI message files and contents therein so that each executable model in the channel can determine its role/position in the channel optimization.

COMBINED STATISTICAL AND TIME-DOMAIN TRAINING FLOW FOR CHANNELS WITH NO REPEATER (BCI_TRAINING_MODE SET TO "BOTH")

The EDA tool shall make calls as described below to the Tx AMI_Init and Rx AMI_Init functions and Tx AMI_Impulse and Rx AMI_Impulse functions.

1. Tx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State "Training") (BCI_Protocol "<name>") (BCI_ID "<my_ ID>")
   (BCI_Training_Mode "Both" or "Impulse")

   If (BCI_Training_Mode "GetWave") follow the flow in Section 10.9.1. The impulse_matrix argument contains the impulse response of the channel. If the Tx executable model does not implement the BCI_Protocol and BCI_Training_Mode, it returns "Error" in BCI_State.
2. Rx AMI_Init is called by the EDA tool using the following settings as part of the AMI parameter string:

   (BCI_State "Training") (BCI_Protocol "<name>") (BCI_ID "<my_ID>")
   (BCI_Training_Mode "Both" or "Impulse")

   The impulse_matrix argument contains the impulse response output of Tx AMI_Init. If the Rx executable model does not implement the BCI_Protocol and BCI_Training_Mode, it returns "Error" in BCI_State. The EDA tool may analyze the results of Rx AMI_Init.
3. Tx AMI_Impulse is called with the same impulse_matrix used in the call to Tx AMI_Init. The value of BCI_parameters_in shall be Null (0) on the first call to Tx AMI_Impulse and the value of BCI_parameters_out of the previous call to Rx AMI_Impulse on subsequent calls to Tx AMI_Impulse.
4. Rx AMI_Impulse is called using the impulse_matrix output of Tx AMI_Impulse. The value of BCI_parameters_in shall be set to the value of BCI_parameters_out of the previous call to Tx AMI_Impulse.
5. Steps 3 and 4 are repeated until the Rx AMI_Impulse returns AMI Reserved Parameter BCI_State "Converged", "Failed", or "Error"
   a. "Converged": The impulse_matrix and AMI_parameters_out returned by the receiver AMI_Impulse function are used by EDA tool to complete the simulation.
   b. "Fail": This tells the EDA tool that the training failed to converge. The EDA tool can terminate the simulation or proceed using the outputs of AMI_Impulse or AMI_Init to complete the simulation at its own risk.
   c. "Error": Tells the EDA tool that an error was detected that prevented optimization to continue. The EDA tool can terminate the simulation or proceed using the outputs of AMI_Impulse or AMI_Init to complete the simulation at its own risk.
6. As BCI_Training_Mode is "Both", the time-domain training as described above in this section is invoked.

7. Tx AMI_Close and Rx AMI_Close are called.

COMBINED STATISTICAL AND TIME-DOMAIN TRAINING FLOW FOR CHANNELS WITH REPEATERS (BCI_TRAINING_MODE SET TO "BOTH")

The EDA tool shall make calls as described below to the Tx AMI_Init and Rx AMI_Init functions and Tx AMI_Impulse and Rx AMI_Impulse functions and the Tx AMI_GetWave and Rx AMI_GetWave functions.

1. The AMI_Init flow is identical to the flow defined for the statistical simulation flow for a link containing a Repeater as shown in Figure 45 and documented in Section 10.8.2.
2. This same flow is repeated with the calls to AMI_Init replaced by calls to AMI_Impulse.
3. The BCI training terminates when BCI_State returns "Converged", "Fail", or "Error" in the AMI_parameters_out of the Terminal Rx AMI_Impulse function.
4. Upstream Tx AMI_GetWave is called with the stimulus pattern. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
5. Repeater Rx AMI_GetWave is called with the Upstream Tx AMI_GetWave function's output waveform combined with the Upstream Channel Impulse Response. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
6. Repeater Tx AMI_GetWave is called with the waveform output of the Repeater Rx AMI_GetWave. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol.
7. Downstream Rx AMI_GetWave is called with the Repeater Tx AMI_GetWave function's output waveform combined with the Downstream Channel Impulse Response. The executable model may read, write, modify and/or delete message files in the BCI_ID namespace under BCI_Protocol
8. Steps 5 through 8 are repeated until the EDA tool stops the simulation.
    a. The EDA tool should start processing the output of Rx AMI_GetWave after Ignore_Bits and either:

        after BCI_Training_UI, or

        when the downstream Rx AMI_GetWave function returns BCI_State "Converged" or "Failed" or any executable model in the channel returns "Error".

Note that it is the responsibility of the BCI _Protocol to define the BCI message files and contents therein so that each executable model in the channel can determine its role/position in the channel optimization.

10.9.2 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Table 34 – General Rules and Allowable Usage for BCI Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage				
	Required	Default[2,5]	Info	In	Out	Dep[1]	InOut
BCI_Message_Interval_UI	No[4]	--	X				
BCI_ID	No[3]	--		X			
BCI_Protocol	No	--		X			
BCI_State	No[3]	--					X
BCI_Training_UI	No[4]	--		X			
BCI_Training_Mode	No[6]	"GetWave"		X			

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) Required if BCI_Protocol is present.
4) Required for Rx models if BCI_Protocol is present.
5) "--" means that an entry must be provided if the parameter is present; no default is assumed or permitted.
6) Illegal for AMI_Version 7.0 and earlier, required if BCI_Protocol supports statistical training.

Table 35 – Allowable Data Types for BCI Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
BCI_Message_Interval_UI			X		
BCI_ID				X	
BCI_Protocol				X	
BCI_State				X	
BCI_Training_UI			X		
BCI_Training_Mode				X	

Table 36 – Allowable Data Formats for BCI Reserved Parameters

Reserved Parameter	Data Format									
	Value	**Range**	**Corner**	**List**	**Increment**	**Steps**	**Gaussian**	**Dual-Dirac**	**DjRj**	**Table**
BCI_Message_Interval_UI	X									
BCI_ID	X									
BCI_Protocol	X			X						
BCI_State				X						
BCI_Training_UI	X									
BCI_Training_Mode	X			X						

10.10 ALTERNATIVE AMI ANALOG BUFFER MODELING

This section discusses an alternative analog buffer modeling technique, specifically designed for AMI applications. The approach uses 4-port analog circuit data provided in a Touchstone file specified by the AMI parameter named Ts4file (note: Ts4file implies a restricted Touchstone format, where the number of ports is four and the port numbering is predefined).

10.10.1 TRANSMITTER ANALOG CIRCUIT

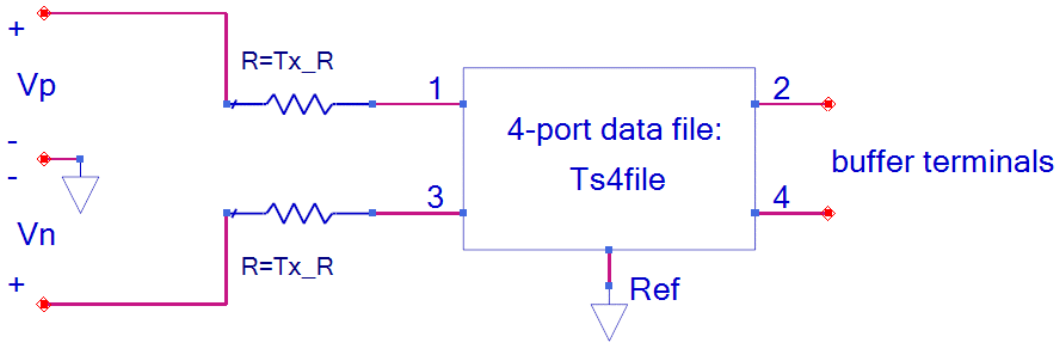


Figure 46 – Transmitter Analog Circuit, Default Port Ordering

For logic level 1, Vp=Tx_V / 2 and Vn=-Tx_V / 2, where Tx_V is a Reserved Parameter (defined below). For logic level 0, Vp=-Tx_V / 2 and Vn=Tx_V / 2. The ideal step stimulus is a differential voltage waveform Vp - Vn when the logic level is switched from 0 to 1. This may be used to determine the impulse response needed for the AMI flow. For Tx models that have the Reserved Parameter Ts4file, the Reserved Parameter Tx_V is required, and the Reserved Parameter Tx_R is optional (default is 0.0 ohms). For a Tx buffer, the transmitter circuit defines the analog buffer model between the zero-impedance stimulus input voltage source and the buffer terminals.

Ports 1, 2, 3 and 4 of the 4-port network are between the nodes 1, 2, 3 and 4 and the common reference node Ref, respectively. For the default port ordering, ports 1 and 3 are at the stimulus source side, and ports 2 and 4 are the transmitter analog buffer model's output. Furthermore, for the default port ordering, ports 1 and 2 correspond to the non-inverting signal path and ports 3 and 4 to the inverting signal path. The reference node, represented by the triangle reference symbol in Figure 46, is the reference node A_gnd as defined in this specification.

10.10.2 RECEIVER ANALOG CIRCUIT

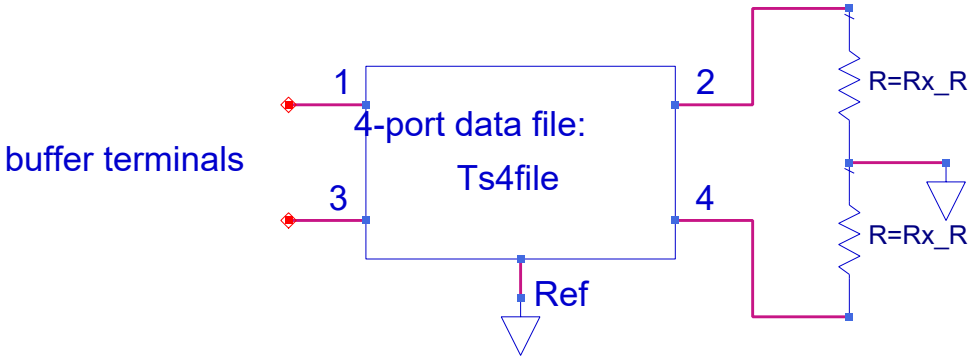


Figure 47 – Receiver Analog Circuit, Default Port Ordering

Ports 1, 2, 3 and 4 of the 4-port network are between the nodes 1, 2, 3 and 4 and the common reference node Ref, respectively. For the default port ordering, ports 1 and 3 are the receiver analog buffer model's input, and the waveforms at ports 2 and 4 are the differential input of the Rx algorithmic model. Furthermore, for the default port ordering, ports 1 and 2 correspond to the non-inverting signal path and ports 3 and 4 to the inverting signal path. The reference node, represented by the triangle reference symbol in Figure 47, is the reference node A_gnd as defined in this specification. For Rx models that have the Reserved Parameter Ts4file, the Reserved Parameter Rx_R is optional (default is open circuit). For an Rx buffer, the receiver circuit defines the analog buffer model between the buffer terminals and the high impedance input of the Rx Algorithmic model.

By definition, the placement of the Ts4file information within .ami files makes the Ts4file data exclusively limited to AMI applications. If the same electrical behavior is desired for non-AMI applications of the same IBIS model (the one referencing the Algorithmic Model) the model maker can optionally provide an equivalent description using the [External Model] keyword. However, the latter is not needed if the model is intended for AMI applications only.

10.10.3 RESERVED PARAMETER DEFINITIONS

Parameter: **Ts4file**

Required: No

Direction: Rx, Tx

Descriptors:

Usage: Info, Dep
Type: String
Format: Value, List, Corner
Default: <string literal>
Description: <string>

Definition: This parameter provides the file reference for a 4-port Touchstone file to be used in the Analog Circuit. See the Analog Circuit definitions above for the port order associated with the Touchstone file data.

Example:

```
(Ts4file (Usage Info)(Type String)(Corner "typ.s4p" "min.s4p" "max.s4p"))
```

Parameter: **Tx_V**

Required: No, unless the .ami file is defined for the Tx direction and **Ts4file** parameter is defined. Illegal otherwise.

Direction: Tx

Descriptors:

Usage: Info, Dep
Type: Float
Format: Value, List, Corner, Range, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: This parameter defines the voltage swing of the stimulus input to the transmitter circuit.

Example:

```
(Tx_V (Usage Info)(Type Float)(Range 1.0 0.5 1.0))
```

Parameter: **Tx_R**

Required: No, illegal if parameter **Ts4file** is not defined.

Direction: Tx

Descriptors:

Usage: Info, Dep
Type: Float
Format: Value, List, Corner, Range, Increment, Steps

Default: <numeric_literal>
Description: <string>

Definition: This parameter is optional and defines the value Tx_R in ohms of the series resistors shown in Figure 46. It can only be present if the .ami file is defined for the Tx direction. If this parameter is not present in the .ami file, the value of Tx_R defaults to zero.

Example:

```
(Tx_R (Usage Info)(Type Float)(Value 0.0))
```

Parameter: **Rx_R**

Required: No, illegal if parameter **Ts4file** is not defined.

Direction: Rx

Descriptors:

Usage: Info, Dep
Type: Float
Format: Value, List, Corner, Range, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: This parameter is optional and defines the value of Rx_R in ohms of the resistors shown in Figure 47. It can only be present if the .ami file is defined for the Rx direction. If this parameter is not present in the .ami file, the value of Rx_R defaults to infinity, or a reasonable approximation thereof.

Example:

```
(Rx_R (Usage Info)(Type Float)(Value 1.0e6))
```

Parameter: **Tx_Port_Order**

Required: No, and illegal before AMI_Version 8.0; illegal if the Ts4file and Tx_V parameters are not defined

Direction: Tx

Descriptors:

Usage: Info
Type: String
Format: Value
Default: <string_literal>
Description: <string>

Definition: Defines the port order for the provided Tx Ts4file parameter 4-port Touchstone file.

Usage Rules: The only input values allowed are “13-24” (stands for IEEE or even/odd port ordering) and “12-34” (stands for Gonzalez or sequential port ordering). “13-24” means that ports 1 and 3 of the Ts4file are connected to the stimulus source side non-inverting and inverting terminals respectively, and ports 2 and 4 are the transmitter analog buffer model’s non-inverting

and inverting outputs respectively to the component package. "12-34" means that ports 1 and 2 of the Ts4file are connected to the stimulus source side non-inverting and inverting terminals respectively, and ports 3 and 4 are the transmitter analog buffer model's non-inverting and inverting outputs respectively to the component package. If omitted, the default value is "13-24". If the Ts4file Reserved Parameter uses Format List or Corner, all of its enumerated files shall use the port order defined by this parameter.

Example:

```
(Tx_Port_Order (Usage Info) (Type String) (Value "13-24")
   (Description "This indicates the port order of the Tx s4p Ts4file")
)
```

The Ts4file transmitter analog circuit port orders and associated Tx_Port_Order entries are shown in the figures below:

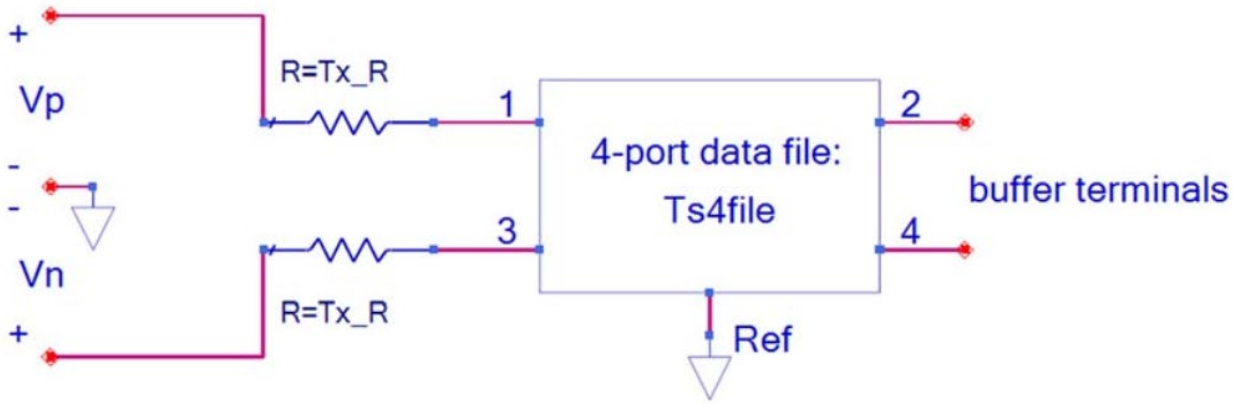


Figure 48 – Transmitter Analog Circuit with Tx_Port_Order 13-24 (Default if Tx_Port_Order is Omitted)

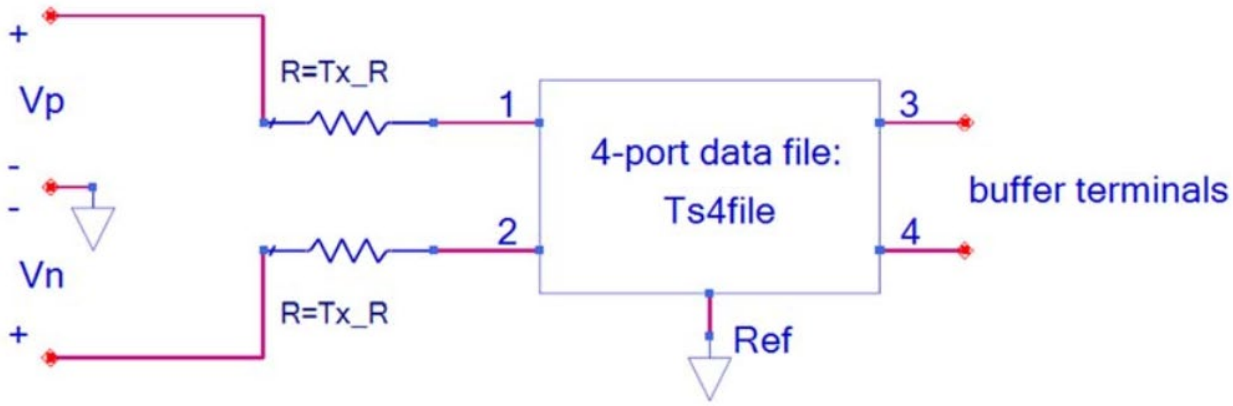


Figure 49– Transmitter Analog Circuit with Tx_Port_Order 12-34

Parameter: **Rx_Port_Order**

Required: No, and illegal before AMI_Version 8.0; illegal if the Ts4file parameter is not defined for the Rx direction

Direction: Rx

Descriptors:

Usage:	Info
Type:	String
Format:	Value
Default:	<string_literal>
Description:	<string>

Definition: Defines the port order for the provided Rx Ts4file parameter 4-port Touchstone file.

Usage Rules: The only input values allowed are "13-24" (stands for IEEE or even/odd port ordering) and "12-34" (stands for Gonzalez or sequential port ordering). "13-24" means that ports 1 and 3 of the Ts4 file are connected to the receiver analog buffer model's inputs, non-inverting and inverting respectively, from the component package, and the waveform at ports 2 and 4, non-inverting and inverting respectively, are the differential input to the Rx algorithmic model. "12-34" means that ports 1 and 2 of the Ts4 file are connected to the receiver analog buffer model's inputs, non-inverting and inverting respectively, from the component package, and the waveform at ports 3 and 4, non-inverting and inverting respectively, are the differential input to the Rx algorithmic model. If omitted, the default value is "13-24". If the Ts4file Reserved Parameter uses Format List or Corner, all of its enumerated files shall use the port order defined by this parameter.

Example:

```
(Rx_Port_Order (Usage Info) (Type String) (Value "13-24")
   (Description "This indicates the port order of the Rx s4p Ts4file")
)
```

The Ts4file receiver analog circuit port orders and associated Rx_Port_Order entries are shown in the figures below:

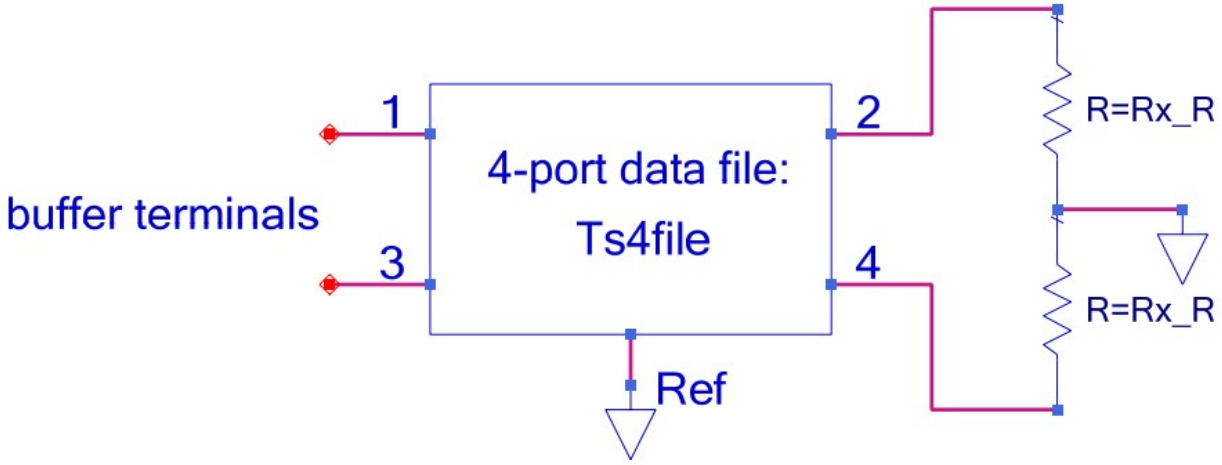


Figure 50 – Receiver Analog Circuit with Rx_Port_Order 13-24 (Default if Rx_Port_Order is Omitted)

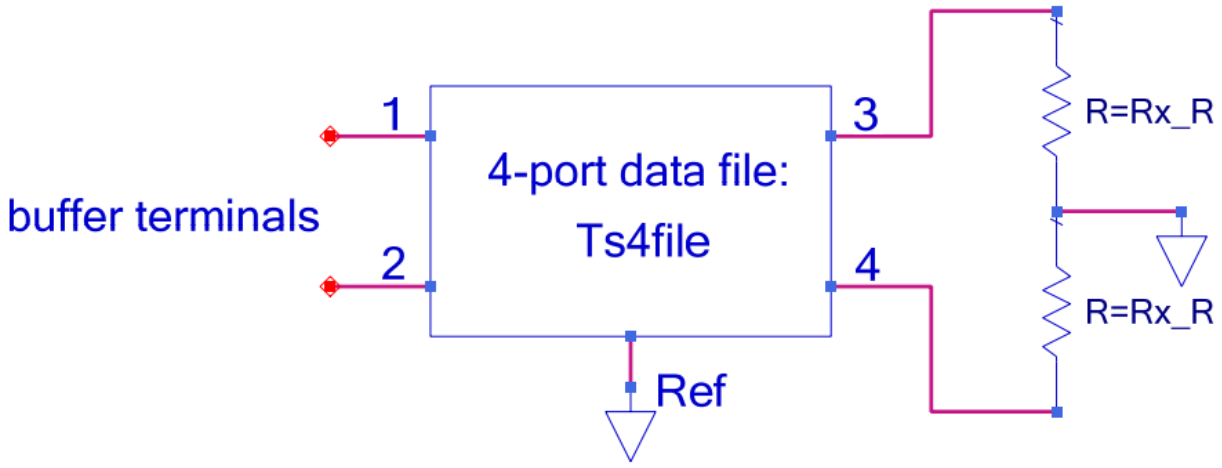


Figure 51 – Receiver Analog Circuit with Rx_Port_Order 12-34

10.10.4 SUMMARY TABLES FOR USAGE, TYPE AND FORMAT

Table 37 – General Rules and Allowable Usage for Alternative Analog Modeling Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage				
	Required	Default[2,4]	Info	In	Out	Dep[1]	InOut
Ts4file	No	--	X			X	
Tx_V	No[3,5]	--	X			X	
Tx_R	No[5]	0	X			X	
Rx_R	No[5]	Infinity	X			X	
Tx_Port_Order	No	"13-24"	X				
Rx_Port_Order	No	"13-24"	X				

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) Required if Ts4file is present for a Tx model.
4) "--" means that an entry must be provided if the parameter is present; no default is assumed or permitted.
5) Illegal if Ts4file is not present.

Table 38 – Allowable Data Types for Alternative Analog Modeling Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Ts4file				X	

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Tx_V	X				
Tx_R	X				
Rx_R	X				
Tx_Port_Order				X	
Rx_Port_Order				X	

Table 39 – Allowable Data Formats for Alternative Analog Modeling Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
Ts4file	X		X	X						
Tx_V	X	X	X	X	X	X				
Tx_R	X	X	X	X	X	X				
Rx_R	X	X	X	X	X	X				
Tx_Port_Order	X									
Rx_Port_Order	X									

10.11 AMI TEST CONFIGURATION

To assist in ensuring that algorithmic model behavior is identical to what was intended by the model maker, the [AMI Test Configuration] keyword defines input and output test data for use with algorithmic model simulation. The keyword's subparameters include the required information that would be passed by the simulation tool into the algorithmic models for which the keyword is specified. The keyword also supports providing input waveforms, where appropriate, and reporting associated output waveforms generated by the model-maker as separate files. Note that channel correlation is not covered by [AMI Test Configuration]; the keyword includes a complete channel response as part of the input.

10.11.1 KEYWORD DEFINITION

Keywords: **[AMI Test Configuration]**

Required: No, and illegal before IBIS 7.3.

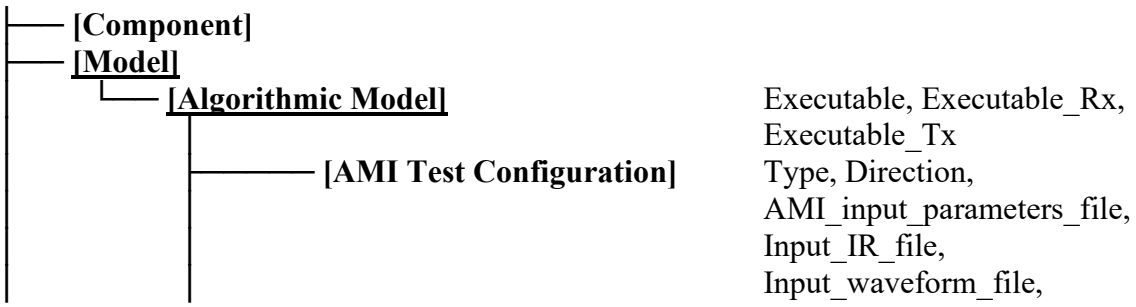
Description: Defines example input stimulus and resulting output waveform data plus AMI configuration information for testing of the associated [Algorithmic Model].

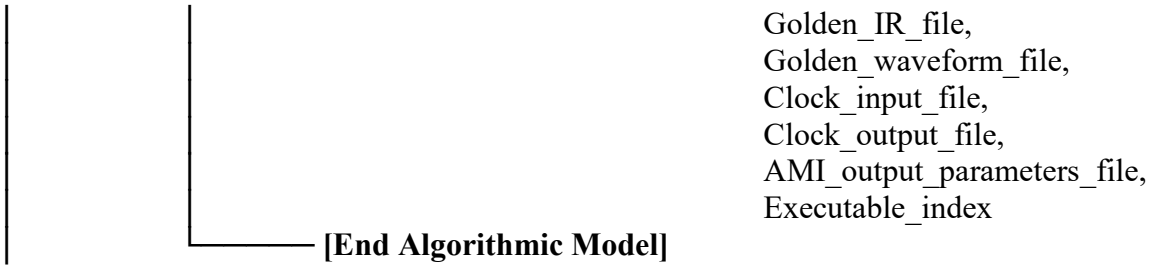
The intent of [AMI Test Configuration] is to enable direct comparison by a simulation tool of the output of the associated AMI model, with appropriate configuration and stimulus information, against a "Golden Waveform" provided by the model maker. The enabled comparison is solely at the AMI level and does not involve simulating using a channel model or buffer loading.

Sub-Params: Type, Direction, AMI_input_parameters_file, Input_IR_file, Input_waveform_file, Golden_IR_file, Golden_waveform_file, Clock_input_file, Clock_output_file, AMI_output_parameters_file, Executable_index

Usage Rules: The [AMI Test Configuration] keyword takes a single name argument, which is required, as it supports selection of specific input and output configurations for a given buffer and test waveform combination. The [AMI Test Configuration] argument shall be fewer than 1000 characters in length and shall not be duplicated for any other [AMI Test Configuration] keywords appearing under the same [Algorithmic Model] keyword. A given [Algorithmic Model] may contain any number of [AMI Test Configuration] sections, representing different simulation conditions.

The placement of this keyword within the hierarchy of IBIS is shown below:





The Type subparameter is required, and its arguments shall be one of either "Statistical" or "Time_domain". This indicates the kind of configuration data required, the impulse response and/or waveform input data required, and impulse response or waveform output to expect from a simulation of the buffer being tested.

The Direction subparameter is required. Its argument specifies the direction of the "device under test" and shall be one of the following strings: "Tx" or "Rx". Note that using "Tx" for a buffer which is incapable of a driving mode of operation is illegal. Similarly, using "Rx" for a buffer which is incapable of a receiving mode of operation is illegal. Use of "Tx" or "Rx" for an I/O buffer will identify the mode of operation for the test information and waveform(s).

The AMI_input_parameters_file subparameter is required. This subparameter takes a single string argument corresponding to a filename present in the same directory as the calling .ibs file. The file's contents consist of rows of space-separated string data, using formatting rules identical to that of the AMI_parameters_in string. This data is grouped in two sections. The first section, which uses the required identifier "Simulator_parameters", includes those parameters and values which are needed to unambiguously define the simulation conditions used to generate the golden waveforms provided and, in a normal simulation context, would be supplied by the user through a simulator user interface (i.e., AMI function signature parameters). The second section, which uses the required identifier "Model_parameters", includes the AMI_parameters_in string contents that would be generated by an EDA tool and passed to the model, specifically the Usage In and Usage InOut parameters from the .ami file contents and their associated values.

The structure is shown below; note that the line feeds shown may be omitted as in the AMI_parameters_in string.

For Type Statistical waveforms, the "Simulator_parameters" section shall include four required AMI_Init and/or AMI_Impulse function signature parameters: Sample_interval, Symbol_time, Number_of_rows, and Aggressors, as shown below.

```
(Simulator_parameters
        (Sample_interval <float>)
        (Symbol_time <float>)
        (Number_of_rows <integer>)
        (Aggressors <integer>)
)
(Model_parameters
        (<root name>
                (<Usage In or InOut parameter name> <value>)
                (<Usage In or InOut parameter name> <value>)
```

```
            (<Usage In or InOut parameter name> <value>)
            …
        )
)
```

For Type Time_domain waveforms, the "Simulator_parameters" section shall include three required AMI_GetWave and AMI_Init and/or AMI_Impulse function signature parameters: Sample_interval, Symbol_time, and Wave_size, as shown below.

```
(Simulator_parameters
        (Sample_interval <float>)
        (Symbol_time <float>)
        (Wave_size <integer>)
)
(Model_parameters
        (<root name>
            (<Usage In or InOut parameter name> <value>)
            (<Usage In or InOut parameter name> <value>)
            (<Usage In or InOut parameter name> <value>)
            …
        )
)
```

All Usage In and InOut parameters supported by the associated model's .ami file shall be included in the AMI_input_parameters_file contents. Note that these Usage In and InOut parameters are preceded by several parameters supplied as part of the AMI_Init and AMI_GetWave function calls; the values supplied as part of those calls should match those provided in the AMI_input_parameters_file.

```
long AMI_Init (double *impulse_matrix, long number_of_rows, long
aggressors, double sample_interval, double symbol_time, char
*AMI_parameters_in, char **AMI_parameters_out, void **AMI_memory_handle,
char **msg)
```

```
long AMI_GetWave (double *wave, long wave_size, double *clock_times,
char **AMI_parameters_out, void *AMI_memory)
```

Though not a requirement of AMI_GetWave simulations generally, for the purposes of test generation using [AMI Test Configuration], wave_size shall be a fixed value across all calls to AMI_GetWave (see below).

No Usage Out or Usage Info parameters shall be listed in the AMI_input_parameters_file contents; only Usage In and Usage InOut parameters shall be provided.

The Input_IR_file subparameter is required for all Direction and Type combinations. This subparameter takes a single string argument corresponding to a filename present in the same directory as the calling .ibs file.

The Input_IR_file contents consist of rows of whitespace-separated numeric data representing an impulse response. The values shall be floating-point numbers in the standard ANSI "C" notation (e.g., 2.0e-9). Each file shall contain at least one row. The vectors contain the impulse responses for the channel connected to the model being characterized, and optionally the crosstalk impulse responses from aggressor channels. The number of columns in the Input_IR_file shall be at least 1 and shall be equal to 1 plus the value of the Aggressors parameter.

The Input_waveform_file subparameter is required for Type Time_domain and is prohibited for Type Statistical. The subparameter takes a single string argument corresponding to a filename present in the same directory as the calling .ibs file.

The Input_waveform_file contents consist of rows of numeric data representing voltages. The values shall be floating-point numbers in the standard ANSI "C" notation (e.g., 2.0e-9). The contents of the Input_waveform_file are equivalent to the contents of the "wave" argument of the AMI_GetWave function; these are described in the functional reference flows of Section 10.2.2. The Input_waveform_file contents for Direction Rx are equivalent to the input to the Rx AMI_GetWave function during a normal channel simulation that includes both Tx and Rx AMI executables.

The time duration represented by the contents of the Input_waveform_file shall match the time duration represented by the contents of the Golden_waveform_file (see below). Note that the time values are implied based on the assumption of simulation starting at time = 0 and the use of a fixed timestep specified by the Sample_interval parameter.
The number of columns in the contents of the Input_waveform_file shall be 1.

The Golden_IR_file subparameter is required for Type Statistical and is optional for Type Time_domain. Its argument is a single string corresponding to a filename present in the same directory as the calling .ibs file. The Golden_IR_file contents shall consist of rows of whitespace-separated numeric data representing an impulse response. In the case of Type Statistical, this data will be compared to the output generated by the algorithmic model after the input IR data has been processed by the AMI_Init function. The values shall be floating-point numbers in the standard ANSI "C" notation (e.g., 2.0e-9).

The number of values per row (columns) in the Golden_IR_file contents shall correspond to the sum of the number of crosstalk channels represented plus 1 for the main communication channel; as the data represents a matrix of impulse responses, a time column is not included. Note that the time values are implied based on the assumption of simulation starting at time = 0 and the use of a fixed timestep specified by the Sample_interval parameter. The number of whitespace-separated columns in the contents of the Golden_IR_file shall match the number of whitespace-separated columns in the contents of the Input_IR_file. The time duration represented by the contents of the Input_IR_file and the Golden_IR_file shall be identical.

The Golden_waveform_file subparameter is required for Type Time_domain and is prohibited for Type Statistical. The data consists of a single column of analog waveform values representing the concatenated output of an integer number of calls to the associated model's AMI_GetWave function if it exists. The values shall be floating-point numbers in the standard ANSI "C" notation (e.g., 2.0e-9). Data will be generated through successive calls to the AMI_GetWave function, if

present, where the number of samples in the data passed to and received from the function in each call is specified by wave_size. The time duration represented by the contents of the Input_waveform_file and the Golden_waveform_file shall be identical. Note that the time values are implied based on the assumption of simulation starting at time = 0 and the use of a fixed timestep specified by the Sample_interval parameter.

Note that time domain simulations, and therefore Type Time_domain comparisons, can be executed if AMI_GetWave is not present for a given model; in this case, time domain data is generated by the EDA simulation tool by convolving bitstream stimulus information as provided in the Input_waveform_file and an impulse response. In the case where Init_Returns_Impulse for the model is True, this impulse response is the filtered output from the statistical model represented by the AMI_Init function. In this situation, providing Golden_IR_file data would be strongly recommended for Type Time_domain, to allow direct examination of the filtered impulse response. Type Time_domain analysis is possible in cases where Init_Returns_Impulse for the model is False. Here, the model does not filter the incoming impulse response, so the bitstream stimulus information as provided in the Input_waveform_file would be convolved with the Input_IR_file to produce the Golden_waveform_file. Golden_IR_file data, if present, would be identical to the Input_IR_file data.

The Clock_input_file subparameter is required if the Direction subparameter value is “Rx”, the associated model uses the Rx_Use_Clock_Input parameter, and the Type subparameter value is “Time_domain”. Clock_input_file is prohibited for all other combinations of Direction and Type. Its argument is a single string corresponding to a filename present in the same directory as the calling .ibs file. The contents of the Clock_input_file consist of multiple rows of numeric data representing a vector of clock information provided to the Rx model; note that only a single value (a single column) is present per row. In cases where Rx_Use_Clock_Input is set to “Times”, multiple blocks of clock information comprise the file, with adjacent blocks separated by a value of -1, and the entire file ending with the -1 value. The number of clock values may differ from the number of symbol times in the waveform whose size (number of samples) is given by wave_size. The vector shall contain increasing clock times (except for the values of -1) which may or may not increase according to a regular interval. With the exception of the values of -1, the values in the file shall be floating-point numbers in the standard ANSI “C” notation (e.g., 2.0e-9).
In cases where Rx_Use_Clock_Input is set to “Wave”, the contents of the Clock_input_file are the input clock waveform to the buffer, where the time duration represented by the contents of the Clock_input_file shall match the time duration represented by the associated Golden_waveform_file. The Clock_input_file contents shall use a uniform time step identical to that in the contents of the Golden_waveform_file. The values in the file shall be floating-point numbers in the standard ANSI “C” notation (e.g., 2.0e-9). Only one value per row is permitted.
The Clock_output_file subparameter is required if both the Direction subparameter value is “Rx” and the Type subparameter value is “Time_domain”. Clock_output_file is prohibited for all other combinations of Direction and Type. Its argument is a single string corresponding to a filename present in the same directory as the calling .ibs file. The Clock_output_file contents consist of rows of numeric values representing a vector of clock output information from the model. Note that multiple blocks of clock information comprise the file, with adjacent blocks separated by a value of -1. The Clock_output_file shall have at least one row; in the case of the contents consisting of a single row, the value shall be -1 (representing a receiver that does not generate clock

output). The final value of the Clock_output_file time values shall be -1. With the exception of the -1 values, the values in the file shall be floating-point numbers in the standard ANSI "C" notation (e.g., 2.0e-9); while representing the -1 values in floating-point format is permitted, it is not preferred. The number of clock values may differ from the number of symbol times in each waveform block, whose size is given by wave_size.

The AMI_output_parameters_file subparameter is required. Its argument is a single string corresponding to a filename present in the same directory as the calling .ibs file.

The AMI_output_parameters_file contents consist of one or more rows of whitespace-separated string data, using formatting rules identical to that of the AMI_parameters_out string. This data contains the AMI_parameters_out string contents that would be generated by the model and passed to the EDA tool. This includes the Usage Out and Usage InOut parameters from the model AMI file contents and their associated values; in the absence of Out or InOut parameters, the file contents would consist only of the root name. An example of the structure for Type Statistical is shown below; note that the line feeds shown may be omitted as in the AMI_parameters_out string.

(<root name>
    (<Usage Out or InOut parameter name> <value>)
    (<Usage Out or InOut parameter name> <value>)
    (<Usage Out or InOut parameter name> <value>)
    …
)

For Type Time_domain, because outputs will be generated for each group, or "block", of wave_size samples being processed by the AMI_GetWave model, the AMI_output_parameters_file will consist of groups of parameters which are repeated in each group, with different values as appropriate; the parameters output from the first instance of wave_size data will be appended to the parameters output from execution of the AMI_Init function, the parameters output from the second instance of wave_size data will be appended to the output of the first, the third will be appended to the output of the second, and so on. Each parameter in each block of output parameters shall be preceded by an integer block index value, starting with one and increasing by one with each subsequent block added. This preserves the data for each block allowing correlation of output data as it changes across blocks.

    (<root name>
        (<block index> <Usage Out or InOut parameter name 1> <value>)
        (<block index> <Usage Out or InOut parameter name 2 > <value>)
        (<block index> <Usage Out or InOut parameter name 3> <value>)
        …
        (<block index+1> <Usage Out or InOut parameter name 1> <value>)
        (<block index+1> <Usage Out or InOut parameter name 2> <value>)
        (<block index+1> <Usage Out or InOut parameter name 3> <value>)
        …
)

The Executable_index parameter is required for all Direction and Type combinations. Its argument is a single non-zero positive integer. This value corresponds to the specific Executable, Executable_Rx, and/or Executable_Tx lines under the same [Algorithmic Model] keyword, listing

the architecture, compiler, etc. combination used to generate the input and output data provided for the [AMI Test Configuration] keyword. The integer argument to Executable_index shall not be larger than the number of Executable, Executable_Rx, and/or Executable_Tx lines under the same [Algorithmic Model] keyword.

Other Notes: Note that "Statistical" and "Time_domain" Types are distinct from the capabilities supported by the AMI_Init/AMI_Impulse and AMI_GetWave functions. Type Statistical assumes that statistical processing is performed by the model through the AMI_Init function; Type Statistical is supported for models regardless of whether they contain or omit the AMI_GetWave function. AMI_Init is a required function of all models, but not all models containing AMI_Init support Statistical analysis. As time domain simulations often involve both AMI_Init and AMI_GetWave functions, it is assumed that the EDA tool will execute the appropriate AMI_Init function for named models using Type Time_domain before executing AMI_GetWave, if present. As noted above, Type Time_domain may be supported for a model only including the AMI_Init function (and excluding AMI_GetWave) through the EDA simulation tool convolving the input stimulus bitstream pattern with an impulse response.

The EDA tool is assumed to pass the contents of the Clock_input_file into the Rx AMI executable models in cases where the associated model's AMI parameter file contains "Rx_Use_Clock_Input" values of "Time" or "Wave".

The structure of the [AMI Test Configuration] keyword is intended to ensure that output can be generated unambiguously by any EDA tool using the Input_IR_file, AMI_input_parameters_file, and, where appropriate, Input_waveform_file and/or Clock_input_file provided, for comparison against the contents of the Golden_IR_file, Golden_waveform_file, AMI_output_parameters_file, and Clock_output_file (where appropriate).

The permitted combinations of [AMI Test Configuration] Direction and Type subparameters with other subparameters are shown in the table below.

Table 40 – [AMI Test Configuration] Subparameter Rules

Subparameter	Direction	Type	Required, Optional, or Prohibited
Input_IR_file	Rx, Tx	Statistical, Time_domain	Required
AMI_input_parameters_file	Rx, Tx	Statistical, Time_domain	Required
Input_waveform_file	Rx, Tx	Statistical	Prohibited
	Rx, Tx	Time_domain	Required
Golden_IR_file	Rx, Tx	Statistical	Required
	Rx, Tx	Time_domain	Optional

Golden_waveform_file	Rx, Tx	Statistical	Prohibited
	Rx, Tx	Time_domain	Required
Clock_input_file	Rx	Time_domain	Required if the Rx_Use_Clock_Input parameter is present, Prohibited otherwise
	Tx	Statistical, Time_domain	Prohibited
Clock_output_file	Rx	Time_domain	Required
	Tx	Statistical, Time_domain	Prohibited
AMI_output_parameters_file	Rx, Tx	Statistical, Time_domain	Required
Executable_index	Rx, Tx	Statistical, Time_domain	Required

Examples:

```
[Algorithmic Model]
Executable_Tx Solaris_cc_64 libtx_getwave.so tx_getwave.ami
Executable_Rx Solaris_cc_64 librx_getwave.so rx_getwave.ami
|
Executable_Tx Windows_VisualStudio_64 tx_getwave.dll tx_getwave.ami
Executable_Rx Windows_VisualStudio_64 rx_getwave.dll rx_getwave.ami
|
| The Executable_index value below identifies the data for AMI Test
| Configuration as having been collected using the libtx_getwave.so
| executable.
|
[AMI Test Configuration]  Typ_corner_statistical
Type Statistical
Direction Tx
Input_IR_file  four_tap_input_IR.txt
AMI_input_parameters_file  four_tap_tx_params_typ_stat.txt
Golden_IR_file  four_tap_output_IR_typ.txt
AMI_output_parameters_file  four_tap_output_params_typ_stat.txt
Executable_index 1
|
| The Executable_index value below identifies the data for AMI Test
| Configuration as having been collected using the tx_getwave.dll
| executable.
|
[AMI Test Configuration]  Typ_corner_time_domain
Type Time_domain
Direction Tx
```

```
Input_IR_file  four_tap_input_IR.txt
Input_waveform_file  four_tap_input_bits.txt
AMI_input_parameters_file  four_tap_tx_params_typ_TD.txt
Golden_IR_file  four_tap_output_IR_typ.txt
Golden_waveform_file  four_tap_output_wave_typ_TD.txt
AMI_output_parameters_file  four_tap_output_params_typ_TD.txt
Executable_index 3

| A separate [AMI Test Configuration] file could be provided with a
| different number of crosstalk nodes represented by different
| Input_IR_file and Golden_waveform_file entries, but with
| otherwise identical parameters and a different name.
```

10.12 MODEL SPECIFIC PARAMETERS

The following section describes the Model Specific (user-defined) Parameters. The model maker can specify any number of Model Specific Parameters for their model. The Model Specific Parameter branch shall begin with the reserved words “Model_Specific”.

Example:

```
(Model_Specific
  (CTLE
    (Description "CTLE consists of two selectable sets of Poles and Zeros")
    (Row (Range 0 0 1)(Type Integer)(Usage InOut)(Description "Two CTLEs"))
    (Poles (Usage In) (Description "CTLE Poles")
      (Type Integer Float Float Float Float Float Float)
      (Table
      (Labels "Row" "Real_1" "Imag_1" "Real_2" "Imag_2" "Real_3" "Imag_3")
               (0   -3.06e+9  9.94e+9 -2.91e+9  5.94e+9 -1.36e+9  0.0)
               (1   -1.03e+10 0.0     -4.21e+9  5.42e+9  0.0      0.0)
      )
    )
    (Zeros (Usage In) (Description "CTLE Zeros")
      (Type Integer Float Float Float Float)
      (Table
      (Labels "Row" "Real_1" "Imag_1" "Real_2" "Imag_2")
              (0    -3.62e+9  0.0     -2.33e+9  6.68e+9)
              (1    -2.93e+9  1.10e+9  0.0      0.0)
      )
    )
  )
)
```

10.12.1 TAPPED DELAY LINE EXAMPLE

A tapped delay line can be described by creating a separate parameter for each tap weight and grouping all the tap weights for a given tapped delay line in a single parameter group which is given the name of the tapped delay line. If, in addition, the individual tap weights are each given a name which is their tap number (i.e., “-1” is the name of the first precursor tap, “0” is the name of the main tap, “1” is the name of the first postcursor tap, etc.) and the tap weights are declared to be of type Tap, then the EDA tool can assume that the individual parameters are tap weights in a tapped delay line, and use that assumption to perform tasks such as optimization. The model developer is responsible for choosing whether or not to follow this convention.

The type Tap implies that the parameter takes on floating-point values. Note that if the type Tap is used and the parameter name is not a number, this is an error condition for which EDA tool behavior is not specified.

Example:

```
(mySampleAMI                          | AMI parameter definition root name
  (Description "Sample AMI File")
  (Reserved_Parameters                | Required heading
    (AMI_Version (Usage Info) (Type String) (Value "8.0")
```

```
      (Description "Valid for AMI_Version 5.1 and above"))
    (Ignore_Bits (Usage Info) (Type Integer) (Value 21)
      (Description "Ignore 21 Bits"))
    (Max_Init_Aggressors (Usage Info) (Type Integer) (Value 25))
    (Init_Returns_Impulse (Usage Info) (Type Boolean) (Value True))
    (GetWave_Exists (Usage Info) (Type Boolean) (Value True))
  )                                     | End Reserved_Parameters

  (Model_Specific                       | Required heading
    (txtaps
      (-2 (Usage InOut) (Type Tap) (Range 0.1 -0.1 0.2)
          (Description "Second Precursor Tap"))
      (-1 (Usage InOut) (Type Tap) (Range 0.2 -0.4 0.4)
          (Description "First Precursor Tap"))
      (0  (Usage InOut) (Type Tap) (Range 1 0.4 1)
          (Description "Main Tap"))
      (1  (Usage InOut) (Type Tap) (Range 0.2 -0.4 0.4)
          (Description "First Postcursor Tap"))
      (2  (Usage InOut) (Type Tap) (Range 0.1 -0.1 0.2)
          (Description "Second Postcursor Tap"))
    ) | End txtaps
  ) | End Model_Specific
) | End mySampleAMI
```

10.13 RESERVED PARAMETER AND DATA TYPE RULE SUMMARY TABLES

The tables below summarize the supporting versions and valid combinations of AMI Reserved Parameters, defaults, data Types and data Formats.

Table 41 – Reserved Parameters and Supported AMI_Versions

Reserved Parameter	First Supported AMI_Version
Adaptation_Valid	8.0
AMI_Version	Required in 5.1 and later
BCI_ID	7.0
BCI_Message_Interval_UI	7.0
BCI_Protocol	7.0
BCI_State	7.0
BCI_Training_Mode	7.1
BCI_Training_UI	7.0
Component_Name	7.1
DC_Offset	7.1
DLL_ID	6.0
DLL_Path	6.0
GetWave_Exists	Implicit 5.0
Ignore_Bits	Implicit 5.0
Init_Returns_Impulse	Implicit 5.0
Max_Init_Aggressors	Implicit 5.0
Model_Name	6.1
Modulation	6.1
Modulation_Levels	7.2
PAM_Offsets	7.2
PAM_Thresholds	7.2
PAM4_CenterEyeOffset	6.1
PAM4_CenterThreshold	6.1
PAM4_LowerEyeOffset	6.1
PAM4_LowerThreshold	6.1
PAM4_Mapping	6.1
PAM4_UpperEyeOffset	6.1
PAM4_UpperThreshold	6.1
Repeater_Type	6.0

Reserved Parameter	First Supported AMI_Version
Resolve_Exists	6.1
Rx_Clock_PDF	Implicit 5.0
Rx_Clock_Recovery_DCD	6.0
Rx_Clock_Recovery_Dj	6.0
Rx_Clock_Recovery_Mean	6.0
Rx_Clock_Recovery_Rj	6.0
Rx_Clock_Recovery_Sj	6.0
Rx_Decision_Time	7.1
Rx_DCD	6.0
Rx_Dj	6.0
Rx_GaussianNoise	7.0
Rx_Noise	6.0
Rx_Port_Order	8.0
Rx_R	7.0
Rx_Receiver_Sensitivity	Implicit 5.0
Rx_Rj	6.0
Rx_Sj	6.0
Rx_UniformNoise	7.0
Rx_Use_Clock_Input	7.1
Signal_Name	7.1
Special_Param_Names	7.0
Supporting_Files	6.0
Ts4file	7.0
Tx_DCD	Implicit 5.0
Tx_Dj	6.0
Tx_Impulse_Input	7.2
Tx_Jitter	Implicit 5.0
Tx_Port_Order	8.0
Tx_R	7.0
Tx_Rj	6.0
Tx_Sj	6.0
Tx_Sj_Frequency	6.0
Tx_V	7.0
Use_Init_Output	Implicit 5.0, illegal in 5.1 and later

Table 42 – General Rules and Allowable Usage for Reserved Parameters

Reserved Parameter	General Rules			Allowable Usage				
	Required	Default[2,7]	Place-holder[8]	Info	In	Out	Dep[1]	InOut
Adaptation_Valid	No	--				X		
AMI_Version	Yes[10]	--		X				
BCI_ID	No[3]	--		X				
BCI_Message_Interval_UI	No[4]	--			X			
BCI_Protocol	No	--			X			
BCI_State	No[3]	--						X
BCI_Training_Mode	No	"GetWave"			X			
BCI_Training_UI	No[4]	--			X			
Component_Name	No[9]	--	X		X			
DC_Offset	No	--	X		X			
DLL_ID	No	--	X		X			
DLL_Path	No	--	X		X			
GetWave_Exists	Yes	--		X				
Ignore_Bits	No	0		X				
Init_Returns_Impulse	Yes	--		X				
Max_Init_Aggressors	No	0		X				
Model_Name	No	--	X		X			
Modulation	No	"NRZ"		X	X			
Modulation_Levels	No	(see Usage Rules)			X			
PAM_Offsets	No	(see Usage Rules)				X		
PAM_Thresholds	No[12]	(see Usage Rules)				X		
PAM4_CenterEyeOffset	No	0		X		X	X	X
PAM4_CenterThreshold	No	0		X		X	X	X
PAM4_LowerEyeOffset	No	0		X		X	X	X
PAM4_LowerThreshold	No	(see Usage Rules)		X		X	X	X

Reserved Parameter	General Rules			Allowable Usage				
	Required	Default[2,7]	Place-holder[8]	Info	In	Out	Dep[1]	InOut
PAM4_Mapping	No	"0132"		X	X			
PAM4_UpperEyeOffset	No	0		X		X	X	X
PAM4_UpperThreshold	No	(see Usage Rules)		X		X	X	X
Repeater_Type	No[5]	--		X				
Resolve_Exists	No	False		X				
Rx_Clock_PDF	No	0		X		X	X	
Rx_Clock_Recovery_DCD	No	0		X		X	X	
Rx_Clock_Recovery_Dj	No	0		X		X	X	
Rx_Clock_Recovery_Mean	No	0		X		X	X	
Rx_Clock_Recovery_Rj	No	0		X		X	X	
Rx_Clock_Recovery_Sj	No	0		X		X	X	
Rx_DCD	No	0		X		X	X	
Rx_Decision_Time	No	--				X		
Rx_Dj	No	0		X		X	X	
Rx_GaussianNoise, Rx_Noise	No	0		X		X	X	
Rx_Port_Order	No	"13-24"		X				
Rx_R	No[11]	Infinity		X			X	
Rx_Receiver_Sensitivity	No	0		X		X	X	
Rx_Rj	No	0		X		X	X	
Rx_Sj	No	0		X		X	X	
Rx_UniformNoise	No	0		X		X	X	
Rx_Use_Clock_Input	No	"None"			X			
Signal_Name	No[9]	--	X		X			
Special_Param_Names	No	--		X				
Supporting_Files	No	--	X	X				
Ts4file	No	--		X			X	
Tx_DCD	No	0		X		X	X	
Tx_Dj	No	0		X		X	X	

Reserved Parameter	General Rules			Allowable Usage				
	Required	Default[2,7]	Place-holder[8]	Info	In	Out	Dep[1]	InOut
Tx_Impulse_Input	No	"Downstream"		X				
Tx_Jitter	No	0		X		X	X	
Tx_Port_Order	No	"13-24"		X				
Tx_R	No[11]	0		X			X	
Tx_Rj	No	0		X		X	X	
Tx_Sj	No	0		X		X	X	
Tx_Sj_Frequency	No	(see Usage Rules)		X		X	X	
Tx_V	No[6,11]	--		X			X	
Use_Init_Output	No	True		X				

Notes:

1) Illegal for AMI_Version 6.0 and earlier.
2) "Default" in this context means "behavior if Reserved Parameter is absent".
3) Required if BCI_Protocol is present.
4) Required for Rx models if BCI_Protocol is present.
5) Required if [Repeater Pin] is present.
6) Required if Ts4file is present for a Tx model.
7) "--" means that an entry must be provided if the parameter is present; no default is assumed or permitted.
8) In this context, "Placeholder X" means that the EDA tool will supply or calculate an entry for the parameter string to replace the entry found in the .ami file.
9) Either both Component_Name and Signal_Name, or neither, shall be present.
10) Required for AMI_Version 5.1 and later; illegal earlier.
11) Illegal if Ts4file is not present.
12) Required if Modulation_Levels is present.

Table 43 – Allowable Data Types for Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Adaptation_Valid					X
AMI_Version				X	
BCI_Message_Interval_UI			X		
BCI_ID				X	
BCI_Protocol				X	
BCI_State				X	

Reserved Parameter	Data Type				
	Float	**UI**	**Integer**	**String**	**Boolean**
BCI_Training_Mode				X	
BCI_Training_UI			X		
Component_Name				X	
DC_Offset	X				
DLL_ID				X	
DLL_Path				X	
GetWave_Exists					X
Ignore_Bits			X		
Init_Returns_Impulse					X
Max_Init_Aggressors			X		
Model_Name				X	
Modulation				X	
Modulation_Levels			X		
PAM_Offsets				X	
PAM_Thresholds				X	
PAM4_CenterEyeOffset	X	X			
PAM4_CenterThreshold	X				
PAM4_LowerEyeOffset	X	X			
PAM4_LowerThreshold	X				
PAM4_Mapping				X	
PAM4_UpperEyeOffset	X	X			
PAM4_UpperThreshold	X				
Repeater_Type				X	
Resolve_Exists					X
Rx_Clock_PDF	X	X			
Rx_Clock_Recovery_DCD	X	X			
Rx_Clock_Recovery_Dj	X	X			
Rx_Clock_Recovery_Mean	X	X			
Rx_Clock_Recovery_Rj	X	X			
Rx_Clock_Recovery_Sj	X	X			
Rx_DCD	X	X			
Rx_Decision_Time	X	X			
Rx_Dj	X	X			

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Rx_GaussianNoise, Rx_Noise	X				
Rx_Port_Order				X	
Rx_R	X				
Rx_Receiver_Sensitivity	X				
Rx_Rj	X	X			
Rx_Sj	X	X			
Rx_UniformNoise	X				
Rx_Use_Clock_Input				X	
Signal_Name				X	
Special_Param_Names				X	
Supporting_Files				X	
Ts4file				X	
Tx_DCD	X	X			
Tx_Dj	X	X			
Tx_Impulse_Input				X	
Tx_Jitter	X	X			
Tx_Port_Order				X	
Tx_R	X				
Tx_Rj	X	X			
Tx_Sj	X	X			
Tx_Sj_Frequency	X				
Tx_V	X				
Use_Init_Output					X

Table 44 – Allowable Data Formats for Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
Adaptation_Valid	X									
AMI_Version	X									
BCI_ID	X									

Reserved Parameter	Data Format									
	Value	**Range**	**Corner**	**List**	**Increment**	**Steps**	**Gaussian**	**Dual-Dirac**	**DjRj**	**Table**
BCI_Message_Interval_UI	X									
BCI_Protocol	X			X						
BCI_State				X						
BCI_Training_Mode	X			X						
BCI_Training_UI	X									
Component_Name	X									
DC_Offset	X									
DLL_ID	X									
DLL_Path	X									
GetWave_Exists	X									
Ignore_Bits	X									
Init_Returns_Impulse	X									
Max_Init_Aggressors	X									
Model_Name	X									
Modulation	X			X						
Modulation_Levels	X			X						
PAM_Offsets	X									
PAM_Thresholds	X									
PAM4_CenterEyeOffset	X									
PAM4_CenterThreshold	X									
PAM4_LowerEyeOffset	X									
PAM4_LowerThreshold	X									
PAM4_Mapping	X			X						
PAM4_UpperEyeOffset	X									
PAM4_UpperThreshold	X									
Repeater_Type	X									
Resolve_Exists	X									
Rx_Clock_PDF							X	X	X	X
Rx_Clock_Recovery_DCD	X	X	X	X	X	X				
Rx_Clock_Recovery_Dj	X	X	X	X	X	X				

Reserved Parameter	Data Format									
	Value	**Range**	**Corner**	**List**	**Increment**	**Steps**	**Gaussian**	**Dual-Dirac**	**DjRj**	**Table**
Rx_Clock_Recovery_Mean	X	X	X	X	X	X				
Rx_Clock_Recovery_Rj	X	X	X	X	X	X				
Rx_Clock_Recovery_Sj	X	X	X	X	X	X				
Rx_DCD	X	X	X	X	X	X				
Rx_Decision_Time	X									
Rx_Dj	X	X	X	X	X	X				
Rx_GaussianNoise, Rx_Noise	X	X	X	X	X	X				
Rx_Port_Order	X									
Rx_R	X	X	X	X	X	X				
Rx_Receiver_Sensitivity	X	X	X	X	X	X				
Rx_Rj	X	X	X	X	X	X				
Rx_Sj	X	X	X	X	X	X				
Rx_UniformNoise	X	X	X	X	X	X				
Rx_Use_Clock_Input	X			X						
Signal_Name	X									
Special_Param_Names										X
Supporting_Files										X
Ts4file	X		X	X						
Tx_DCD	X	X	X	X	X	X				
Tx_Dj	X	X	X	X	X	X				
Tx_Impulse_Input	X									
Tx_Jitter							X	X	X	X
Tx_Port_Order	X									
Tx_R	X	X	X	X	X	X				
Tx_Rj	X	X	X	X	X	X				
Tx_Sj	X	X	X	X	X	X				
Tx_Sj_Frequency	X	X	X	X	X	X				
Tx_V	X	X	X	X	X	X				
Use_Init_Output	X									

Table 45 summarizes the relationships between the different Format and Data Types for Reserved or Model Specific Parameters.

Table 45 – Allowable Data Types for Format Values

Format	Data Type					
	Float	**UI**	**Integer**	**String**	**Boolean**	**Tap**
Corner	X	X	X	X	X	X
DjRj	X	X				
Dual-Dirac	X	X				
Gaussian	X	X				
Increment	X	X	X			X
List	X	X	X	X	X	X
Range	X	X	X			X
Steps	X	X	X			X
Table	X	X	X	X	X	
Value	X	X	X	X	X	X

AMI parameter definition file Reserved Parameters and [Model] Model_type subparameter declarations shall be mutually consistent. Additionally, both Reserved Parameters and Model_type subparameter arguments shall be consistent with the associated [Algorithmic Model] Executable_Tx and Executable_Rx subparameters, if present (i.e., for I/O-capable buffers that can handle both Tx and Rx functions).

To maintain consistency with the directionality of the associated buffer, only certain Reserved Parameters may be combined in the same .ami file. Tx-only and Rx-only Reserved Parameters shall not be present in the same .ami file. Further, Tx-only Reserved Parameters shall not be present in .ami files associated with [Algorithmic Model] Executable_Rx subparameters. Similarly, Rx-only Reserved Parameters shall not be present in .ami files associated with [Algorithmic Model] Executable_Tx subparameters.

The directions supported for each Reserved Parameter are shown in Table 46 below. The Model_type and permitted associated subparameter arguments for the [Algorithmic Model] keyword are shown in Table 47.

Table 46 – Defined Directions for Reserved Parameters

Reserved Parameter	Supported Direction(s)
Adaptation_Valid	Rx
AMI_Version	Rx, Tx
BCI_ID	Rx, Tx

Reserved Parameter	Supported Direction(s)
BCI_Message_Interval_UI	Rx
BCI_Protocol	Rx, Tx
BCI_State	Rx, Tx
BCI_Training_Mode	Rx, Tx
BCI_Training_UI	Rx
Component_Name	Rx, Tx
DC_Offset	Rx
DLL_ID	Rx, Tx
DLL_Path	Rx, Tx
GetWave_Exists	Rx, Tx
Ignore_Bits	Rx, Tx
Init_Returns_Impulse	Rx, Tx
Max_Init_Aggressors	Rx, Tx
Model_Name	Rx, Tx
Modulation	Rx, Tx
Modulation_Levels	Rx, Tx
PAM_Offsets	Rx
PAM_Thresholds	Rx
PAM4_CenterEyeOffset	Rx
PAM4_CenterThreshold	Rx
PAM4_LowerEyeOffset	Rx
PAM4_LowerThreshold	Rx
PAM4_Mapping	Rx, Tx
PAM4_UpperEyeOffset	Rx
PAM4_UpperThreshold	Rx

Reserved Parameter	Supported Direction(s)
Repeater_Type	Rx
Resolve_Exists	Rx, Tx
Rx_Clock_PDF	Rx
Rx_Clock_Recovery_DCD	Rx
Rx_Clock_Recovery_Dj	Rx
Rx_Clock_Recovery_Mean	Rx
Rx_Clock_Recovery_Rj	Rx
Rx_Clock_Recovery_Sj	Rx
Rx_DCD	Rx
Rx_Decision_Time	Rx
Rx_Dj	Rx
Rx_GaussianNoise, Rx_Noise	Rx
Rx_Port_Order	Rx
Rx_R	Rx
Rx_Receiver_Sensitivity	Rx
Rx_Rj	Rx
Rx_Sj	Rx
Rx_UniformNoise	Rx
Rx_Use_Clock_Input	Rx
Signal_Name	Rx, Tx
Special_Param_Names	Rx, Tx
Supporting Files	Rx, Tx
Ts4file	Rx, Tx
Tx_DCD	Tx
Tx_Dj	Tx

Reserved Parameter	Supported Direction(s)
Tx_Impulse_Input	Tx
Tx_Jitter	Tx
Tx_Port_Order	Tx
Tx_R	Tx
Tx_Rj	Tx
Tx_Sj	Tx
Tx_Sj_Frequency	Tx
Tx_V	Tx
Use_Init_Output	N/A (illegal combination)

Table 47 – [Algorithmic Model] Subparameter and [Model] Model_type Interaction

[Model] Model Type	[Algorithmic Model] Executable Subparameters Permitted
3-state 3-state_ECL	Executable only Executable_Rx and Executable_Tx are not permitted
3-state_diff	Executable only Executable_Rx and Executable_Tx are not permitted
Input Input_ECL	Executable only Executable_Rx and Executable_Tx are not permitted
Input_diff	Executable only Executable_Rx and Executable_Tx are not permitted
I/O I/O_open_drain I/O_open_sink I/O_open_source I/O_ECL	Executable illegal Executable_Rx and/or Executable_Tx are required

[Model] Model Type	[Algorithmic Model] Executable Subparameters Permitted
I/O_diff	Executable illegal Executable_Rx and/or Executable_Tx are required
Open_sink Open_drain Open_source	Executable only Executable_Rx and Executable_Tx are not permitted
Output Output_ECL	Executable only Executable_Rx and Executable_Tx are not permitted
Output_diff	Executable only Executable_Rx and Executable_Tx are not permitted
Series	N/A (illegal)
Series_switch	N/A (illegal)
Terminator	N/A (illegal)

11 INTERCONNECT MODELING

11.1 INTRODUCTION

IBIS supports broadband interconnect models describing connections between the pins of a component and its I/O buffers. These interconnect models may include descriptions of frequency-dependent losses, interconnect coupling and/or complex supply rail distributions.

Interconnect is defined between up to three interface locations:

- pin, where a component connects to a printed circuit board
- die pad, where a component die connects to the routing on a package substrate
- buffer, where the buffer itself connects to the die substrate and routing

The relationship between the terminals at the buffer, die pad, and pin interfaces is shown in Figure 48 below.

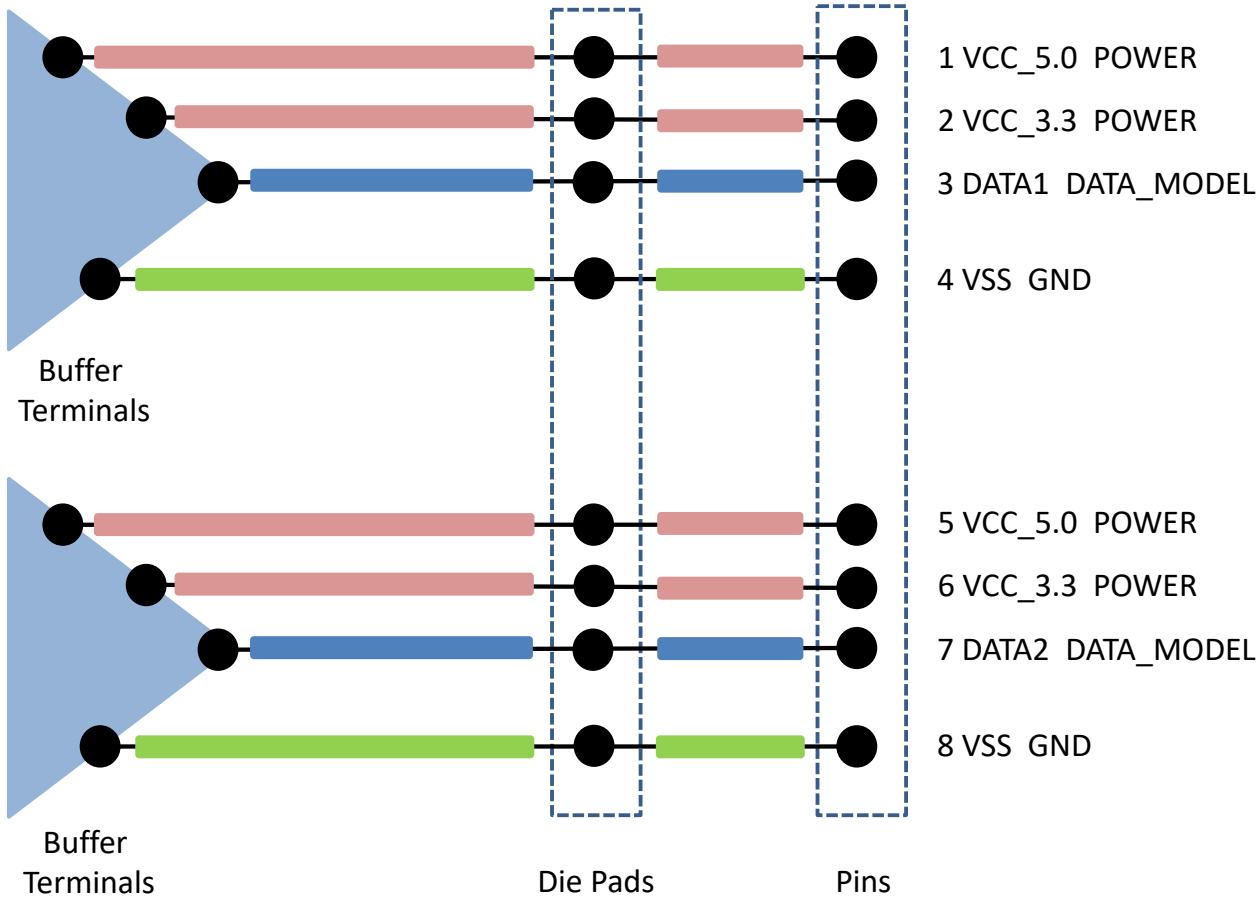


Figure 48 – Example Interconnect Model Structure

The connection between the pin and die pad is generally called “package interconnect”, while the connection between the die pad and the buffer is generally called “on-die interconnect.” The die

pad is distinct from the buffer terminal; the buffer includes the circuitry that would be described through the [Model] keyword and related keywords, and would not include transmission line behavior.

Interconnect Models may be supplied separately for on-die interconnect and package interconnect, or may be supplied as a single model for the entire connection between the package pins and buffers.

The electrical behavior of an interconnect is described through either IBIS-ISS subcircuits or Touchstone network parameters. An Interconnect Model defines the connections to either an IBIS-ISS subcircuit or a Touchstone file. An Interconnect Model may describe the connection between the I/O pins of the package and the buffers, the pins of the package and the die pads, or the die pads and buffers. Rail (supply) terminals related to GND and POWER pins can be described in a similar manner, but can also exist on only one interface for serving as reference terminals or for supporting, for example, decoupling circuitry.

Note: The syntax defined in this section allows the definition of connections, particularly related to Interconnect Model referencing, that might be ambiguous or non-physical (see example 11). The interpretation of Interconnect Model connectivity may depend on specific EDA tool implementations. Interconnect Models are organized into Interconnect Model Sets. An [Interconnect Model Set] keyword consists of one or more [Interconnect Model] keywords. One Interconnect Model Set may contain groups of similar Interconnect Models or different Interconnect Models to describe the complete connections from the buffer to pin interface.

Each I/O pin is associated with one I/O buffer terminal and optionally one I/O die pad. By contrast, there is no required one-to-one relationship between rail pins, (optional) rail die pads, and buffer rail terminals.

Figure 49 below shows the [Interconnect Model] terminals for an I/O path on both package and on-die substrates.

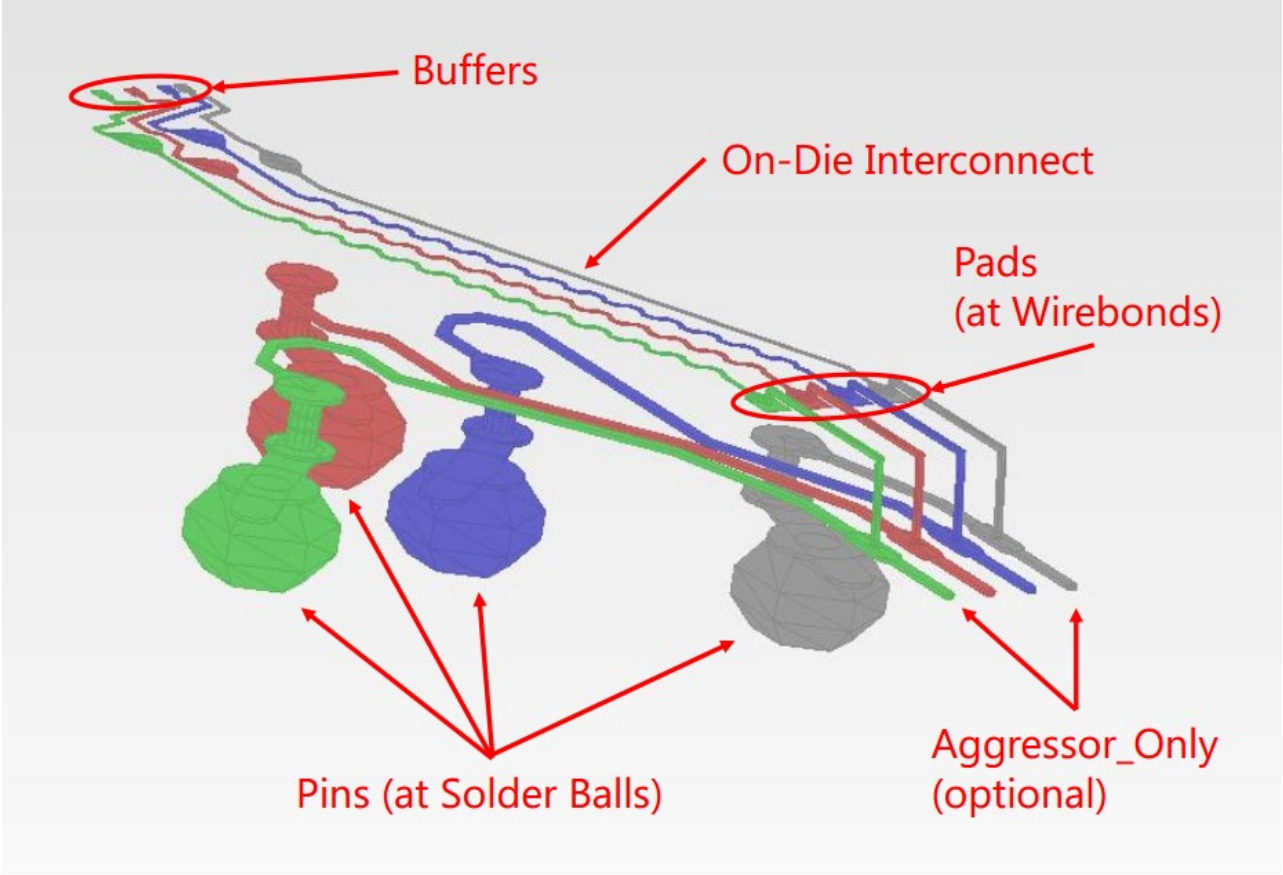


Figure 49 – Package and On-die Substrate I/O Paths

The figure also shows on-die interconnect routes that may experience crosstalk effects. This example assumes that only a few routes out of a larger bus are shown. In such a model, the crosstalk on any one route *in the model* could only be caused by its nearest neighbors.

While each of the inner two routes in the figure may have all potential crosstalk represented in the model, the outer signals would not. The model maker would therefore indicate that connections to the outer routes' terminals do not include all of their aggressors by adding the optional argument "Aggressor_Only" to their terminals. The descriptions of the associated terminals would not use the "Aggressor_Only" designation for the inner routes. The EDA tool may therefore assume in simulation that the inner routes have all (or more practically most of) the coupling to their aggressor connections represented in the model.

Crosstalk simulations require Interconnect Models to have connections to multiple I/O pin_names.

Figure 50, showing a package and die, illustrates graphically the potential [Interconnect Model] terminals for a rail connection. A single terminal for a supply rail connects to multiple die pads. Note that these terminals may be collapsed to a single terminal at the pin (as shown), or alternately at the die pad interface and/or the buffer rail terminals.

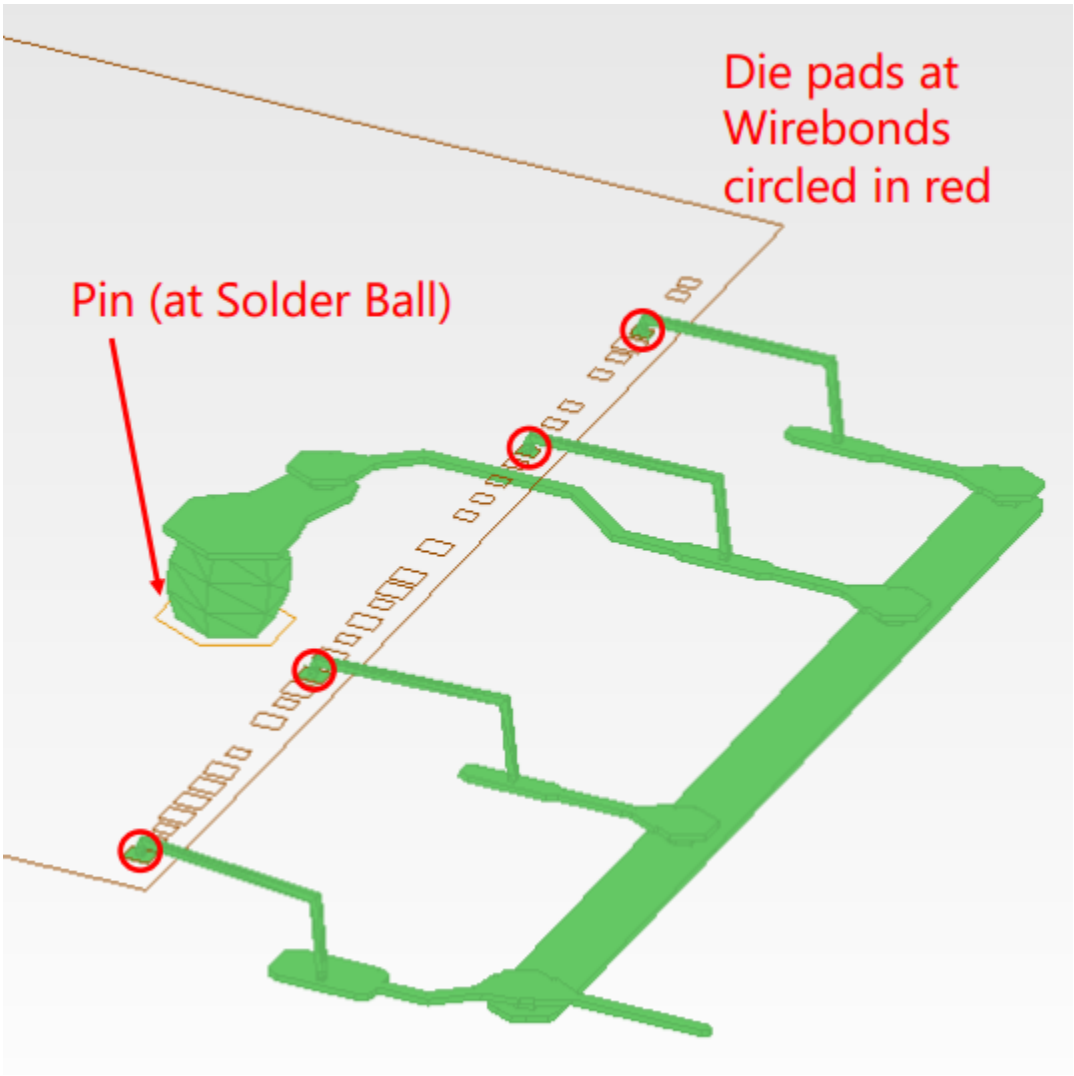


Figure 50 – Package Substrate Rail Terminals

The terminal section of an [Interconnect Model] describes how the terminals of an Interconnect Model subcircuit or Touchstone file instance are connected at a buffer terminal, die pad interface or pin/board interface.

11.2 GENERAL INTERCONNECT SYNTAX REQUIREMENTS

Terminal lines under the [Interconnect Model] keyword describe connections.

I/O terminals shall be connected using only the pin_name qualifier at these locations:

- pins: I/O pin_name
- die pads: I/O pin_name
- buffer: I/O pin_name

Rail terminal connections have more options to support direct connections to terminals or to groups of terminals using signal_name, bus_label, or pad_name entries at the pin, die pad or buffer locations. For the following locations the rail terminal can connect to:

- pins
  - a specific rail pin_name
  - all of the pins of a rail signal_name
  - all of the pins of a bus_label

- die pads
  - all of the die pads with a rail signal_name
  - all of the die pads with a bus_label
  - a specific die pad pad_name

- buffer rail terminals
  - all of the buffer rail terminals of a rail signal_name
  - all of the buffer rail terminals of a bus_label
  - a specific buffer rail terminal for an I/O buffer pin_name

One or more Interconnect Model Sets may be included in a separate Interconnect Model Set file, using a file name with the extension “ims”, or within the .ibs file where [Interconnect Model Group] is used. The [Interconnect Model Set] keyword can contain the optional [Manufacturer] and [Description] keywords and one or more [Interconnect Model] keywords and the [Interconnect Model] associated subparameters, as is listed in Table 48.

Table 48 – Interconnect Modeling Keywords and Subparameters

Keyword or Subparameter	Notes
[Interconnect Model Set]	
[Manufacturer]	(note 1)
[Description]	(note 1)
[Interconnect Model]	(note 2)

Keyword or Subparameter	Notes
Param	
File_TS	(note 3)
File_IBIS-ISS	(note 3)
Unused_port_termination	(note 4)
Number_of_terminals	(note 5)
<terminal line>	(note 6)
[End Interconnect Model]	(note 7)
[End Interconnect Model Set]	(note 8)

Notes:

1) [Manufacturer] and [Description] are each optional keywords within any [Interconnect Model Set].
2) At least one [Interconnect Model] is required for each [Interconnect Model Set].
3) One of either the File_TS or File_IBIS-ISS subparameters is required.
4) Required for Touchstone files where ports are unused, illegal if there are no unused ports or for IBIS-ISS file.
5) This subparameter shall be followed by the "=" character and an integer value, with both optionally surrounded by whitespace.
6) See text below.
7) Required when the [Interconnect Model] keyword is used.
8) Required when the [Interconnect Model Set] keyword is used.

When Interconnect Model Set definitions occur within a .ibs file, their scope is "local"— they are known only within that .ibs file and no other .ibs file. In addition, within that .ibs file, they override any interconnect package models defined for the same pins using the [Package], [Pin], or [Define Package Model] keywords. Interconnect Models in separate .ims files referenced by the [Interconnect Model Group] keyword in a .ibs file also override any interconnect package models defined for the same pins in the same .ibs file using the [Package], [Pin], or [Define Package Model] keywords.

Usage Rules for the .ims file:

Interconnect Models are stored in a file whose file name uses the format:

<stem>.ims

The <stem> provided shall adhere to the rules given for the [File Name] keyword. Use the "ims" extension to identify files containing Interconnect Models. The .ims file shall contain the [IBIS Ver], [File Name], [File Rev], and the [End] keywords. Optional elements include the [Date], [Source], [Notes], [Disclaimer], [Copyright], and [Comment Char] keywords. All of these keywords and associated subparameters follow the same rules as those for a normal .ibs file.

Note that the [Component] and [Model] keywords are not allowed in the .ims file. The .ims file is for Interconnect Models only.

Keyword: **[Interconnect Model Set]**

Required: No

Description: Used to contain Interconnect Models.

Usage Rules: [Interconnect Model Set] has a single argument, which is the name of the Interconnect Model Set. The length of the Interconnect Model Set name shall not exceed 40 characters. Blank characters are not allowed. The [Interconnect Model Set]/[End Interconnect Model Set] keyword pair is hierarchically equivalent in scope to [Component] and [Model].

The section under the [Interconnect Model Set] keyword may contain a [Manufacturer] keyword section and [Description] keyword section and shall contain one or more Interconnect Models. See the [Interconnect Model] section for a description of the content of each Interconnect Model.

An [Interconnect Model Set] contains a list of [Interconnect Model]s that have a logical association such as:

- All models in a bus (e.g., DDR4 or PCIe 3.0)
- Full PDN structure from buffer to pin
- On-die PDN structure from buffers to die pads
- Package-only PDN structure from die pads to pins
- All I/O models between die pad and pin
- All I/O models between buffer and die pad
- All I/O models between buffer and pin
- Combined I/O and PDN models
- All uncoupled models
- Coupled models
- Touchstone electrical models
- Decoupling capacitor models
- IBIS-ISS electrical models

Example:

```
[Interconnect Model Set] Signal_Integrity
[Manufacturer] NoName Corp.
[Description] This set contains one model for each I/O buffer
[Interconnect Model] DQ1
…
[End Interconnect Model]
[Interconnect Model] DQ2
…
[End Interconnect Model]
[Interconnect Model] DQS
…
[End Interconnect Model]
[End Interconnect Model Set]
```

Keyword: **[Manufacturer]**

Required: No

Description: Specifies the name of the [Interconnect Model Set] manufacturer.

Usage Rules: The length of the manufacturer's name shall not exceed 40 characters (blank characters are allowed, e.g., Oklahoma Instruments).

Example:

```
[Manufacturer]  NoName Corp.
```

Keyword: **[Description]**

Required: No

Description: Provides a concise yet easily human-readable description of what kind of interconnect the [Interconnect Model Set] represents.

Usage Rules: The description shall fit on a single line and may contain spaces.

Example:

```
[Description]   220-Pin Quad Ceramic Flat Pack
```

Keyword: [**End Interconnect Model Set**]

Required: Yes, for each instance of the [Interconnect Model Set] keyword.

Description: Indicates the end of the Interconnect Model Set data.

Example:

```
[End Interconnect Model Set]
```

Keyword: **[Interconnect Model]**

Required: No

Description: Marks the beginning of an Interconnect Model description that is used to define the interfaces to IBIS-ISS subcircuits or n-port networks described by Touchstone files.

Sub-Params: Param, File_TS, File_IBIS-ISS, Unused_port_termination, Number_of_terminals

Usage Rules: [Interconnect Model] has a single argument, which is the name of the associated Interconnect Model. The length of the Interconnect Model name shall not exceed 40 characters. Blank characters are not allowed. The [Interconnect Model]/[End Interconnect Model] keyword pair is hierarchically scoped by the [Interconnect Model Set]/[End Interconnect Model Set] keywords.

The [Interconnect Model]/[End Interconnect Model] section defines both the association between a Touchstone file or IBIS-ISS subcircuit and an Interconnect Model, as well as defining the terminals and terminal usage for the Interconnect Model in the context of the given [Component].

An [Interconnect Model] shall contain one and only one of the following combinations:

- pins and buffer terminals (full package model)
- pins and die pads (package-only model)
- die pads and buffer terminals (on-die interconnect model)
- rail terminals at only one interface and no I/O terminals

Other Notes: If a full package model contains an I/O terminal for a pin_name then it shall also contain an I/O buffer terminal for the same pin_name. If a package-only model contains an I/O terminal for a pin_name, then it shall also contain an I/O die pad for the same pin_name. If an on-die interconnect model contains an I/O buffer terminal for a pin_name, then it shall also contain an I/O die pad for the same pin_name.

An [Interconnect Model] may contain:

- only power rail models
- one or more I/O signal models
- both power rail models and one or more I/O signal models
- pin rails only
- die pad rails only
- buffer rails only

Each terminal of an Interconnect Model is connected to a node and has a "voltage". This, as stated, is imprecise. Voltage, by definition, is a potential difference between two points. It is common to probe and plot the potential difference between simulator nodes at a terminal and a simulator global reference node (e.g., SPICE ideal node "0"), the latter of which is often assumed and/or unstated. This is valid for non-power-aware simulations when the local reference (or return path) node is forced to a global reference by the simulator, or for "ground-referenced" power aware simulations that lump the effects of all rail interconnects together. However, this is not valid when the local reference nodes are "floating". In this case it is important that the actual reference node for measurements at the I/O buffer is included as a terminal in the Interconnect Model. If this is not done, then the Interconnect Model will not correctly account for all return currents, particularly from capacitive elements. If an Interconnect Model does not contain a reference terminal, then the user of these models should be aware that using these models in power-aware simulations can potentially introduce errors in simulations.

The following subparameters are defined:

Param

File_IBIS-ISS

File_TS

Unused_port_termination

Number_of_terminals = <value>

In addition to these subparameters, the [Interconnect Model]/[End Interconnect Model] section may contain lines describing terminals and their connections. No specific subparameter name or other string is used to identify terminal lines.

Unless noted below, no Interconnect Model subparameter requires the presence of any other subparameter.

Param rules:

The subparameter Param is optional and only legal with the File_IBIS-ISS subparameter documented below. Param is illegal with the File_TS subparameter documented below. Param shall be followed by three arguments: an unquoted string argument giving the name of the parameter to be passed into the IBIS-ISS subcircuit, a reserved word for the parameter format, and one numerical value or one string value (surrounded by double quotes) for the parameter value to be passed into the IBIS-ISS subcircuit. More than one Param line is permitted. The only defined entry for the format column is Value.

The numerical value rules follow the scaling conventions in Section 3.2, "SYNTAX RULES". The EDA tool is responsible for translating IBIS specified parameters into IBIS-ISS parameters. For example, 1 megaohm, would be represented as 1M in Param value according to the Section 3 rules, but would be converted by the EDA tool to case-insensitive 1meg (1X is not recommended) or 1E6 for IBIS-ISS use. Quoted string parameters in IBIS are converted to the string parameter syntax in IBIS-ISS subcircuits. For example, the Param value "typ.s2p" would be converted to str('typ.s2p') in IBIS-ISS subcircuits.

Examples:

```
| Param   name     format   value
Param     abc      Value    2m         | 2E-3 in IBIS
Param     def      Value    4k         | 4E3 in IBIS
Param     ts_file  Value    "typ.s2p"  | file name string passed
                                       | into IBIS-ISS
```

File_IBIS-ISS rules:

Either File_IBIS-ISS or File_TS is required for a [Interconnect Model]/[End Interconnect Model] group. The File_IBIS-ISS subparameter is followed by two unquoted string arguments consisting of the file_reference and circuit_name (.subckt name) for an IBIS-ISS file. The IBIS-ISS file under file_reference shall be located in the same directory as the referencing .ibs file or .ims file or in a specified directory under the referencing file as determined by the directory path (i.e., a file reference containing a relative path to a directory below that of the referencing .ibs or .ims file is permitted).

Example:

```
| file_type    file_reference    circuit_name(.subckt name)
File_IBIS-ISS  net.iss           netlist_typ
```

File_TS rules:

Either File_TS or File_IBIS-ISS is required for a [Interconnect Model]/[End Interconnect Model] group. File_TS is followed by one unquoted string argument, which is the file_reference for a Touchstone file. The Touchstone file under file_reference shall be

located in the same directory as the referencing .ibs file or .ims file or in a specified directory under the referencing file as determined by the directory path (i.e., a file reference containing a relative path to a directory below that of the referencing .ibs or .ims file is permitted).

Example:

```
| file_type    file_reference
File_TS        typ.s8p
```

Unused_port_termination rules:

The Unused_port_termination subparameter is required under this condition:

File_TS is used and the number of terminal lines (described below) is less than N+1 (where N is the number of ports in the Touchstone file)

Unused_port_termination is illegal under these conditions:

File_IBIS-ISS is used.

File_TS is used and the number of terminal lines is N+1

If required, only one Unused_port_termination subparameter may appear for a given [Interconnect Model] keyword.

The Unused_port_termination subparameter is followed by whitespace and one of these arguments:

Open

Reference

Resistance

“Open” declares that the unused ports remain unterminated (open-circuited).

“Reference” declares that the EDA tool terminates all unused ports with resistors whose resistance values are equal to the reference impedances provided in the Touchstone file for the respective unused ports, and all connected to the model’s reference terminal.

“Resistance” declares that the EDA tool terminates all unused ports with resistors, all having the same value, and all connected to the model’s reference terminal. The “Resistance” entry is followed by a third column entry with the (non-negative) numerical resistance value.

Examples:

```
Unused_port_termination       Open

Unused_port_termination       Reference

Unused_port_termination       Resistance  43.5
```

Number_of_terminals rules:

The Number_of_terminals subparameter is required and defines the number of terminals associated with the Interconnect Model. The subparameter name shall be followed by a single integer argument on the same line. The argument shall be separated from the subparameter name by the "=" character. The subparameter name, "=" character, and argument may optionally be separated by whitespace.

Only one Number_of_terminals subparameter may appear for a given [Interconnect Model] keyword. The Number_of_terminals subparameter shall appear before any terminal lines and after all other subparameters for a given Interconnect Model.

For File_IBIS-ISS, the Number_of_terminals value shall be equal to the number of subcircuit terminals for an IBIS-ISS subcircuit. Because an IBIS-ISS subcircuit requires at least one terminal the Number_of_terminals value shall be 1 or greater. The IBIS-ISS subcircuit terminals shall not contain an ideal reference node (SPICE node 0 or its synonyms).

For File_TS, the Number_of_terminals value shall be a value equal to N+1 (where N is the number of ports in the Touchstone file). Because a Touchstone file requires at least one port, the Number_of_terminals value shall be 2 or greater.

Example:

```
Number_of_terminals = 3
```

Terminal line rules:

The terminal lines shall appear after the Number_of_terminals subparameter and before the [End Interconnect Model] keyword.

Terminal lines are of the following form, with each identifier separated by whitespace:

<Terminal_number> <Terminal_type> <Terminal_type_qualifier> <Qualifier_entry> [Aggressor_Only]

Terminal_number

The Terminal_number is the identifier for a specific terminal. The value shall be 1 or greater and less than or equal to the Number_of_terminals. The same Terminal_number shall not appear more than once for a given Interconnect Model.

For File_IBIS-ISS, the Terminal_number entry shall match the IBIS-ISS terminal (node) position. The Terminal_number entries may be listed in any order as long as there are no duplicate entries. Each IBIS-ISS terminal shall have a terminal line entry.

For File_TS, the Terminal_number entry shall match the Touchstone file port number or reference terminal line, as shown below. The Terminal_number entries may be listed in any order as long as there are no duplicate entries. The terminal line for Terminal_number N+1 is required as a reference terminal for each port and shall be connected to a rail terminal or A_gnd in the Interconnect Model. At least one other terminal line entry is required.

- Terminal_number Port

- 1 1
- 2 2
- …
- N N
- N+1 Reference terminal for the Touchstone file

For n-port networks described by Touchstone files, each unused port and its corresponding Terminal_number shall be terminated in simulation with a resistor whose value corresponds to the Unused_port_termination subparameter entry. The resistor is connected to the model's reference terminal.

Terminal_type
The Terminal_type is a string that identifies whether the terminal is a reference, supply or I/O terminal and whether the terminal is connected at the buffer, die pad, or pin level. (Note that "I/O" in this context is a synonym for "signal", as opposed to "supply" or "rail"; it is not intended to imply model type as used in the "Model_type" subparameter). Furthermore, if the terminal is connected to a buffer supply rail, the Terminal_type identifies to which specific buffer rail the terminal is connected. The Terminal_type shall be one of the following:

- Pin_I/O
- Pad_I/O
- Buffer_I/O
- Pin_Rail
- Pad_Rail
- Buffer_Rail
- Pullup_ref
- Pulldown_ref
- Power_clamp_ref
- Gnd_clamp_ref
- Ext_ref
- A_gnd

Buffer_I/O, Pullup_ref, Pulldown_ref, Power_clamp_ref, Gnd_clamp_ref, Ext_ref and Buffer_Rail are terminals of an Interconnect Model that connect directly to I/O buffers.

Pad_I/O and Pad_Rail are terminals that are at the die pad interface.

Pin_I/O and Pin_Rail are terminals, at the pin interface, that can connect the package to the PCB.

The Terminal_types Buffer_I/O, Pad_I/O and Pin_I/O are used only for any single terminal of a buffer described by the [Model] keyword and for any Model_type subparameter listed in Section 6, Table 1. The Model_types Series and *_diff are used for two-terminal configurations, and their terminals require two separate Buffer_I/O, Pad_I/O or Pin_I/O Terminal_type lines.

Terminal_type A_gnd defines a connection to the simulator global reference node. The A_gnd node can be used at any interface.

Terminal_type A_gnd is not required under File_TS or File_IBIS-ISS.

If present under File_TS, Terminal_type A_gnd may be used only once on the terminal line with Terminal_number N+1.

If present under File_IBIS-ISS, Terminal_type A_gnd may be used on any number of terminal lines.

Terminal_type_qualifier
The Terminal_type_qualifier is a string that identifies the association between a terminal and a specific pin_name, signal_name, bus_label, or pad_name. Only certain Terminal_types may be used with pad_names, pin_names, signal_names, or bus_labels respectively, as outlined in the Connecting Pins, Pads, and Buffer Terminals section below and summarized in Figure 51.

Qualifier_entry
The <Qualifier_entry>, shown in angle brackets, is the name required for the following Terminal_type_qualifiers:

pin_name <pin_name_entry>

signal_name <signal_name_entry>

bus_label <bus_label_entry>

pad_name <pad_name_entry>

Aggressor_Only

The Aggressor_Only entry is optional and is indicated by the string “Aggressor_Only” without the quotation marks.

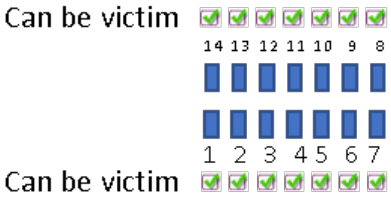
Multi-line Interconnect Models may describe only a subset of a coupled structure (e.g., a 64-line bus may be described by a four-line Interconnect Model). As a result, while the interconnects at the edges of the Interconnect Model may induce crosstalk onto other interconnects nearby, being on the edge of the Interconnect Model, they may not themselves experience the full crosstalk impact that the corresponding interconnect experiences in the real, full structure.

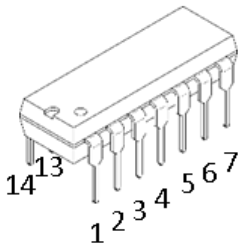
Figure 51 shows examples of Interconnect Models having full coupling for some pins and partial coupling for other pins of an example part package, and the corresponding Aggressor_Only entries. The Aggressor_Only column entry is allowed on all terminal locations for I/O terminals to indicate such incomplete coupling. Terminals that include the Aggressor_Only entry may not be suitable to

be simulated as victims, as they do not experience the full coupling present in the real physical structure. If an I/O terminal is not identified as Aggressor_Only, then the interconnect to that I/O terminal includes coupling to all interconnections deemed necessary for coupled signal analysis. Within any Interconnect Model, if a terminal line is identified as Aggressor_Only, then the corresponding terminal line associated with the same pin_name shall also be identified as Aggressor_Only.

Note that incomplete coupling implied by Aggressor_Only will render an entire conductive path associated with the Aggressor_Only terminal as possibly unsuitable for use as a victim in crosstalk analysis. For example, as shown in Figure 49, the pad-to-pin Interconnect Model terminal for the green path is Aggressor_Only. However, even though the connected buffer-to-pad Interconnect Model terminal for the green path is not marked Aggressor_Only, that connection path should still be regarded as having incomplete coupling.

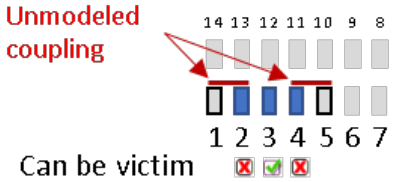
Full chip model supports both victim and aggressor modeling for all pins.
All crosstalk in the package is between adjacent I/O pins.



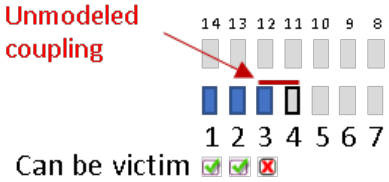



```
1  Pin_I/O      pin_name     1
2  Pin_I/O      pin_name     2
3  Pin_I/O      pin_name     3
4  Pin_I/O      pin_name     4
5  Pin_I/O      pin_name     5
6  Pin_I/O      pin_name     6
7  Pin_Rail     signal_name  VSS
8  Pin_I/O      pin_name     8
9  Pin_I/O      pin_name     9
10 Pin_I/O      pin_name     10
11 Pin_I/O      pin_name     11
12 Pin_I/O      pin_name     12
13 Pin_I/O      pin_name     13
14 Pin_Rail     signal_name  VDD
15 Buffer_I/O   pin_name     1
16 Buffer_I/O   pin_name     2
17 Buffer_I/O   pin_name     3
18 Buffer_I/O   pin_name     4
19 Buffer_I/O   pin_name     5
20 Buffer_I/O   pin_name     6
21 Buffer_Rail  signal_name  VSS
22 Buffer_I/O   pin_name     8
23 Buffer_I/O   pin_name     9
24 Buffer_I/O   pin_name     10
25 Buffer_I/O   pin_name     11
26 Buffer_I/O   pin_name     12
27 Buffer_I/O   pin_name     13
28 Buffer_Rail  signal_name  VDD
```

Partial models support victim modeling only where all coupled pins are modeled



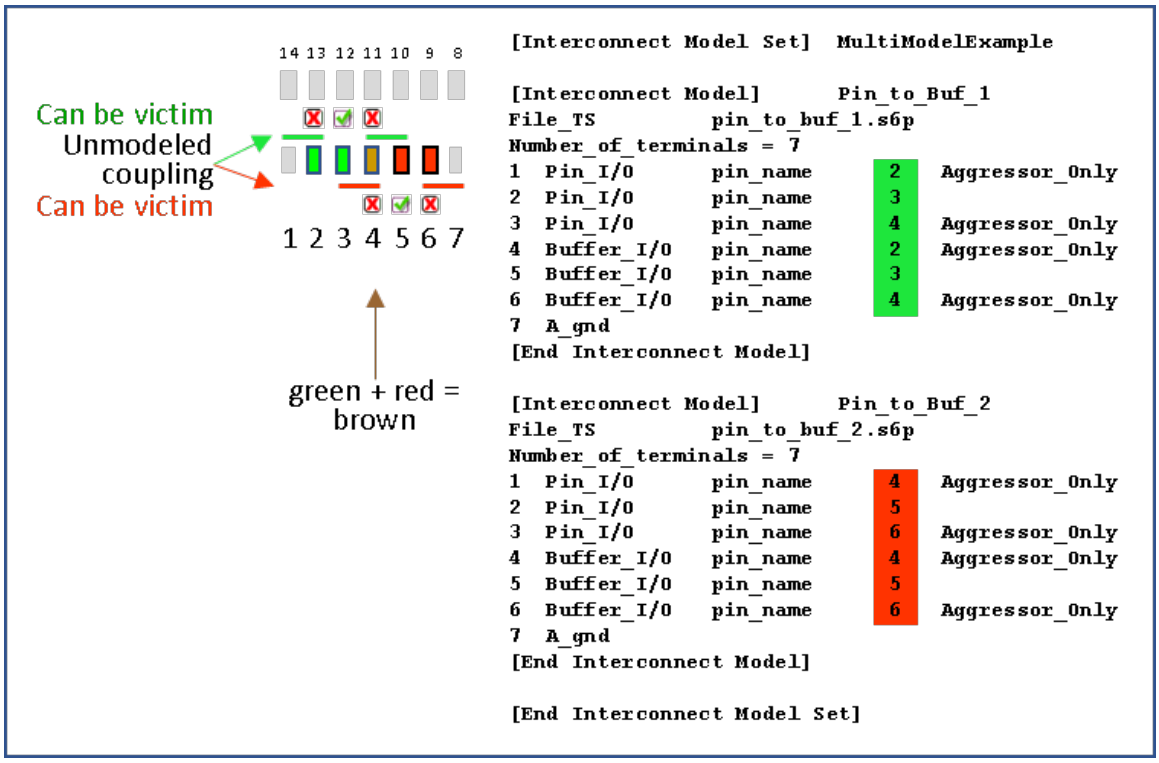

```
1  Pin_I/O      pin_name     2  Aggressor_Only
2  Pin_I/O      pin_name     3
3  Pin_I/O      pin_name     4  Aggressor_Only
4  Buffer_I/O   pin_name     2  Aggressor_Only
5  Buffer_I/O   pin_name     3
6  Buffer_I/O   pin_name     4  Aggressor_Only
```




```
1  Pin_I/O      pin_name     1
2  Pin_I/O      pin_name     2
3  Pin_I/O      pin_name     3  Aggressor_Only
4  Buffer_I/O   pin_name     1
5  Buffer_I/O   pin_name     2
6  Buffer_I/O   pin_name     3  Aggressor_Only
```

Figure 51 – Aggressor_Only Examples

Figure 52 illustrates a special situation when a pin (pin 4 in this case) is associated with more than one Interconnect Model within the same Interconnect Model Group in one or more Interconnect Model Sets, and all of the terminal lines associated with that pin are marked as Aggressor_Only. The first Interconnect Model in this example is associated with pins 2-4 and is shown in green. The second Interconnect Model is associated with pins 4-6 and is shown in red. Note that pin 4 is marked as Aggressor_Only in both Interconnect Models, and there are no other Interconnect Models referenced through Interconnect Model Set(s) in this Interconnect Model Group with a terminal line for pin 4 without the Aggressor_Only marking. Since EDA tools are not expected to provide a selection user interface for Interconnect Models in Interconnect Model Sets, this case would present an ambiguity if the user wanted to run a simulation with pin 4.




```
[Interconnect Model Set]  MultiModelExample

[Interconnect Model]      Pin_to_Buf_1
File_TS          pin_to_buf_1.s6p
Number_of_terminals = 7
1  Pin_I/O       pin_name      2   Aggressor_Only
2  Pin_I/O       pin_name      3
3  Pin_I/O       pin_name      4   Aggressor_Only
4  Buffer_I/O    pin_name      2   Aggressor_Only
5  Buffer_I/O    pin_name      3
6  Buffer_I/O    pin_name      4   Aggressor_Only
7  A_gnd
[End Interconnect Model]

[Interconnect Model]      Pin_to_Buf_2
File_TS          pin_to_buf_2.s6p
Number_of_terminals = 7
1  Pin_I/O       pin_name      4   Aggressor_Only
2  Pin_I/O       pin_name      5
3  Pin_I/O       pin_name      6   Aggressor_Only
4  Buffer_I/O    pin_name      4   Aggressor_Only
5  Buffer_I/O    pin_name      5
6  Buffer_I/O    pin_name      6   Aggressor_Only
7  A_gnd
[End Interconnect Model]

[End Interconnect Model Set]
```

Figure 52 – A Special Case with Aggressor_Only

11.2.1 CONNECTING PINS, PADS AND BUFFER TERMINALS

Terminal lines describe the IBIS-ISS node or Touchstone port that each terminal should be connected to. Terminals may be at pins, die pads or the buffer. The arrangement of the terminal line entries (columns) is described below.

- The first column, Terminal_number, contains an integer between 1 and the Number_of_terminals that describes the ordinal (positional) number of the IBIS-ISS node in the [Interconnect Model] subcircuit or Touchstone file port. The second column is Terminal_type, the third column is Terminal_type_qualifier, the fourth column is Qualifier_entry and there is an optional fifth column “Aggressor_Only”.

- The second column, Terminal_type, determines if the terminal is at a pin, die pad or buffer.
  - For I/O connections
    - At pins, die pads or buffers
      - Terminal_type can be Pin_I/O, Pad_I/O and Buffer_I/O
      - Terminal_type_qualifier shall be pin_name
      - Qualifier_entry shall be the pin_name of an I/O pin

  - For rail connections
    - At pins
      - Terminal_type shall be Pin_Rail
      - Terminal_type_qualifier shall be one of the following
        - pin_name
          - Qualifier_entry shall be a rail pin_name
        - signal_name
          - Qualifier_entry shall be a rail signal_name
        - bus_label
          - Qualifier_entry shall be a bus_label

    - At die pads
      - Terminal_type shall be Pad_Rail
      - Terminal_type_qualifier shall be
        - signal_name
          - Qualifier_entry shall be a rail signal_name
        - bus_label
          - Qualifier_entry shall be a bus_label
        - pad_name
          - Qualifier_entry shall be the pad_name of a rail pad

    - At buffers
      - Terminal_type shall be Buffer_Rail or any of the five *_ref terminals associated with an I/O buffer below
      - Buffer_Rail Terminal_type_qualifier shall be
        - signal_name
          - Qualifier_entry shall be a rail signal_name
        - bus_label
          - Qualifier_entry shall be a bus_label

- Pullup_ref, Pulldown_ref, Power_clamp_ref, Gnd_clamp_ref or Ext_ref Terminal_type_qualifiers shall be
  - pin_name
    - Qualifier_entry shall be the I/O buffer pin_name

- At any interface
  - Terminal_type A_gnd is available at any interface and without any Terminal_type qualifier

Table 49 summarizes the rules described above.

Table 49 – Allowed Terminal_type Associations for Interconnect Models[1]

Terminal_type	Terminal_type_qualifier				Aggressor_Only
	pin_name	signal_name	bus_label	pad_name	
Pin_I/O	X				A
Pad_I/O	X				A
Buffer_I/O	X				A
Pin_Rail	Y	Y	Y		
Pad_Rail		Y	Y	Z	
Buffer_Rail		Y	Y		
Pullup_ref	X				
Pulldown_ref	X				
Power_clamp_ref	X				
Gnd_clamp_ref	X				
Ext_ref	X				
A_gnd					

Notes:

1) In the table, "X" refers to I/O pin names. "Y" and "Z" are POWER and GND names. The letter "A" designates "Aggressor_Only".

Each terminal of an interface represents either 1) a list of pins at the pin interface, 2) a list of die pads at the die pad interface, or 3) a list of buffer model terminals. It is illegal in one interface, in

one model, for 1) a pin to appear in two terminals, 2) a die pad to appear in two terminals, or 3) a buffer model terminal to appear in two terminals.

For I/O terminals, the pin_name value shall not be repeated at any one interface. For rail terminals, the rail terminal name shall not be repeated at any one interface. Also, a rail terminal name that overlaps with another rail terminal name (expressed as pin_name, pad_name, bus_label, signal_name) shall not be entered at any one interface. For example, if the [Pin] keyword contains the following row:

```
[Pin]
…
10  VDD POWER
…
```

then signal_name VDD overlaps with pin_name 10. So, Terminal_type lines "Pin_Rail signal_name VDD" and "Pin_Rail pin_name 10" shall not both be entered in a single Interconnect Model.

For Interconnect Model Sets containing several Interconnect Models, the Terminal_types at the same interface are considered connected if the terminal names match. I/O terminals assigned to the same pin_name at the die pad interface in two Interconnect Models are connected. For rail terminals, identical names are connected and rail terminal names that overlap with another rail terminal name are connected. An excepton exists if the Interconnect Models are not to be used together because of different Aggressor_Only entries, as illustrated in Figure 51 and Figure 52 above. In these cases, overlapping I/O pin_names are permitted because the Interconnect Models are not to be used together in simulations. The rails connections and paths in the unused Interconnect Models are also not used.

When an Interconnect Model Group references several Interconnect Model Sets as shown under the [Interconnect Model Group] keyword, the same connection rules apply for all Interconnect Models in the Interconnect Model Sets that are used in the simulation.

In the examples below, the Interconnect Models have unique Terminal_type names at each interface. Some examples illustrate several Interconect Models within an Interconnect Model Set with identical or overlapping Terminal_type names. During simulations, the EDA tool should connect these terminals.

Examples:

```
| All examples show a [Interconnect Model Set] for grouping of the
|  [Interconnect Model] descriptions that can be referenced
|
| Naming convention for [Interconnect Model Set] examples is below
|  ([Interconnect Model] may show additional details)
|
| Full        - Includes all I/O pins
| A1 or A1_A3 - Designated pin or pins
| TS          - Touchstone representation
| ISS         - IBIS-ISS representation
| PDN         - Includes power delivery network, can also be PU and PD
| IO          - Only if modified differently than PDN below for buf_pad_pin
```

```
| buf_pad_pin – Includes models for buf_pad, pad_pin; if missing, buf_pad
| sn          - Uses signal_name; if missing assumes pin_name
| bl          - Uses bus_label; if missing assumes pin_name
| pn          - Uses pad_name; if missing assumes pin_name
| XTALK       - Cross talk analysis (coupled nets may include Aggressor_Only)

| Examples 1 – 11 apply to the configuration below:

[Pin] signal_name model_name      R_pin   L_pin   C_pin
A1    DQ1         DQ
A2    DQ2         DQ
A3    DQ3         DQ
D1    DQS+        DQS
D2    DQS-        DQS
P1    VDD         POWER
P2    VDD         POWER
P3    VDD         POWER
P4    VDD         POWER
P5    VDD         POWER
G1    VSS         GND
G2    VSS         GND
G3    VSS         GND
G4    VSS         GND

[Diff Pin] inv_pin  vdiff  tdelay_typ tdelay_min tdelay_max
D1         D2       NA     NA         NA         NA

[Die Supply Pads]  signal_name bus_label
VDD1               VDD
VDD2               VDD
VDD3               VDD
VSS1               VSS
VSS2               VSS

[Pin Mapping] pulldown_ref pullup_ref gnd_clamp_ref power_clamp_ref ext_ref
A1            VSS          VDD        NC            NC              NC
A2            VSS          VDD        NC            NC              NC
A3            VSS          VDD        NC            NC              NC
D1            VSS          VDD        NC            NC              NC
D2            VSS          VDD        NC            NC              NC
| Pins below are optional per [Pin Mapping] rules
P1            NC           VDD
P2            NC           VDD
P3            NC           VDD
P4            NC           VDD
P5            NC           VDD
G1            VSS          NC
G2            VSS          NC
G3            VSS          NC
G4            VSS          NC

|*****************************************************************************
|
| Example 1: Terminals for full IBIS-ISS component with PDN, as depicted below.
|
[Interconnect Model Set]      Full_ISS_PDN_1
```

```
|-----
[Interconnect Model]          Full_ISS_buf_pin_1
File_IBIS-ISS   full_buf_pin_1.iss           full_buf_pin_typ
Number_of_terminals = 29
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      D1    |  DQS+        DQS
5  Pin_I/O      pin_name      D2    |  DQS-        DQS
6  Pin_Rail     pin_name      P1    |  VDD         POWER
7  Pin_Rail     pin_name      P2    |  VDD         POWER
8  Pin_Rail     pin_name      P3    |  VDD         POWER
9  Pin_Rail     pin_name      P4    |  VDD         POWER
10 Pin_Rail     pin_name      P5    |  VDD         POWER
11 Pin_Rail     pin_name      G1    |  VSS         GND
12 Pin_Rail     pin_name      G2    |  VSS         GND
13 Pin_Rail     pin_name      G3    |  VSS         GND
14 Pin_Rail     pin_name      G4    |  VSS         GND
15 Buffer_I/O   pin_name      A1    |  DQ1         DQ
16 Buffer_I/O   pin_name      A2    |  DQ2         DQ
17 Buffer_I/O   pin_name      A3    |  DQ3         DQ
18 Buffer_I/O   pin_name      D1    |  DQS+        DQS
19 Buffer_I/O   pin_name      D2    |  DQS-        DQS
20 Pullup_ref   pin_name      A1    |  DQ1         DQ
21 Pullup_ref   pin_name      A2    |  DQ2         DQ
22 Pullup_ref   pin_name      A3    |  DQ3         DQ
23 Pullup_ref   pin_name      D1    |  DQS+        DQS
24 Pullup_ref   pin_name      D2    |  DQS-        DQS
25 Pulldown_ref pin_name      A1    |  DQ1         DQ
26 Pulldown_ref pin_name      A2    |  DQ2         DQ
27 Pulldown_ref pin_name      A3    |  DQ3         DQ
28 Pulldown_ref pin_name      D1    |  DQS+        DQS
29 Pulldown_ref pin_name      D2    |  DQS+        DQS
[End Interconnect Model]
[End Interconnect Model Set]
```

The internal electrical connection paths established by Example 1 above are graphically depicted in Figure 53. An illustration of external connections to the terminals of that device is shown in Figure 54.

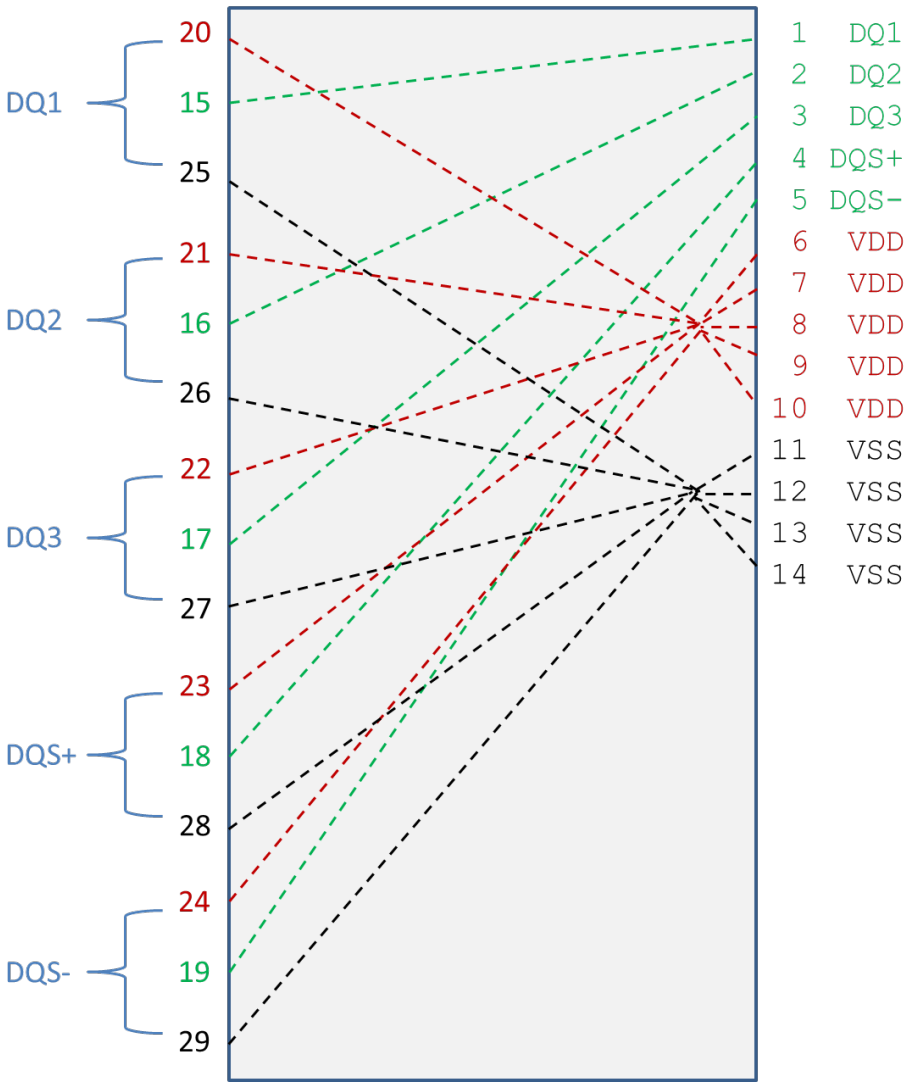


Figure 53 – Electrical Connections for Full Buffer Pin Model with Power Routing

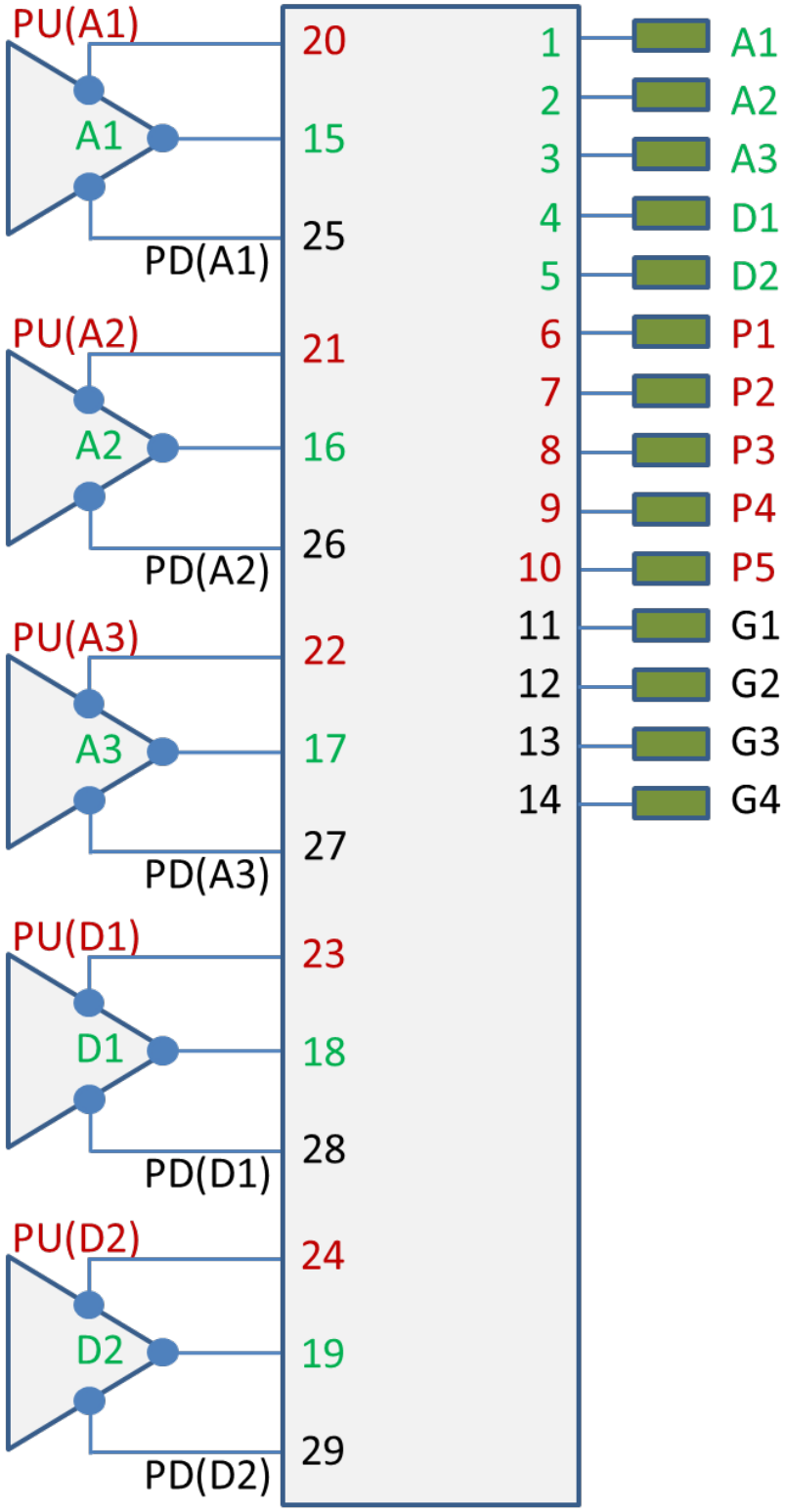


Figure 54 – Electrical Terminals for Full Buffer Pin Model with Power Routing

```
|*****************************************************************************
|
| Example 2: Same as Example 1 except the PDN networks are simplified with
|   signal_name qualifiers to create a pair of POWER terminals and a pair
|   of GND terminals

[Interconnect Model Set]      Full_ISS_PDN_sn_2
|-----
[Interconnect Model]          Full_ISS_buf_pin_2
File_IBIS-ISS   full_buf_pin.iss             full_buf_pin_2_typ
Number_of_terminals = 14

1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      D1    |  DQS+        DQS
5  Pin_I/O      pin_name      D2    |  DQS-        DQS
|
| POWER and GND terminals with signal_names
|
6  Pin_Rail     signal_name   VDD   |  VDD         POWER
7  Pin_Rail     signal_name   VSS   |  VSS         GND
|
8  Buffer_I/O   pin_name      A1    |  DQ1         DQ
9  Buffer_I/O   pin_name      A2    |  DQ2         DQ
10 Buffer_I/O   pin_name      A3    |  DQ3         DQ
11 Buffer_I/O   pin_name      D1    |  DQS+        DQS
12 Buffer_I/O   pin_name      D2    |  DQS-        DQS
|
| POWER and GND terminals with signal_names
|
13 Buffer_Rail  signal_name   VDD   |  VDD         POWER
14 Buffer_Rail  signal_name   VSS   |  VSS         GND
|
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 2b: Same as Example 2 except using File_TS with a single, common
|             reference for all ports
|
[Interconnect Model Set]      Full_TS_PDN_sn_2b
|-----
[Interconnect Model]          Full_TS_buf_pin_2b
File_TS         full_buf_pin.s14p
Number_of_terminals = 15
|
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      D1    |  DQS+        DQS
5  Pin_I/O      pin_name      D2    |  DQS-        DQS
|
6  Pin_Rail     signal_name   VDD   |  VDD         POWER
7  Pin_Rail     signal_name   VSS   |  VSS         GND
```

```
|
8  Buffer_I/O   pin_name      A1     |  DQ1          DQ
9  Buffer_I/O   pin_name      A2     |  DQ2          DQ
10 Buffer_I/O   pin_name      A3     |  DQ3          DQ
11 Buffer_I/O   pin_name      D1     |  DQS+         DQS
12 Buffer_I/O   pin_name      D2     |  DQS-         DQS
|
13 Buffer_Rail  signal_name   VDD    |  VDD          POWER
14 Buffer_Rail  signal_name   VSS    |  VSS          GND
15 A_gnd
|
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 3: Single I/O Touchstone connection with one extra terminal for the
|   N+1 .s2p reference terminal
|
[Interconnect Model Set]      A1_TS
|-----
[Interconnect Model]          A1_TS_buf_pin
File_TS         dq_ts_buf_pin.s2p
Number_of_terminals = 3
1  Pin_I/O      pin_name      A1
2  Buffer_I/O   pin_name      A1
3  A_gnd                          | A_gnd reference for .s2p file
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 4: Single I/O pin documenting both IBIS-ISS and Touchstone files and
|   showing that the File_TS Touchstone N+1 reference connection is to A_gnd
|
[Interconnect Model Set]      A1_TS_pad_pin_ISS_buf_pad
|-----
[Interconnect Model]          A1_TS_pad_pin
File_TS         dq_ts_pad_pin.s2p
Number_of_terminals = 3
1  Pin_I/O      pin_name      A1
2  Pad_I/O      pin_name      A1
3  A_gnd
|
[End Interconnect Model]
[Interconnect Model]          A1_ISS_buf_pad
File_IBIS-ISS   dq_iss_buf_pad.iss           DQ_buf_pad_typ
Number_of_terminals = 3
1  Pad_I/O      pin_name      A1
2  Buffer_I/O   pin_name      A1
3  A_gnd                          | Explicit connection to A_gnd is preferred
                                  | instead of connections to node0 inside the
                                  | subcircuit
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
```

```
|
| Example 5: Full I/O IBIS-ISS configuration with PDN terminals in a separate
|   [Interconnect Model Set]; when connected the individual Pin_Rail
|   terminals G1-G4 become shorted together with common VSS reference

[Interconnect Model Set]     Full_ISS_buf_pin_IO_1
|-----
[Interconnect Model]         Full_ISS_buf_pin_IO
File_IBIS-ISS   full_buf_pin.iss            full_buf_pin_typ
Number_of_terminals = 11
1  Pin_I/O      pin_name      A1   |   DQ1         DQ
2  Pin_I/O      pin_name      A2   |   DQ2         DQ
3  Pin_I/O      pin_name      A3   |   DQ3         DQ
4  Pin_I/O      pin_name      D1   |   DQS+        DQS
5  Pin_I/O      pin_name      D2   |   DQS-        DQS
6  Buffer_I/O   pin_name      A1   |   DQ1         DQ
7  Buffer_I/O   pin_name      A2   |   DQ2         DQ
8  Buffer_I/O   pin_name      A3   |   DQ3         DQ
9  Buffer_I/O   pin_name      D1   |   DQS+        DQS
10 Buffer_I/O   pin_name      D2   |   DQS-        DQS
11 Pin_Rail     signal_name   VSS  |   VSS         GND
[End Interconnect Model]
[End Interconnect Model Set]

[Interconnect Model Set]      Full_ISS_buf_pin_PDN_1
|-----
[Interconnect Model]          Full_ISS_buf_pin_PDN_1
File_IBIS-ISS   full_ISS_buf_pin_pdn.iss      full_buf_pin_PDN_typ
Number_of_terminals = 19
1  Pin_Rail     pin_name      P1    |  VDD         POWER
2  Pin_Rail     pin_name      P2    |  VDD         POWER
3  Pin_Rail     pin_name      P3    |  VDD         POWER
4  Pin_Rail     pin_name      P4    |  VDD         POWER
5  Pin_Rail     pin_name      P5    |  VDD         POWER
6  Pullup_ref   pin_name      A1    |  DQ1         DQ
7  Pullup_ref   pin_name      A2    |  DQ2         DQ
8  Pullup_ref   pin_name      A3    |  DQ3         DQ
9  Pullup_ref   pin_name      D1    |  DQS+        DQS
10 Pullup_ref   pin_name      D2    |  DQS-        DQS
11 Pin_Rail     pin_name      G1    |  VSS         GND
12 Pin_Rail     pin_name      G2    |  VSS         GND
13 Pin_Rail     pin_name      G3    |  VSS         GND
14 Pin_Rail     pin_name      G4    |  VSS         GND
15 Pulldown_ref pin_name      A1    |  DQ1         DQ
16 Pulldown_ref pin_name      A2    |  DQ2         DQ
17 Pulldown_ref pin_name      A3    |  DQ3         DQ
18 Pulldown_ref pin_name      D1    |  DQS+        DQS
19 Pulldown_ref pin_name      D2    |  DQS-        DQS
[End Interconnect Model]
[End Interconnect Model Set]

|*************************************************************************
|
| Example 6: Full IBIS-ISS IOs and separate PDNs, all with buf_pad and
|   pad_pin [Interconnect Model]s in separate [Interconnect Model]s
```

```
[Interconnect Model Set]      Full_ISS_buf_pad_pin_PDN_4
|-----
[Interconnect Model]          Full_ISS_pad_pin_IO
File_IBIS-ISS   full_pad_pin_io.iss          full_pad_pin_IO_typ
Number_of_terminals = 11
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      D1    |  DQS+        DQS
5  Pin_I/O      pin_name      D2    |  DQS-        DQS
|
6  Pad_I/O      pin_name      A1    |  DQ1         DQ
7  Pad_I/O      pin_name      A2    |  DQ2         DQ
8  Pad_I/O      pin_name      A3    |  DQ3         DQ
9  Pad_I/O      pin_name      D1    |  DQS+        DQS
10 Pad_I/O      pin_name      D2    |  DQS-        DQS
11 Pin_Rail     signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_buf_pad_IO
File_IBIS-ISS   full_buf_pad_io.iss          full_buf_pad_IO_typ
Number_of_terminals = 11
1  Pad_I/O      pin_name      A1    |  DQ1         DQ
2  Pad_I/O      pin_name      A2    |  DQ2         DQ
3  Pad_I/O      pin_name      A3    |  DQ3         DQ
4  Pad_I/O      pin_name      D1    |  DQS+        DQS
5  Pad_I/O      pin_name      D2    |  DQS-        DQS
|
6  Buffer_I/O   pin_name      A1    |  DQ1         DQ
7  Buffer_I/O   pin_name      A2    |  DQ2         DQ
8  Buffer_I/O   pin_name      A3    |  DQ3         DQ
9  Buffer_I/O   pin_name      D1    |  DQS+        DQS
10 Buffer_I/O   pin_name      D2    |  DQS-        DQS
11 Buffer_Rail  signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_pad_pin_PDN
File_IBIS-ISS   full_iss_pad_pin_pdn.iss     full_iss_pad_pin_PDN_typ
Number_of_terminals = 14
1  Pin_Rail     pin_name      P1    |  VDD         POWER
2  Pin_Rail     pin_name      P2    |  VDD         POWER
3  Pin_Rail     pin_name      P3    |  VDD         POWER
4  Pin_Rail     pin_name      P4    |  VDD         POWER
5  Pin_Rail     pin_name      P5    |  VDD         POWER
|
6  Pad_Rail     pad_name      VDD1  |  VDD         POWER
7  Pad_Rail     pad_name      VDD2  |  VDD         POWER
8  Pad_Rail     pad_name      VDD3  |  VDD         POWER
|
9  Pin_Rail     pin_name      G1    |  VSS         GND
10 Pin_Rail     pin_name      G2    |  VSS         GND
11 Pin_Rail     pin_name      G3    |  VSS         GND
12 Pin_Rail     pin_name      G4    |  VSS         GND
|
13 Pad_Rail     pad_name      VSS1  |  VSS         GND
14 Pad_Rail     pad_name      VSS2  |  VSS         GND
```

```
[End Interconnect Model]

[Interconnect Model]          Full_ISS_buf_pad_PDN
File_IBIS-ISS   full_iss_buf_pad_pdn.iss     full_iss_buf_pad_PDN_typ
Number_of_terminals = 15
1  Pad_Rail     pad_name      VDD1   |  VDD         POWER
2  Pad_Rail     pad_name      VDD2   |  VDD         POWER
3  Pad_Rail     pad_name      VDD3   |  VDD         POWER
|
4  Pullup_ref   pin_name      A1     |  DQ1         DQ
5  Pullup_ref   pin_name      A2     |  DQ2         DQ
6  Pullup_ref   pin_name      A3     |  DQ3         DQ
7  Pullup_ref   pin_name      D1     |  DQS+        DQS
8  Pullup_ref   pin_name      D2     |  DQS-        DQS
|
9  Pad_Rail     pad_name      VSS1   |  VSS         GND
10 Pad_Rail     pad_name      VSS2   |  VSS         GND
|
11 Pulldown_ref pin_name      A1     |  DQ1         DQ
12 Pulldown_ref pin_name      A2     |  DQ2         DQ
13 Pulldown_ref pin_name      A3     |  DQ3         DQ
14 Pulldown_ref pin_name      D1     |  DQS+        DQS
15 Pulldown_ref pin_name      D2     |  DQS-        DQS
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 7: Full IBIS-ISS model with I/O only [Interconnect Model] and a
|   separate PDN [Interconnect Model] with signal_name qualifiers

[Interconnect Model Set]     Full_ISS_PDN_sn_5
|-----
[Interconnect Model]         Full_ISS_buf_pin_IO
File_IBIS-ISS   full_buf_pin.iss            full_buf_pin_typ
Number_of_terminals = 11
1  Pin_I/O      pin_name      A1    |   DQ1         DQ
2  Pin_I/O      pin_name      A2    |   DQ2         DQ
3  Pin_I/O      pin_name      A3    |   DQ3         DQ
4  Pin_I/O      pin_name      D1    |   DQS+        DQS
5  Pin_I/O      pin_name      D2    |   DQS-        DQS
6  Buffer_I/O   pin_name      A1    |   DQ1         DQ
7  Buffer_I/O   pin_name      A2    |   DQ2         DQ
8  Buffer_I/O   pin_name      A3    |   DQ3         DQ
9  Buffer_I/O   pin_name      D1    |   DQS+        DQS
10 Buffer_I/O   pin_name      D2    |   DQS-        DQS
11 Pin_Rail     signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_buf_pin_PDN_2
File_IBIS-ISS   full_iss_buf_pin_pdn_2.iss   full_iss_buf_pad_PDN_2
Number_of_terminals = 4
1  Pin_Rail     signal_name   VDD   |  VDD         POWER
2  Buffer_Rail  signal_name   VDD   |  VDD         POWER
3  Pin_Rail     signal_name   VSS   |  VSS         GND
4  Buffer_Rail  signal_name   VSS   |  VSS         GND
```

```
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 8: Same full IBIS-ISS model with PDN as in Example 7, but with the
|   [Interconnect Model]s describing buf_pad and pad_pin connections
|   separately

[Interconnect Model Set]      Full_ISS_buf_pad_pin_PDN_sn_6
|-----
[Interconnect Model]          Full_ISS_pad_pin_IO
File_IBIS-ISS   full_pad_pin_io.iss          full_pad_pin_IO_typ
Number_of_terminals = 11
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      D1    |  DQS+        DQS
5  Pin_I/O      pin_name      D2    |  DQS-        DQS
|
6  Pad_I/O      pin_name      A1    |  DQ1         DQ
7  Pad_I/O      pin_name      A2    |  DQ2         DQ
8  Pad_I/O      pin_name      A3    |  DQ3         DQ
9  Pad_I/O      pin_name      D1    |  DQS+        DQS
10 Pad_I/O      pin_name      D2    |  DQS-        DQS
11 Pin_Rail     signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_buf_pad_IO
File_IBIS-ISS   full_buf_pad_io.iss          full_buf_pad_IO_typ
Number_of_terminals = 11
1  Pad_I/O      pin_name      A1    |  DQ1         DQ
2  Pad_I/O      pin_name      A2    |  DQ2         DQ
3  Pad_I/O      pin_name      A3    |  DQ3         DQ
4  Pad_I/O      pin_name      D1    |  DQS+        DQS
5  Pad_I/O      pin_name      D2    |  DQS-        DQS
|
6  Buffer_I/O   pin_name      A1    |  DQ1         DQ
7  Buffer_I/O   pin_name      A2    |  DQ2         DQ
8  Buffer_I/O   pin_name      A3    |  DQ3         DQ
9  Buffer_I/O   pin_name      D1    |  DQS+        DQS
10 Buffer_I/O   pin_name      D2    |  DQS-        DQS
11 Buffer_Rail  signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_pad_pin_PDN_3
File_IBIS-ISS   full_iss_pad_pin_pdn_3.iss   full_iss_pad_pin_pdn_3
Number_of_terminals = 4
1  Pin_Rail     signal_name   VDD   |  VDD         POWER
2  Pad_Rail     signal_name   VDD   |  VDD         POWER
3  Pin_Rail     signal_name   VSS   |  VSS         GND
4  Pad_Rail     signal_name   VSS   |  VSS         GND
[End Interconnect Model]

[Interconnect Model]          Full_ISS_buf_pad_PDN_3
File_IBIS-ISS   full_iss_buf_pad_pdn_3.iss   full_iss_buf_pad_pdn_3
Number_of_terminals = 4
```

```
1  Buffer_Rail  signal_name   VDD   |  VDD         POWER
2  Pad_Rail     signal_name   VDD   |  VDD         POWER
3  Buffer_Rail  signal_name   VSS   |  VSS         GND
4  Pad_Rail     signal_name   VSS   |  VSS         GND
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 9: Same full IBIS-ISS configuration with PDN as in Example 8, except
|   that I/O connections are direct from buf_pin while the PDN connections are
|   from buf_pad and pad_pin using the signal_name qualifier

[Interconnect Model Set]     Full_ISS_IO_buf_pad_pin_PDN_sn_7
|-----
[Interconnect Model]         Full_ISS_buf_pin_IO
File_IBIS-ISS   full_buf_pin.iss            full_buf_pin_typ
Number_of_terminals = 11
1  Pin_I/O      pin_name      A1   |   DQ1         DQ
2  Pin_I/O      pin_name      A2   |   DQ2         DQ
3  Pin_I/O      pin_name      A3   |   DQ3         DQ
4  Pin_I/O      pin_name      D1   |   DQS+        DQS
5  Pin_I/O      pin_name      D2   |   DQS-        DQS
6  Buffer_I/O   pin_name      A1   |   DQ1         DQ
7  Buffer_I/O   pin_name      A2   |   DQ2         DQ
8  Buffer_I/O   pin_name      A3   |   DQ3         DQ
9  Buffer_I/O   pin_name      D1   |   DQS+        DQS
10 Buffer_I/O   pin_name      D2   |   DQS-        DQS
11 Pin_Rail     signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_pad_pin_PDN_3
File_IBIS-ISS   full_iss_pad_pin_pdn_3.iss   full_iss_pad_pin_pdn_3
Number_of_terminals = 4
1  Pin_Rail     signal_name   VDD   |  VDD         POWER
2  Pad_Rail     signal_name   VDD   |  VDD         POWER
3  Pin_Rail     signal_name   VSS   |  VSS         GND
4  Pad_Rail     signal_name   VSS   |  VSS         GND
[End Interconnect Model]

[Interconnect Model]          Full_ISS_buf_pad_PDN_3
File_IBIS-ISS   full_iss_buf_pad_pdn_3.iss   full_iss_buf_pad_pdn_3
Number_of_terminals = 4
1  Buffer_Rail  signal_name   VDD   |  VDD         POWER
2  Pad_Rail     signal_name   VDD   |  VDD         POWER
3  Buffer_Rail  signal_name   VSS   |  VSS         GND
4  Pad_Rail     signal_name   VSS   |  VSS         GND
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 10: Terminals A1 through A3 are IBIS-ISS connections with coupling
|   for crosstalk analysis – Aggressor_Only pins and buffers are labeled
|
[Interconnect Model Set]      A1_A3_DQ_TS_XTALK
```

```
|-----
[Interconnect Model]          A1_A3_DQ_TS_buf_pin_XTALK
File_TS  dq_buf_pin_xtalk.s6p
Number_of_terminals = 7
1  Pin_I/O      pin_name      A1    Aggressor_Only
2  Buffer_I/O   pin_name      A1    Aggressor_Only
3  Pin_I/O      pin_name      A2
4  Buffer_I/O   pin_name      A2
5  Pin_I/O      pin_name      A3    Aggressor_Only
6  Buffer_I/O   pin_name      A3    Aggressor_Only
7  A_gnd
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 11: Same as Example 10, but with a PDN network added
|   While this syntax is valid, it is not recommended because the S-parameter
|   reference terminal is not connected to the VDD or VSS rail from the
|   IBIS-ISS model.
|
[Interconnect Model Set]      A1_A3_DQ_TS_XTALK_ISS_PDN
|-----
[Interconnect Model]          A1_A3_DQ_TS_buf_pin_XTALK
File_TS  dq_buf_pin_xtalk.s6p
Number_of_terminals = 7
1  Pin_I/O      pin_name      A1    Aggressor_Only
2  Buffer_I/O   pin_name      A1    Aggressor_Only
3  Pin_I/O      pin_name      A2
4  Buffer_I/O   pin_name      A2
5  Pin_I/O      pin_name      A3    Aggressor_Only
6  Buffer_I/O   pin_name      A3    Aggressor_Only
7  A_gnd
[End Interconnect Model]
|
[Interconnect Model]          Full_ISS_buf_pin_PDN_2
File_IBIS-ISS   full_iss_buf_pin_pdn_2.iss   full_iss_buf_pin_PDN_2
Number_of_terminals = 4
1  Pin_Rail     signal_name   VDD   |  VDD         POWER
2  Buffer_Rail  signal_name   VDD   |  VDD         POWER
3  Pin_Rail     signal_name   VSS   |  VSS         GND
4  Buffer_Rail  signal_name   VSS   |  VSS         GND
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Examples 12 and 13 apply to the configuration below

[Pin] signal_name model_name      R_pin   L_pin   C_pin
A1    DQ1         DQ
A2    DQ2         DQ
A3    DQ3         DQ
A4    DQ4         DQ
P1    VDD         POWER
P2    VDD         POWER
G1    VSS         GND
```

```
G2    VSS         GND

[Bus Label] signal_name
VDD1        VDD
VDD2        VDD

[Pin Mapping] pulldown_ref pullup_ref gnd_clamp_ref power_clamp_ref ext_ref
A1            VSS           VDD1        NC            NC              NC
A2            VSS           VDD1        NC            NC              NC
A3            VSS           VDD2        NC            NC              NC
A4            VSS           VDD2        NC            NC              NC
| Entries below may optionally be deleted and replaced with [Bus Label] per
| [Bus Label] and [Pin Mapping] rules
P1            NC            VDD1        NC            NC              NC
P2            NC            VDD2        NC            NC              NC
G1            VSS           NC          NC            NC              NC
G2            VSS           NC          NC            NC              NC

|*****************************************************************************
|
| Example 12: Full IBIS-ISS configuration with PDN described using both
|   bus_label and signal_name qualifiers for the Rails

[Interconnect Model Set]      Full_ISS_IO_PDN_bl_sn_6
|-----
[Interconnect Model]          Full_ISS_buf_pin_IO_4
File_IBIS-ISS   full_iss_buf_pin_io_4.iss    full_iss_buf_pin_IO_4_typ
Number_of_terminals = 9
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      A4    |  DQ4         DQ
5  Buffer_I/O   pin_name      A1    |  DQ1         DQ
6  Buffer_I/O   pin_name      A2    |  DQ2         DQ
7  Buffer_I/O   pin_name      A3    |  DQ3         DQ
8  Buffer_I/O   pin_name      A4    |  DQ4         DQ
9  Pin_Rail     signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_PDN_bl_sn
File_IBIS-ISS   buf_pin_pdn.iss      buf_pin_PDN_typ
Number_of_terminals = 5
1  Pin_Rail     signal_name   VDD   |  VDD         POWER
2  Pin_Rail     signal_name   VSS   |  VSS         GND
|
3  Buffer_Rail  bus_label     VDD1  |  VDD         POWER
4  Buffer_Rail  bus_label     VDD2  |  VDD         POWER
5  Buffer_Rail  signal_name   VSS   |  VSS         GND
[End Interconnect Model]
[End Interconnect Model Set]

| The EDA tool connects the terminals and pins as follows:
|
| 1 Pins P1 and P2
| 2 Pins G1 and G2
| 3 Pullup_ref of buffers A1 and A2
```

```
| 4 Pullup_ref of buffers A3 and A4
| 5 Pulldown_ref of buffers A1, A2, A3 and A4


|*****************************************************************************
|
| Example 13: Same as Example 12, but adds decoupling capacitors at the buffer
|   interface in separate Interconnect Models to show how single-interface
|   Interconnect Models with rail-only terminals can be used

[Interconnect Model Set]      Full_ISS_IO_PDN_bl_sn_7
|-----
[Interconnect Model]          Full_ISS_buf_pin_IO_4
File_IBIS-ISS   full_iss_buf_pin_io_4.iss    full_iss_buf_pin_IO_4_typ
Number_of_terminals = 9
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      A4    |  DQ4         DQ
5  Buffer_I/O   pin_name      A1    |  DQ1         DQ
6  Buffer_I/O   pin_name      A2    |  DQ2         DQ
7  Buffer_I/O   pin_name      A3    |  DQ3         DQ
8  Buffer_I/O   pin_name      A4    |  DQ4         DQ
9  Pin_Rail     signal_name   VSS
[End Interconnect Model]

[Interconnect Model]          Full_ISS_PDN_bl_sn
File_IBIS-ISS   buf_pin_pdn.iss      buf_pin_PDN_typ
Number_of_terminals = 5
1  Pin_Rail     signal_name   VDD   |  VDD         POWER
2  Pin_Rail     signal_name   VSS   |  VSS         GND
|
3  Buffer_Rail  bus_label     VDD1  |  VDD         POWER
4  Buffer_Rail  bus_label     VDD2  |  VDD         POWER
5  Buffer_Rail  signal_name   VSS   |  VSS         GND
[End Interconnect Model]

[Interconnect Model]          Decap1
File_IBIS-ISS   buf_pin_pdn.iss      single_decoupling_cap_model
Number_of_terminals = 2
1  Buffer_Rail  bus_label     VDD1  |  VDD         POWER
2  Buffer_Rail  signal_name   VSS   |  VSS         GND
[End Interconnect Model]

[Interconnect Model]          Decap2
File_IBIS-ISS   buf_pin_pdn.iss      single_decoupling_cap_model
Number_of_terminals = 2
1  Buffer_Rail  bus_label     VDD2  |  VDD         POWER
2  Buffer_Rail  signal_name   VSS   |  VSS         GND
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 14: Full IBIS-ISS configuration with I/Os (and no PDN) and using
|   A_gnd to connect some I/O terminals and the VSS terminal to the simulator
```

```
|   global reference node.
|
|   A_gnd is used to connect the VSS subcircuit terminal (the first terminal)
|   to the simulator global reference, and A_gnd is also used to connect some
|   I/O terminals (3 and 7) to the simulator global reference.

[Interconnect Model Set]      Full_ISS_IO_with_A_gnd
|-----
[Interconnect Model]          Full_ISS_IO_A_gnd
File_IBIS-ISS   full_iss_buf_pin_io_4.iss    full_iss_buf_pin_IO_4_A_gnd_typ
Number_of_terminals = 9
1  A_gnd                            |  VSS terminal connected to simulator
                                    |  global reference
2  Pin_I/O      pin_name      A1    |  DQ1         DQ
3  A_gnd                            |  DQ2         DQ A2 connected to
                                    |               simulator global reference
4  Pin_I/O      pin_name      A3    |  DQ3         DQ
5  Pin_I/O      pin_name      A4    |  DQ4         DQ
6  Buffer_I/O   pin_name      A1    |  DQ1         DQ
7  A_gnd                            |  DQ2         DQ A2 connected to
                                    |                 simulator global reference
8  Buffer_I/O   pin_name      A3    |  DQ3         DQ
9  Buffer_I/O   pin_name      A4    |  DQ4         DQ
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
|
| Example 15: Full Touchstone configuration with I/Os and A_gnd reference,
|   but without any PDN.
|
| A_gnd can be used as a reference only on the terminal line with
| Terminal_number N+1.

[Interconnect Model Set]      Full_TS_IO_A_gnd_reference
|-----
[Interconnect Model]          Full_TS_IO_A_gnd_reference
File_TS         full_ts_buf_pin_io.s8p
Number_of_terminals = 9
Full_TS_IO_A_gnd_reference
1  Pin_I/O      pin_name      A1    |  DQ1         DQ
2  Pin_I/O      pin_name      A2    |  DQ2         DQ
3  Pin_I/O      pin_name      A3    |  DQ3         DQ
4  Pin_I/O      pin_name      A4    |  DQ4         DQ
5  Buffer_I/O   pin_name      A1    |  DQ1         DQ
6  Buffer_I/O   pin_name      A2    |  DQ2         DQ
7  Buffer_I/O   pin_name      A3    |  DQ3         DQ
8  Buffer_I/O   pin_name      A4    |  DQ4         DQ
9  A_gnd                            |  Reference terminal connected to
                                    |  simulator global reference
[End Interconnect Model]
[End Interconnect Model Set]

|*****************************************************************************
```

Keyword: [**End Interconnect Model**]

Required: Yes, for each instance of the [Interconnect Model] keyword

Description: Indicates the end of the Interconnect Model data.

Example:

```
[End Interconnect Model]
```

12 ELECTRICAL MODULE DESCRIPTION (EMD)

12.1 INTRODUCTION

"Module" is a generic term describing a printed circuit board (PCB), multi-chip module (MCM), stacked die component, interposer, or substrate which can contain components or other modules, and which can connect to another board or module through a set of user-visible pins. For example, a DIMM module is a module-level component that is used to attach several DRAM components on the PCB to another module through edge connector pins. The electrical connectivity of such a board or module-level component is described through an "Electrical Module Description". An [EMD Model] defines an electrical model of the interconnect between external pin(s) of the module (referred to elsewhere as "EMD pins"), pin(s) of the designator(s) in the module (referred to elsewhere as "designator pins"), or both. A designator name or set of names (e.g., U23, U24) is associated with distinct part names; this association is defined using the [EMD Designator List] keyword. Each part name is associated with a component in an IBIS (.ibs) file or a module in an EMD (.emd) file; this association is defined using the [EMD Parts] keyword. For designators, the user-visible designator pins are listed under the [Designator Pin List] keyword. Other details are described later.

I/O pins in the [EMD Pin List] and the Designator Pin List that have the same signal_name are considered "connected" by the content of the [EMD Model]. Rail pins in the EMD Pin List and the Designator Pin List that have the same signal_name (or, as applicable, bus_label) are considered connected. This assumption is due to the expectation that some EMD files will be generated automatically from computer-aided design (CAD) layout databases. Each pin in a CAD database is assigned with a CAD "net" (short for "network") name, and when two pins are assigned with the same CAD net name, they are considered connected. Normally, the signal_name of EMD pins and designator pins will be the same as their assigned CAD net name in the layout database. An exception to this is when there are series terminations. In this case the model maker can choose to either:

1) Combine two CAD nets into an "extended net". All the pins in the two CAD nets will use the extended net name as their signal_name in the EMD file. The termination resistor or capacitor would be included in the EMD Models for this extended net. An extended net is defined as the list of EMD and designator pins associated with a common path through an EMD Model.
2) Create separate EMD Models for each CAD net. The termination component must be assigned a designator in this case.

What is and is not included in an EMD Model is defined by its boundaries, referred to here as interfaces. Interfaces exist at the EMD Pin List and Designator Pin List levels.

Terminals are the connection points to IBIS-ISS terminals, Touchstone ports, IBIS Pins, or other EMD Pins defined in each EMD Model. Terminal lines describe the IBIS-ISS terminal or Touchstone port to which each terminal of an EMD Model is connected. Terminals exist at [EMD Pin List] and [Designator Pin List] interfaces.

12.2 EMD FILES

An Electrical Module Description file (a .emd file) describes the connections of a module-level component between the module pins and its components on the module.

Usage Rules:

A .emd file is intended to be a stand-alone file, not referenced by or included in any .ibs, .ebd, or .pkg file. Electrical Module Descriptions are stored in a file whose name is <stem>.emd, where <stem> must conform to the naming rules given in Section 3.1, "FILE NAMING CONVENTIONS" of this specification. The emd extension is mandatory.

Contents:

A .emd file is structured like a standard .ibs file. It must contain the following keywords, as defined in IBIS: [IBIS Ver], [File Name], [File Rev], and [End]. It may also contain the following optional keywords: [Comment Char], [Date], [Source], [Notes], [Disclaimer], and [Copyright].

The actual module description is contained between the keywords [Begin EMD] and [End EMD], and includes the keywords listed below:

.emd file keywords

[Begin EMD]
[Manufacturer]
[Description]
[Number Of EMD Pins]
[EMD Pin List]
[End EMD Pin List]
[EMD Parts]
[End EMD Parts]
[EMD Designator List]
[End EMD Designator List]
[Designator Pin List]
[End Designator Pin List]
[Voltage List]
[End Voltage List]
[EMD Group]
[End EMD Group]
[End EMD]

[EMD Set] [EMD Set] keywords permitted within a .emd file and covered later
[Manufacturer]
[Description]
[EMD Model]
[End EMD Model]
[End EMD Set]

12.3 KEYWORD DEFINITIONS

Keyword: **[Begin EMD]**

Required: Yes

Description: Marks the beginning of an Electrical Module Description

Usage Rules: The keyword is followed by the name of the module. The length of the module name must not exceed 40 characters, and blank characters are allowed. There must be a matching [End EMD] keyword.

Other Notes: Only one [Begin EMD] keyword is permitted in a .emd file. This is different than the similar rules for .ibs, .pkg, and .ebd files.

Example:

```
[Begin EMD]  16X8_SIMM
```

Keyword: **[Manufacturer]**

Required: Yes

Description: Declares the manufacturer of the module enclosed within the [Begin EMD] / [End EMD] keywords that uses this .emd file.

Usage Rules: Following the keyword is the manufacturer's name. It must not exceed 40 characters and can include blank characters. Each manufacturer must use a consistent name in all .emd files.

Example:

```
[Manufacturer] NoName Corp.
```

Keyword: **[Description]**

Required: No

Description: Provides a concise explanation of what kind of interconnect the EMD represents.

Usage Rules: The text shall fit on a single line and may contain spaces.

Example:

```
[Description]   6-Pin Quad Ceramic Flat Pack
```

Keyword: **[Number Of EMD Pins]**

Required: Yes

Description: Defines the number of EMD pins, which shall match the number of pins found in the [EMD Pin List] keyword. EMD pins are any externally accessible electrical connection to the module.

Usage Rules: The field must be a positive integer. The [Number Of EMD Pins] keyword must be positioned before the [EMD Pin List] keyword.

Example:

```
[Number Of EMD Pins] 128
```

Keyword: **[EMD Pin List]**

Required: Yes

Description: Defines the pin names of the user accessible pins. It also defines which pins are connected to power and ground.

Sub-Params: signal_name, signal_type, bus_label

Usage Rules: The [EMD Pin List] keyword shall be followed by the subparameter names "signal_name", "signal_type", and "bus_label", serving as column headings. The keyword and the list of its subparameters shall be followed by as many rows of information as the number of EMD pins declared by the preceding [Number Of EMD Pins] keyword. Each row may contain up to four columns of information.

The first two columns are required on each row for each pin type.

The first column lists the pin name (in the data book this can also be called pin number). Each pin_name entry must be unique, i.e., duplicate pin names are not permitted. Pin names must be the alphanumeric external pin names of the module. The pin_name entry shall not exceed twenty (20) characters in length. All non-rail pins (generically referred to as I/O pins) are required to be listed.

The second column (signal_name) lists the name of the signal connected to that pin. The signal_name entries are not required to be unique for each row. Also, these signal_name entries may be different from the signal_names found under the designator .ibs [Component] or the designator .emd [Begin EMD] keywords. This allows the interchange of attached components or attached electrical module descriptions with standardized pin_name positions but with different manufacturer naming conventions. All EMD pins and designator pins that have the same signal_name are considered to be part of the same electrical net. The signal_name entry may also be used to signify the primary connection to other I/O pins (necessary for Aggressor_Only described later).

I/O pins shall consist of exactly two columns containing the pin_name and signal_name entries. No signal_type or bus_label entry is permitted for I/O pins.

The third column (signal_type) is required for rail pins and no-connect pins. The allowed values for this third column (as defined in Section 3.2, "SYNTAX RULES") are:

POWER	- reserved model name, used with power supply pins
GND	- reserved model name, used with ground pins
NC	- reserved model name, used with no-connect pins

"NC" is a legal signal_type and indicates that the pin is a "no-connect". As described in Section 3.2, "SYNTAX RULES" the reserved words "GND", "POWER", and "NC" are case-insensitive.

If a pin has a signal_type POWER, then all other pins with the same signal_name as this pin shall have signal_type POWER. If a pin has signal_type GND, then all other pins with the same signal_name as this pin shall have signal_type GND.

The fourth column (bus_label) is optional for rail pins (signal_type POWER or GND). The bus_label entry is a name assigned to a subset of the pins with a rail signal_name. As its name implies, bus_label entries are not required to be unique for each row. However, all pins that have the same bus_label must have the same signal_name. If the bus_label column is not specified for signal_type POWER or GND, then the bus_label shall be assumed to be the signal_name.

Multiple rail pins may be merged into a single [EMD Model] terminal using the terminal line syntax of the [EMD Model] keyword. This merged terminal combines all the rail pins with the same signal_name on the same interface, or all of the rail pins with the same bus_label on the same interface. In this case, all the pins that are merged into a single terminal are shorted.

Example:

```
|  A SIMM Module Example:
|
[Begin EMD]  16X8_SIMM
[Manufacturer] NoName Corp.

[Number Of EMD Pins]  6
[EMD Pin List]    signal_name   signal_type     bus_label
A1                GND           GND
A2                DQ1                                | I/O pin
A3                DQ2                                | I/O pin
A4                POWER5        POWER           Power5x
A5                RFU           NC
A6                POWER3.3      POWER
[End EMD Pin List]
```

Keyword: **[End EMD Pin List]**

Required: Yes

Description: Indicates the end of the data after [EMD Pin List].

Example:

```
[End EMD Pin List]
```

Keyword: **[EMD Parts]**

Required: No

Description: Maps an EMD part_name to an IBIS component or EMD module.

Usage Rules: The [EMD Parts] keyword shall be followed by a list of all the EMD parts (also called part numbers or part names in industry). Each EMD part_name entry in the list is followed by the file reference of the .ibs or .emd file containing the electrical description of the component or board, then the name of the component itself as given by the .ibs or .emd file's [Component] or [Begin EMD] keyword respectively. Official names of parts is recommended, but not required. The referenced .ibs or .emd files shall exist in the same directory as the calling .emd file or shall exist in a relative path under this directory.

A .emd file that describes a part can itself reference other EMD modules. No more than six levels of hierarchy for nested .emd files are permitted. A .emd file shall not reference itself directly or indirectly.

The EMD part_name entry, file reference, and component/module name terms are separated by whitespace. The EMD part_name entry is limited to forty characters.

A part_name entry shall be listed only once.

NAs in the file reference and component/module name columns are permitted if the part has functionality outside of the scope of the IBIS specification, such as certain analog parts. The NA in the file reference column indicates that the part model is not fully available. However, its designator shall be included under the [EMD Designator List] keyword, and its pinout shall be included as [Designator Pin List] keyword entries described below.

Other Notes: It is permitted to use a .ibs file or .emd file and a component or module name to show the part pinout and to document some known rails and digital I/O pins that are supported by the IBIS specification. Pins whose functions are not supported by the IBIS specification could be documented as "NC" pins or with Terminator models within these .ibs or .emd files.

A [Notes] section or a separate readme file should document these unknown parts or parts where certain pins cannot be modeled in IBIS. Some EDA tools may deal with these special cases in a tool-specific manner.

The [EMD Parts] keyword may be omitted if there are no EMD parts on the EMD module, such as in the case of a backplane or loopback board.

Example:

```
[EMD Parts]
|
| part_name       file_reference    component/module name
Processor         pp100.ibs         Processor
Memory_16X8       simm.emd          16X8_SIMM
74LS244           ls244.ibs         NoName_74LS244
Res_10K           r10K.ibs          My_10K_Pullup
|
ABC               NA                NA    | Undocumented Parts
BCD               NA                NA    | without files
|
C555              timer.ibs         X555  | Timer with digital control
|
[End EMD Parts]
```

Keyword: **[End EMD Parts]**

Required: Yes, if [EMD Parts] is present

Description: Indicates the end of the data after [EMD Parts].

Example:

```
[End EMD Parts]
```

Keyword: **[EMD Designator List]**

Required: Yes, if [EMD Parts] is present

Description: Maps an EMD designator to an IBIS or EMD part_name.

Usage Rules: The [EMD Designator List] keyword must be followed by a list of all the EMD designators (also called reference designators in industry). Each EMD designator is followed by a part_name.

For the context in this Electrical Module Description section, a "designator" shall be the first column in the data following [EMD Designator List].

The EMD designator and part_name are separated by whitespace.

The EMD designator is limited to ten characters. "*" is an illegal designator name.

Example:

```
[EMD Designator List]
|
| EMD Designator  part_name
u23               Processor
u24               Memory_16X8
u25               74LS244a
u26               Res_10K
[End EMD Designator List]
```

Keyword: **[End EMD Designator List]**

Required: Yes, if [EMD Designator List] is present

Description: Indicates the end of the data after [EMD Designator List].

Example:

```
[End EMD Designator List]
```

Keyword: **[Designator Pin List]**

Required: Yes, if [EMD Designator List] is present

Description: Defines the pin names of the designator pins. It also defines which designator pins are connected to power or ground. Designators are defined in the [EMD Designator List] section and can be instances of either a .ibs [Component] or a .emd [Begin EMD].

Sub-Params: signal_name, signal_type, bus_label

Usage Rules: The [Designator Pin List] keyword shall be followed by the subparameter names "signal_name", "signal_type", and "bus_label", serving as column headings. The keyword and the list of its subparameters shall be followed by rows of information containing all designator pins references by terminal lines found in all the EMD models contained by EMD Sets and referenced by all EMD Groups found in the .emd file. Designator pin names which are not referenced by any terminal line Qualifer_entry in any of the EMD models may optionally be listed under the [Designator Pin List] keyword.

The first two columns are required on each row for each pin type.

The first column must contain the alphanumeric external pin_names of the designator. The pin_name entry shall be preceded by the reference designator followed by a "." (e.g., U1.12). Each pin_name entry for a particular reference designator must be unique, i.e., duplicate pin names are not permitted. The pin_name entry (the portion after the reference designator and the ".") shall not exceed twenty (20) characters in length.

The second column (signal_name) lists the name of the signal connected to that pin. The signal_name entries are not required to be unique for each row. Also, these signal_name entries may be different from the signal_names found under the designator .ibs [Component] or the designator .emd [Begin EMD] keywords. This allows the interchange of attached components or attached electrical module descriptions with standardized pin_name positions but with different manufacturer naming conventions. All EMD pins and designator pins that have the same signal_name are considered to be part of the same electrical net.

I/O pin entries shall consist of exactly two columns containing the pin_name and signal_name entries. No signal_type or bus_label entry is permitted for I/O pins. The signal_name entry may also be used to signify the primary connection to other I/O pins (necessary for Aggressor_Only described later).

The third column (signal_type) is required for rail pins or for designator pins that have nothing connected to them from any of the EMD models. The allowed values for this third column (as defined in Section 3.2, "SYNTAX RULES") are:

POWER	- reserved model name, used with power supply pins
GND	- reserved model name, used with ground pins
NC	- reserved model name, used with no-connect pins

"NC" is a legal signal_type and indicates that the pin is a "no-connect". As described in Section 3.2, "SYNTAX RULES", the reserved words "GND", "POWER", and "NC" are case-insensitive.

If a pin has a signal_type POWER, then all other pins with the same signal_name as this pin shall have signal_type POWER. If a pin has signal_type GND, then all other pins with the same signal_name as this pin shall have signal_type GND. If a pin has signal_type NC, then the designator pin and signal_name shall not appear on any of the terminal lines of any EMD Model. The use of signal_type NC allows the model maker to document designator pins even if no connections are made to them from any EMD Model.

The fourth column (bus_label) is optional for rail pins (signal_type POWER or GND). The bus_label entry is a name assigned to a subset of the pins with a rail signal_name. As its name implies, bus_label entries are not required to be unique for each row. However, all pins that have the same bus_label must have the same signal_name within EMD files as well as within any IBIS [Component] referenced in [EMD Parts]. If the bus_label column is not specified for signal_type POWER or GND, then the bus_label shall be assumed to be the signal_name.

Multiple rail pins may be merged into a single [EMD Model] terminal using the terminal line syntax of the [EMD Model] keyword. This merged terminal combines all the rail pins with the same signal_name on the same interface, or all of the rail pins with the same bus_label on the same interface. In this case, all the pins that are merged into a single terminal are shorted.

Example:

```
| A SIMM Module Example:
```

```
|
[Begin EMD]  16X8_SIMM
[Manufacturer] NoName Corp.

[Number Of EMD Pins] 6
[EMD Pin List]  signal_name   signal_type bus_label
A1              VSS           GND
A2              DQ1                                | I/O pin
A3              DQ2                                | I/O pin
A4              VDD           POWER       VDD1
A5              VDD           POWER       VDD2
A6              VDDQ          POWER
[End EMD Pin List]

[Designator Pin List] signal_name signal_type bus_label
U1.11                 VSS         GND
U1.12                 DQ1                            | I/O pin
U1.13                 DQ2                            | I/O pin
U1.14                 VDD         POWER       VDD1
U2.21                 VDD         POWER       VDD2
U2.22                 DQ1                            | I/O pin
U2.23                 DQ2                            | I/O pin
U2.24                 VDDQ        POWER
[End Designator Pin List]
```

Keyword: **[End Designator Pin List]**

Required: Yes, if [Designator Pin List] is present

Description: Indicates the end of the data after [Designator Pin List].

Example:

```
[End Designator Pin List]
```

Keyword: **[Voltage List]**

Required: No

Description: Defines the voltage value for rail signal_names or bus_labels.

Usage Rules: Under the [Voltage List] keyword are four columns:

The first column lists a rail signal_name or bus_label found within EMD Pin List or Designator Pin List. Duplicate entries in the first column are prohibited.

The second column, V(typ), lists the typ value of the voltage. This entry is required.

The third column, V(min), lists the min (by magnitude) value of the voltage. If missing, 'NA' is entered, and the default value is V(typ).

The fourth column, V(max), lists the max (by magnitude) value of the voltage. If missing, 'NA' is entered, and the default value is V(typ).

It is not required to list under [Voltage List] all rail signal_names or bus_labels found within the EMD Pin List or Designator Pin List keywords.

Other Notes: This keyword can be used in several ways:

- Provides information about expected voltage source values at EMD Pin List and Designator Pin List interfaces for any or all the rail signal_names or bus_labels. The EDA tool can override these values. This might occur in the following cases:
  - With a SPICE netlist that provides its own sources
  - If V(min) and V(max) values are not supplied (as might occur with a SPICE netlist and its sources)
  - With [Model] corner setting using the typ, min, and max sources that are declared within the [Model] keyword
- Declares external sources at the EMD Pin List and/or Designator Pin List interfaces for the named voltages.

Since the [Voltage List] entries may be incomplete, or V(min) and/or V(max) values may be omitted, combinations of the above options are permitted.

In simulation, [Voltage List] entries shall be selected along with the corresponding corner values in [Model] entries. That is, V(typ) values should be used with typ corner conditions, V(min) with min corner conditions, and V(max) with max corner conditions.

In a power aware simulation, voltages will be supplied by the EDA tool at the EMD pins from voltage sources in the board or module that uses the EMD.

Example:

```
[Voltage List]
| V(name) V(typ) V(min) V(max)
VSS       0.0    0.0    0.0
VDD       1.2    1.1    1.3
[End Voltage List]
```

Keyword: [**End Voltage List**]

Required: Yes

Description: Indicates the end of the data after [Voltage List].

Example:

```
[End Voltage List]
```

Keyword: **[EMD Group]**

Required: Yes

Description: [EMD Group] has a single argument, which is the name of the associated EMD Group. The length of the EMD Group name shall not exceed 40 characters. Blank characters are not allowed. The [EMD Group]/[End EMD Group] keyword pair is hierarchically scoped by the [Begin EMD] keyword. The [EMD Group] keyword is used to define a list of [EMD Set]s by name that shall be used together to define EMD Models to be used in a simulation. A simulation may contain EMD Models from the EMD Sets listed in only one EMD Group.

Usage Rules: [Begin EMD] must contain one or more [EMD Group] keywords (identified by a name). Each [EMD Group] must contain at least one [EMD Set] name. EMD Sets contain EMD Models used to describe EMD pin or IBIS designator pin connections to IBIS-ISS subcircuits or n-port networks described by Touchstone files.

EMD Sets that exist for the module shall be listed in one or more EMD Groups. An EMD Group is required even if it references only one EMD Set.

The section under the [EMD Group] keyword shall have two entries per line, with each line identifying one EMD Set associated with the module. The entries shall be separated by at least one whitespace character. The first entry lists the EMD Set name (up to 40 characters long). The second entry is the file reference of the file containing the EMD Set and shall have the extension "ems". This file reference shall conform to the rules given in Section 3, "GENERAL SYNTAX RULES AND GUIDELINES". If the EMD Set is in the same .emd file as [Begin EMD], then the second entry shall be "NA".

The files containing the EMD Sets with the "ems" extension shall be located in the same directory as the .emd file or in a specified directory under the .emd file as determined by the directory path according to the file name rules given in Section 3, "GENERAL SYNTAX RULES AND GUIDELINES" (i.e., a file reference containing a relative path to a directory below that of the referencing .emd file is permitted). An EMD Set with matching name shall be found in the stated location for each EMD Set named in the [EMD Group] keyword.

Each EMD Set name and its file_reference may only appear once under each [EMD Group] keyword for a given designator.

Refer to Section 13.6, "CONNECTION RULES FOR EMD GROUP, EMD SET, AND EMD MODEL" for connection rules and limitations on the permissible EMD Set links under each [EMD Group] keyword and after some more terms and rules related to [EMD Set] and [EMD Model] keywords are defined.

Examples:

```
| Example 1
|
[EMD Group]             Full_ISS_PDN_1
| EMD Set               file_reference
Full_ISS_PDN_1          NA                  | The [EMD Set] is
                                            | present in the .emd file for
                                            | all pins

[End EMD Group]
|
| Example 2
|
[EMD Group]             Full_ISS_PDN_sn_2
| EMD Set               file_reference
Full_ISS_PDN_sn_2       NA                  | The [EMD Set] is
                                            | present in the .emd file for
                                            | all I/O pins and PDN
[End EMD Group]
```

Keyword: **[End EMD Group]**

Required: Yes, for each instance of the [EMD Group] keyword

Description: Indicates the end of the data for one [EMD Group].

Example:

```
[End EMD Group]
```

Keyword: **[End EMD]**

Required: Yes

Description: Marks the end of an electrical module description.

Usage Rules: This keyword shall be placed at the end of each complete electrical module description.

Example:

```
[End EMD]
```

13 EMD SET AND EMD MODEL DESCRIPTION

13.1 EMD SET KEYWORD DESCRIPTION

Keyword: **[EMD Set]**

Required: Yes

Description: Used to contain EMD Models

Usage Rules: [EMD Set] has a single argument, which is the name of the EMD Set. The length of the EMD Set name shall not exceed 40 characters. Blank characters are not allowed. The [EMD Set]/[End EMD Set] keyword pair is hierarchically equivalent in scope to [Begin EMD].

The section under the [EMD Set] keyword may contain a [Manufacturer] keyword section and [Description] keyword section and shall contain one or more EMD Models. See the section [EMD Model] for a description of the content of each EMD Model.

An EMD Set contains a list of EMD Models that have a logical association such as:

- All signals in a bus (e.g., DDR4, or PCI Express)
- Full Power Delivery Network (PDN) structures from EMD pins to designator pins
- Full PDN structures from EMD pins to EMD pins
- All I/O structures between EMD pins and designator pins
- I/O structures from designator pins to designator pins
- Combinations of I/O and PDN structures
- Coupled models
- Touchstone electrical models
- Decoupling capacitor models
- IBIS-ISS electrical models

Example:

```
[EMD Set] Signal_Integrity
[Manufacturer] NoName Corp.
[Description] This set contains one model for each I/O buffer
[EMD Model] DQ1
…
[End EMD Model]

[EMD Model] DQ2
…
[End EMD Model]

[EMD Model] DQS
…
[End EMD Model]
[End EMD Set]
```

Keyword: **[Manufacturer]**

Required: No

Description: Declares the manufacturer of the module described within the enclosing [EMD Set] found in the.emd or .ems file.

Usage Rules: Following the keyword is the manufacturer's name. It must not exceed 40 characters and can include blank characters. Each manufacturer must use a consistent name in all .emd files.

Example:

```
[Manufacturer] NoName Corp.
```

Keyword: **[Description]**

Required: No

Description: Provides a concise yet easily human-readable description of what kind of interconnect the [EMD Set] represents.

Usage Rules: The description shall fit on a single line and may contain spaces.

Example:

```
[Description]   6-Pin Quad Ceramic Flat Pack
```

Keyword: [**End EMD Set**]

Required: Yes, for each instance of the [EMD Set] keyword.

Description: Indicates the end of the EMD Set data.

Example:

```
[End EMD Set]
```

13.2 GENERAL EMD SET AND EMD MODEL FILE SYNTAX REQUIREMENTS

One or more EMD Sets may be included in a separate EMD Set file, using a file name with the extension "ems", or within the .emd file. The [EMD Set] keyword can contain the optional [Manufacturer] and [Description] keywords and one or more [EMD Model] keywords and the [EMD Model] associated subparameters, as listed in Table 50.

Table 50 – EMD Set and EMD Model Keywords and Subparameters

Keyword or Subparameter	Notes
[EMD Set]	
[Manufacturer]	(note 1)

Keyword or Subparameter	Notes
[Description]	(note 1)
[EMD Model]	(note 2)
Param	
File_TS	(note 3)
File_IBIS-ISS	(note 3)
Unused_port_termination	(note 4)
Number_of_terminals	(note 5)
<terminal line>	(note 6)
[End EMD Model]	(note 7)
[End EMD Set]	(note 8)

Note 1 [Manufacturer] and [Description] are each optional keywords within any [EMD Set].

Note 2 At least one [EMD Model] is required for each [EMD Set].

Note 3 One of either the File_TS or File_IBIS-ISS subparameters is required.

Note 4 Required for Touchstone files where ports are unused, illegal if there are no unused ports or for IBIS-ISS files.

Note 5 This subparameter shall be followed by the "=" character and an integer value, with both optionally surrounded by whitespace.

Note 6 See Section 13.3 below.

Note 7 Required when the [EMD Model] keyword is used.

Note 8 Required when the [EMD Set] keyword is used.

When EMD Set definitions occur within a .emd file, their scope is "local"— they are known only within that .emd file and no other .emd file.

Usage Rules for the .ems file:

EMD Models are stored in a file whose file name uses the format:

<stem>.ems

The <stem> provided shall adhere to the rules given for the [File Name] keyword. Use the "ems" extension to identify files containing EMD Models. The .ems file shall contain the [IBIS Ver], [File Name], [File Rev], and the [End] keywords. Optional elements include the [Date], [Source], [Notes], [Disclaimer], [Copyright], and [Comment Char] keywords. All these keywords and associated subparameters follow the same rules as those for a normal .ibs file.

13.3 GENERAL EMD MODEL KEYWORD DESCRIPTION

Keyword: **[EMD Model]**

Required: Yes

Description: Marks the beginning of the definition of the electrical model of the interconnect between the external pin(s) of the module and the pin(s) of the designator(s) in the module.

Sub-Params: Unused_port_termination, Param, File_TS, File_IBIS-ISS, Number_of_terminals

Usage Rules: [EMD Model] has a single argument, which is the name of the associated EMD Model. The length of the EMD Model name shall not exceed 40 characters. Blank characters are not allowed. The [EMD Model]/[End EMD Model] keyword pair is hierarchically scoped by the [EMD Set]/[End EMD Set] keywords.

The [EMD Model]/[End EMD Model] section defines both the association between a Touchstone file or IBIS-ISS subcircuit and an EMD Model, as well as defining the terminals and terminal usage for the EMD Model in the context of the given [Begin EMD].

An [EMD Model] may contain terminals from one or more interfaces including those listed in the [EMD Pin List] and/or those listed in the [Designator Pin List].

An [EMD Model] may contain terminals in combinations such as:

- one or more rails only
- one or more I/O signals
- one or more rails and one or more I/O signals
- one or more rails at the EMD Pin List interface only
- one or more rails at the Designator Pin List interface only

The following subparameters are defined:

Param
File_IBIS-ISS
File_TS
Unused_port_termination
Number_of_terminals = <value>

In addition to these subparameters, the [EMD Model]/[End EMD Model] section may contain lines describing terminals and their connections. No specific subparameter name or other string is used to identify terminal lines.

Unless noted below, no EMD Model subparameter requires the presence of any other subparameter.

Param rules:

The subparameter Param is optional and only legal with the File_IBIS-ISS subparameter documented below. Param is illegal with the File_TS subparameter documented below. Param shall be followed by three arguments: an unquoted string argument giving the name of the parameter to be passed into the IBIS-ISS subcircuit, a reserved word for the parameter format, and one numerical value or one string value (surrounded by double quotes) for the parameter value to be passed into the IBIS-ISS subcircuit.

The numerical value rules follow the scaling conventions in Section 3.2, “SYNTAX RULES”. The EDA tool is responsible for translating IBIS specified parameters into IBIS-ISS parameters. For example, 1 megaohm, would be represented as 1M in Param value according to the Section 3 rules, but would be converted by the EDA tool to case-insensitive 1meg (1X is not recommended) or 1E6 for IBIS-ISS use. Quoted string parameters in IBIS are converted to the string parameter syntax in IBIS-ISS subcircuits. For example, the Param value “typ.s2p” would be converted to str('typ.s2p') in IBIS-ISS subcircuits.

Examples:

```
| Param   name     format   value
Param     abc      Value    2m         | 2E-3 in IBIS
Param     def      Value    4k         | 4E3 in IBIS
Param     ts_file  Value    "typ.s2p"  | file name string passed
                                       | into IBIS-ISS
```

File_IBIS-ISS rules:

Either File_IBIS-ISS or File_TS is required for a [EMD Model]/[End EMD Model] group. The File_IBIS-ISS subparameter is followed by two unquoted string arguments consisting of the file_reference and circuit_name (.subckt name) for an IBIS-ISS file. The IBIS-ISS file under file_reference shall be located in the same directory as the referencing .emd file or .ems file or in a specified directory under the referencing file as determined by the directory path (i.e., a file reference containing a relative path to a directory below that of the referencing .emd or .ems file is permitted).

Example:

```
| file_type    file_reference    circuit_name(.subckt name)
File_IBIS-ISS  net.iss           netlist_typ
```

File_TS rules:

Either File_TS or File_IBIS-ISS is required for a [EMD Model]/[End EMD Model] group. File_TS is followed by one unquoted string argument, which is the file_reference for a Touchstone file. The Touchstone file under file_reference shall be located in the same directory as the referencing .emd file or .ems file or in a specified directory under the referencing file as determined by the directory path (i.e., a file reference containing a relative path to a directory below that of the referencing .emd or .ems file is permitted).

Example:

```
| file_type    file_reference
File_TS        typ.s8p
```

Unused_port_termination rules:

The Unused_port_termination subparameter is required under this condition:

File_TS is used and the number of terminal lines (described below) is less than N+1 (where N is the number of ports in the Touchstone file)

Unused_port_termination is illegal under these conditions:

File_IBIS-ISS is used

File_TS is used, and the number of terminal lines is N+1

If required, only one Unused_port_termination subparameter may appear for a given [EMD Model] keyword.

The Unused_port_termination subparameter is followed by whitespace and one of these arguments:

Open
Reference
Resistance

"Open" declares that the unused ports remain unterminated (open-circuited).

"Reference" declares that the EDA tool terminates all unused ports with resistors whose resistance values are equal to the reference impedances provided in the Touchstone file for the respective unused ports, and all connected to the model's reference terminal.

"Resistance" declares that the EDA tool terminates all unused ports with resistors, all having the same value, and all connected to the model's reference terminal. The "Resistance" entry is followed by a third column entry with the (non-negative) numerical resistance value.

Examples:

```
Unused_port_termination       Open
Unused_port_termination       Reference
Unused_port_termination       Resistance  43.5
```

Number_of_terminals rules:

The Number_of_terminals subparameter is required and defines the number of terminals associated with the EMD Model. The subparameter name shall be followed by a single integer argument on the same line. The argument shall be separated from the subparameter name by the "=" character. The subparameter name, "=" character, and argument may optionally be separated by whitespace.

Only one Number_of_terminals subparameter may appear for a given [EMD Model] keyword. The Number_of_terminals subparameter shall appear before any terminal lines and after all other subparameters for a given EMD Model.

For File_IBIS-ISS, the Number_of_terminals value shall be equal to the number of subcircuit terminals for an IBIS-ISS subcircuit. Because an IBIS-ISS subcircuit requires at least one terminal the Number_of_terminals value shall be 1 or greater. The IBIS-ISS subcircuit terminals shall not contain an ideal reference node (SPICE node 0 or its synonyms).

For File_TS, the Number_of_terminals value shall be a value equal to N+1 (where N is the number of ports in the Touchstone file). Because a Touchstone file requires at least one port, the Number_of_terminals value shall be 2 or greater.

Example:

```
Number_of_terminals = 3
```

Terminal line rules:

The terminal lines shall appear after the Number_of_terminals subparameter and before the [End EMD Model] keyword.

Terminal lines are of the following form, with each identifier separated by whitespace:

<Terminal_number> <Terminal_type> <Terminal_type_qualifier> <Qualifier_entry> [Aggressor_Only]

Terminal_number
The Terminal_number is the identifier for a specific terminal. The value shall be 1 or greater and less than or equal to the Number_of_terminals. The same Terminal_number shall not appear more than once for a given EMD Model.

For File_IBIS-ISS, the Terminal_number entry shall match the IBIS-ISS terminal (node) position. The Terminal_number entries may be listed in any order as long as there are no duplicate entries. Each IBIS-ISS terminal shall have a terminal line entry.

For File_TS, the Terminal_number entry shall match the Touchstone file port number or reference terminal line, as shown below. The Terminal_number entries may be listed in any order as long as there are no duplicate entries. The terminal line for Terminal_number N+1 is required as a reference terminal for each port and shall be connected to a rail terminal or A_gnd in the EMD Model. At least one other terminal line entry is required.

- Terminal_number Port
- 1 1
- 2 2
- …
- N N
- N+1 Reference terminal for the Touchstone file

For Touchstone files, each unused port and its corresponding Terminal_number shall be terminated in simulation with a resistor whose value corresponds to the Unused_port_termination subparameter entry. The resistor is connected to the model's reference terminal.

Terminal_type
The Terminal_type is a string that identifies whether the terminal is a reference, supply or I/O terminal and whether the terminal is connected to an EMD pin or designator pin. Note that "I/O" in this context is a synonym for "signal", as opposed to "supply" or "rail"; it is not intended to imply model type as used in the "Model_type" subparameter.

Terminal_type A_gnd defines a connection to the simulator global reference node. The A_gnd node can be used at any interface.

Terminal_type A_gnd is not required under File_TS or File_IBIS-ISS.

If present under File_TS, Terminal_type A_gnd may be used only once. If used, it may only appear on the terminal line with Terminal_number N+1.

If present under File_IBIS-ISS, Terminal_type A_gnd may be used on any number of terminal lines.

Furthermore, if the terminal is connected to a buffer supply rail, the Terminal_type identifies to which specific buffer rail the terminal is connected. The Terminal_type shall be one of the following:

- Pin_I/O
- Pin_Rail
- A_gnd

Terminal_type_qualifier
Terminal_type_qualifier is a string that identifies the association between a terminal and a specific pin_name, signal_name or bus_label in the [EMD Pin List], or specific pin_name, signal_name, or bus_label in the [Designator Pin List].

Qualifier_entry
The <Qualifier_entry>, shown in angle brackets, is the name required for the following Terminal_type_qualifiers:

pin_name <pin_name_entry>

signal_name <signal_name_entry>

bus_label <bus_label_entry>

Designator pin names or designator signal names whose signal_type is "NC" are prohibited to appear as a <pin_name_entry>.

Aggressor_Only
The Aggressor_Only entry is optional and is indicated by the string "Aggressor_Only" without the quotation marks. Assigning Aggressor_Only to a pin assigns the Aggressor_Only properties to all pins of the same signal_name listed in the [EMD Pin List] and [Designator Pin List] keywords.

Any *_I/O Terminal_type without the Aggressor_Only column may be considered an aggressor or a victim.

Multi-line EMD Models may describe only a subset of a coupled structure (e.g., a 64-line bus may be described by a four-line EMD Model). As a result, while the interconnects at the edges of the EMD Model may induce crosstalk onto other interconnects nearby, being on the edge of the EMD Model, they may not themselves experience the full crosstalk impact that the corresponding interconnect experiences in the real, full structure.

Crosstalk simulations use coupled interconnect models consisting of nets, or extended nets that may span packages, EMDs, boards, and connectors. If any terminal in any net or extended net in the coupled interconnect model is marked Aggressor_Only, then the crosstalk contributions included in the simulation results reported for this net or extended net will be incomplete.

13.4 TERMINAL_TYPE ASSOCIATIONS FOR EMD AND DESIGNATOR PINS

Terminal lines describe the IBIS-ISS terminal or Touchstone port to which each terminal of an EMD Model is connected. The arrangement of the terminal line entries (columns) is described below.

- The first column, Terminal_number, contains an integer between 1 and the Number_of_terminals that describes the ordinal (positional) number of the IBIS-ISS node in the EMD Model subcircuit or Touchstone file port. The second column is Terminal_type, the third column is Terminal_type_qualifier, the fourth column is Qualifier_entry, and there is an optional fifth column "Aggressor_Only".
- The second column, Terminal_type is:
    - For I/O connections
        - Terminal_type must be Pin_I/O
        - Terminal_type_qualifier shall be pin_name
            - EMD Pins shall be a pin_name in the [EMD Pin List] list
            - Designator Pins shall be in the form from the [Designator Pin List]:
                - <designator>.< pin_name>
    - For rail connections
        - Terminal_type shall be Pin_Rail
        - Terminal_type_qualifier shall be one of the following:
            - pin_name
                - Qualifier_entry shall be a rail pin_name in the [EMD Pin List] of the form <designator>.<pin_name> entry from the or [Designator Pin List] and with signal_type POWER or GND
            - signal_name
                - Qualifier_entry shall be a rail signal_name in the [EMD Pin List] or of the form <designator>.<signal_name> entry from the [Designator Pin List]
                - For [Designator Pin List] entries, the signal_name values can be assigned so that they can be connected with the same signal_name entries on the [EMD Pin List]. The signal_name entries do not have to match those referenced under the [Component] or [Begin EMD] keywords.
                - *.<signal_name> shall represent all of the [Designator Pin List] <signal_name> entries at all [Designator Pin List] interfaces shorted together.
            - bus_label
                - Qualifier_entry shall be a bus_label in the [EMD Pin List] or of the form <designator>.<bus_label> entry from the [Designator Pin List]
                - The bus_label entry can be assigned to both the [EMD Pin List] and [Designator Pin List] entries to support a subset of

connections that might be connected with a common signal_name. For example, left-side routing and right-side routing might be isolated from each other.

- *.<bus_label> shall represent all of the [Designator Pin List] <bus_label> entries at all [Designator Pin List] interfaces shorted together.

- At any interface
  - Terminal_type A_gnd is available at any interface and without any Terminal_type qualifier

Table 51 summarizes the rules described above and applies to terminals associated with the [EMD Pin List] keyword and with the [Designator Pin List] keyword.

Table 51 – Allowed Terminal_type Associations for EMD Models[1]

Terminal_type	Terminal_type_qualifier			Aggressor_Only
	pin_name	signal_name	bus_label	
Pin_I/O	X			A
Pin_Rail	Y	Y	Y	
Pin_Rail		*.Y[2]	*.Y[2]	
A_gnd				

Notes:

1) In the table, "X" refers to I/O pin names. "Y" indicates POWER and GND terminals. The letter "A" designates "Aggressor_Only".
2) "*.Y" indicates that all of the "Y" named POWER and GND terminals on each of the [Designator Pin List] interfaces are shorted together

There are at least three kinds of connectivity that can relate signal_names, bus_labels and/or terminals. These are described below.

For Rail terminals:

On one interface, terminals with the same signal_name may be reduced to a single terminal for modeling purposes with the syntax:

<Terminal_number> Pin_rail signal_name <signal_name> or
<Terminal_number> Pin_rail signal_name <designator>.<signal_name>

On one interface, terminals with the same bus_label may be reduced to a single terminal for modeling purposes with the syntax:

<Terminal_number> Pin_rail bus_label <bus_label> or
<Terminal_number> Pin_rail bus_label <designator>.<bus_label>

Electrical connections could exist between individual pin_names, but these rail pins are modeled as if they are connected by shorts and are merged into one terminal.

For designator interfaces only, involving rails:

For *all* designator interfaces, terminals with the same signal_name may be reduced to a single terminal for modeling purposes with the syntax:

<Terminal_number> Pin_rail signal_name <*.signal_name>

For *all* designator interfaces, terminals with the same bus_label may be reduced to a single terminal for modeling purposes with the syntax:

<Terminal_number> Pin_rail bus_label <*.bus_label>

This syntax excludes rail terminals at the [EMD Pin List] interface. There may exist electrical connections between all of the *.<name> terminals. The connections are not necessarily physical shorts on any one interface or between any of the interfaces.

Multiple applications exist for EMD Models focused on rail terminals. For example, an EMD Model with only rail terminals and two interfaces (no I/O terminals) can be used for a PDN (note that a PDN structure can exist in an EMD Model with I/O terminals). Also, an EMD Model with only rail terminals (no I/O terminals) and only one interface is permitted for applications such as for modeling rail decoupling circuits.

For I/O terminals:

Terminals at the same interface or at any designator interface that have the same signal_name are considered “connected” in the same electrical net (named by the signal_name entry). The terminals need to be documented in the [EMD Model] keyword and their electrical connections are described by IBIS-ISS or Touchstone data. Connections between these terminals are usually NOT shorts. The common signal_name provides for a way to document net name connection between different components or modules at terminals that may have different pin_names. For example:

```
1 Pin_I/O  pin_name  A1     | signal_name is DQ0
2 Pin_I/O  pin_name  U1.25  | signal_name is DQ0
3 Pin_I/O  pin_name  U2.32  | signal_name is DQ0
4 Pin_I/O  pin_name  U3.32  | signal_name is DQ0
```

The common signal_name in the [EMD Pin List] and/or [Designator Pin List] indicates that the four terminals are in the same net. Their electrical “connections” are described by the electrical content in the IBIS-ISS or Touchstone file data connected to terminals 1, 2, 3, and 4.

For I/O terminals with extended nets:

The EDA tool may establish connections between nets of different names across an IBIS series component using extended nets.

13.5 RDIMM EXAMPLE ILLUSTRATING SYNTAX AND NET OPTIONS

13.5.1 RDIMM FIGURES FOR EXAMPLES IN 13.5.2 THROUGH 13.5.4

Figure 55 shows a DDR4 Registered DIMM containing DRAM components labeled by designators U1, U2, U4, U5 (front side) and U7-U11 (back side, not seen) and a Register component labeled by designator U3.

Also shown is pre-register net A07 connecting from an EMD pin to a designator pin of designator U3 and post-register net BA07 connecting from a designator pin of designator U3 to designator pins of designators U4, U5, U7, and U8 as well as termination resistor RN13 connecting to the VTT rail.

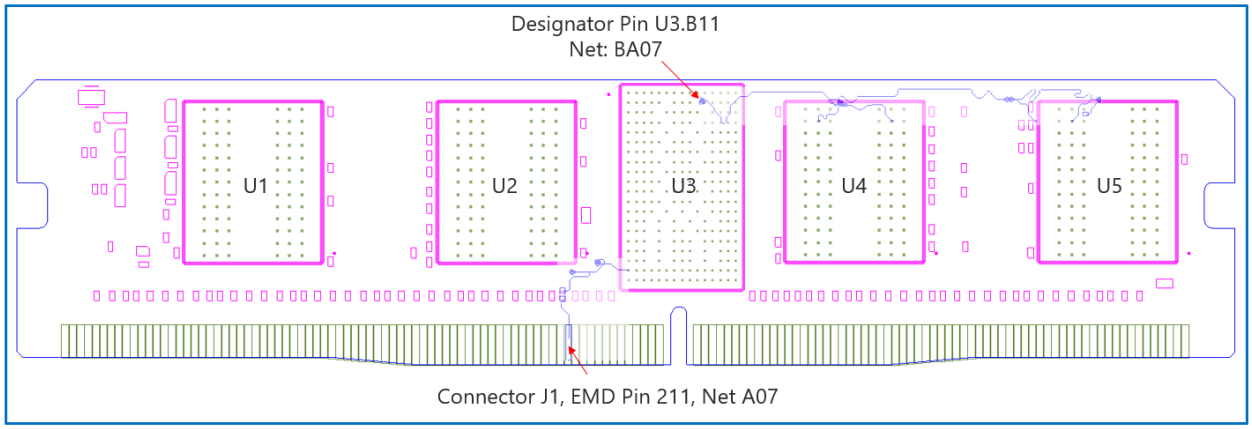


Figure 55 – DDR4 Registered DIMM with Labeling

Figure 56 (Example 1), a zoomed in area of Figure 55, shows an example of an extended net. The extended net A07 can be modeled in two ways:

1. One EMD Model defining only terminals for EMD pin 211 and designator pin U3.W1. The EMD Model contains the complete signal path of net A07, the series resistor R123, and net A07r (referenced in the A07.iss file with the subcircuit named “A07_1”, as shown in Example 1).
2. One EMD Model or multiple EMD Models contained with an EMD Set that include terminals for EMD pin 211 and designator pin U3.W1 and two terminals for the pins of the series resistor. The resistor would be assigned a designator (R123) referencing an IBIS component (see Examples 2, 3). The connection between net A07 and net A07r through R123 might be determined automatically in some EDA tools or entered manually. Alternatively, net A07 and net A07r can be treated as two independent nets.

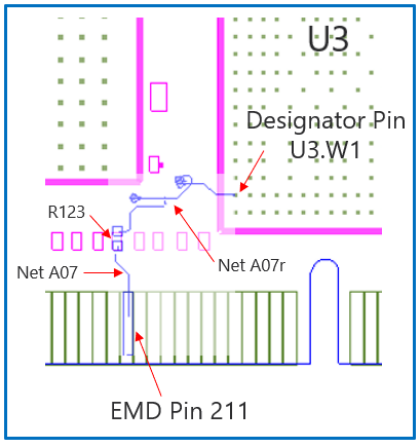


Figure 56 – Extended Net

Figure 57 (Examples 2, 3), a zoomed in area of Figure 55, shows an example of an internal net. The post-register net BA07 connects from the register's designator pin U3.B11 to the DDR4 DRAMs' designator pins U4.M8, U5.M8, U7.M8, and U8.M8 as well as to one designator pin of the termination resistor RN13. RN13 terminates the signal to the VTT rail.

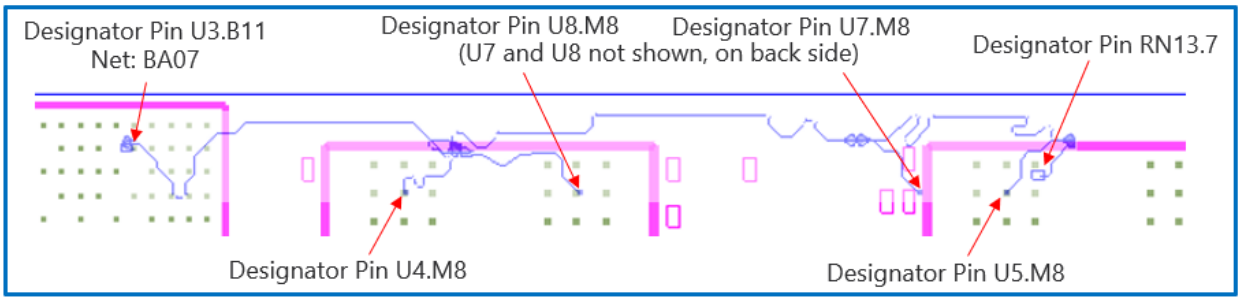


Figure 57 – Internal Net

13.5.2 EXAMPLE 1 (R123 AND RN13 EMBEDDED IN A07_1 AND BA07_1)

```
|*****************************************************************************
| EMD Syntax Example 1 (Net A07 with Embedded Resistors)
| Using DDR4 RDIMM Example

[Begin EMD] DDR4_RDIMM_1
[Number of EMD Pins] 4
[EMD Pin List] signal_name signal_type  bus_label
203            VSS         GND
211            A07
212            VDD         POWER        VDD1
223            VTT         POWER
[End EMD Pin List]
```

```
[EMD Parts]
DDR4_Reg_253b  register.ibs   DDR4_Register
DDR4_x8_78b    dram.ibs       DDR4_8Gb_x8
[End EMD Parts]

[EMD Designator List]
U3        DDR4_Reg_253b
U4        DDR4_x8_78b
U5        DDR4_x8_78b
U7        DDR4_x8_78b
U8        DDR4_x8_78b
[End EMD Designator List]

[Designator Pin List] signal_name  signal_type  bus_label
U3.B9                 VDD          POWER        VDD1
U3.B11                BA07
U3.B12                VSS          GND
U3.V3                 VDD          POWER        VDD1
U3.W1                 A07
U3.W3                 VSS          GND
|
U4.K9                 VSS          GND
U4.M8                 BA07
U4.N9                 VDD          POWER        VDD1
|
U5.K9                 VSS          GND
U5.M8                 BA07
U5.N9                 VDD          POWER        VDD1
|
U7.K9                 VSS          GND
U7.M8                 BA07
U7.N9                 VDD          POWER        VDD1
|
U8.K9                 VSS          GND
U8.M8                 BA07
U8.N9                 VDD          POWER        VDD1
[End Designator Pin List]

[Voltage List]
VDD   1.200   1.140   1.260
VSS   0.000   0.000   0.000
VTT   0.600   0.570   0.630
[End Voltage List]

[EMD Group]    Addr_07_Group_1
Addr_07_1  NA
[End EMD Group]
[End EMD]

[EMD Set]      Addr_07_1
[Manufacturer]  NoName Corp.
[EMD Model]     A07_1
File_IBIS-ISS   A07.iss       A07_1
Number_of_terminals = 6
1  Pin_I/O      pin_name      211
2  Pin_I/O      pin_name      U3.W1  | Connection from 211 to U3.W1 includes
```

```
                                        | Series Resistor modeled in A07.iss A07_1
3  Pin_Rail     bus_label     VDD1
4  Pin_Rail     signal_name   VSS
5  Pin_Rail     bus_label     U3.VDD1
6  Pin_Rail     bus_label     U3.VSS
[End EMD Model]

[EMD Model]     BA07_1
File_IBIS-ISS   A07.iss       BA07_1
Number_of_terminals = 18
1  Pin_I/O      pin_name      U3.B11
2  Pin_Rail     bus_label     U3.VDD1
3  Pin_Rail     signal_name   U3.VSS
4  Pin_I/O      pin_name      U4.M8
5  Pin_Rail     bus_label     U4.VDD1
6  Pin_Rail     signal_name   U4.VSS
7  Pin_I/O      pin_name      U5.M8
8  Pin_Rail     bus_label     U5.VDD1
9  Pin_Rail     signal_name   U5.VSS
10 Pin_I/O      pin_name      U7.M8
11 Pin_Rail     bus_label     U7.VDD1
12 Pin_Rail     signal_name   U7.VSS
13 Pin_I/O      pin_name      U8.M8    | Termination Resistor to VTT
                                       | included in A07.iss BA07_1
14 Pin_Rail     bus_label     U8.VDD1
15 Pin_Rail     signal_name   U8.VSS
16 Pin_Rail     bus_label     VDD1
17 Pin_Rail     signal_name   VTT
18 Pin_Rail     signal_name   VSS
[End EMD Model]
[End EMD Set]
|*****************************************************************************
```

13.5.3 EXAMPLE 2 (R123 AND RN13 MODELED AS SEPARATE IBIS COMPONENTS IN A07_2, BA07_2)

```
|*****************************************************************************
| EMD Syntax Example 2 (External Resistors)
| Using DDR4 RDIMM Example

[Begin EMD] DDR4_RDIMM_2
[Number of EMD Pins] 4
[EMD Pin List] signal_name signal_type  bus_label
203            VSS         GND
211            A07                          |Net A07 Connection
212            VDD         POWER        VDD1
223            VTT         POWER
[End EMD Pin List]

[EMD Parts]
DDR4_Reg_253b  register.ibs         DDR4_Register
DDR4_x8_78b    dram.ibs             DDR4_8Gb_x8
510-500874     resistors.ibs        RES_22ohms
```

```
510-501618     resistors.ibs        RPACK4_33ohms
[End EMD Parts]

[EMD Designator List]
U3         DDR4_Reg_253b
U4         DDR4_x8_78b
U5         DDR4_x8_78b
U7         DDR4_x8_78b
U8         DDR4_x8_78b
R123       510-500874
RN13       510-501618
[End EMD Designator List]

[Designator Pin List] signal_name  signal_type  bus_label
U3.B9                 VDD          POWER        VDD1
U3.B11                BA07
U3.B12                VSS          GND
U3.V3                 VDD          POWER        VDD1
U3.W1                 A07r                 | Net A07r Terminal
U3.W3                 VSS          GND
|
U4.K9                 VSS          GND
U4.M8                 BA07
U4.N9                 VDD          POWER        VDD1
|
U5.K9                 VSS          GND
U5.M8                 BA07
U5.N9                 VDD          POWER        VDD1
|
U7.K9                 VSS          GND
U7.M8                 BA07
U7.N9                 VDD          POWER        VDD1
|
U8.K9                 VSS          GND
U8.M8                 BA07
U8.N9                 VDD          POWER        VDD1
|
R123.1                A07                 | Net A07 Terminal
R123.2                A07r                | Net A07r Terminal
RN13.2                VTT          POWER
RN13.7                BA07
[End Designator Pin List]

[Voltage List]
VDD   1.200   1.140   1.260
VSS   0.000   0.000   0.000
VTT   0.600   0.570   0.630
[End Voltage List]

[EMD Group]    Addr_07_Group_2
Addr_07_2  NA
[End EMD Group]
[End EMD]

[EMD Set]      Addr_07_2
[Manufacturer]  NoName Corp.
```

```
[EMD Model]     A07_2
File_IBIS-ISS   A07.iss       A07_2
Number_of_terminals = 8
1  Pin_I/O      pin_name      211     | Net A07 Terminal and Connection
2  Pin_I/O      pin_name      R123.1  | Net A07 Terminal and Connection
3  Pin_I/O      pin_name      R123.2  | Net A07r Terminal and Connection
4  Pin_I/O      pin_name      U3.W1   | Net A07r Terminal and Connection
5  Pin_Rail     bus_label     VDD1
6  Pin_Rail     signal_name   VSS
7  Pin_Rail     bus_label     U3.VDD1
8  Pin_Rail     signal_name   U3.VSS
[End EMD Model]

[EMD Model]     BA07_2
File_IBIS-ISS   A07.iss       BA07_2
Number_of_terminals = 19
1  Pin_I/O      pin_name      U3.B11
2  Pin_Rail     bus_label     U3.VDD1
3  Pin_Rail     signal_name   U3.VSS
4  Pin_I/O      pin_name      U4.M8
5  Pin_Rail     bus_label     U4.VDD1
6  Pin_Rail     signal_name   U4.VSS
7  Pin_I/O      pin_name      U5.M8
8  Pin_Rail     bus_label     U5.VDD1
9  Pin_Rail     signal_name   U5.VSS
10 Pin_I/O      pin_name      U7.M8
11 Pin_Rail     bus_label     U7.VDD1
12 Pin_Rail     signal_name   U7.VSS
13 Pin_I/O      pin_name      U8.M8
14 Pin_Rail     bus_label     U8.VDD1
15 Pin_Rail     signal_name   U8.VSS
16 Pin_I/O      pin_name      RN13.7
17 Pin_Rail     bus_label     VDD1
18 Pin_Rail     signal_name   RN13.VTT
19 Pin_Rail     signal_name   VSS
[End EMD Model]
[End EMD Set]
|*****************************************************************************
```

13.5.4 EXAMPLE 3 (R123 IBIS MODEL TERMINALS SPLIT INTO TWO [EMD MODEL]S, POWER RAILS IN A SEPARATE [EMD MODEL])

```
|*****************************************************************************
| EMD Syntax Example 3 (External Resistors, Separate A07, A07R, and POWER
| Models)
| Using DDR4 RDIMM Example
|
[Begin EMD] DDR4_RDIMM_3
[Number of EMD Pins] 4
[EMD Pin List] signal_name signal_type  bus_label
203            VSS         GND
211            A07
212            VDD         POWER        VDD1
```

```
223            VTT         POWER
[End EMD Pin List]

[EMD Parts]
DDR4_Reg_253b  register.ibs         DDR4_Register
DDR4_x8_78b    dram.ibs             DDR4_8Gb_x8
510-500874     resistors.ibs        RES_22ohms
510-501618     resistors.ibs        RPACK4_33ohms
[End EMD Parts]

[EMD Designator List]
U3        DDR4_Reg_253b
U4        DDR4_x8_78b
U5        DDR4_x8_78b
U7        DDR4_x8_78b
U8        DDR4_x8_78b
R123      510-500874
RN13      510-501618
[End EMD Designator List]

[Designator Pin List] signal_name  signal_type  bus_label
U3.B9                 VDD          POWER        VDD1
U3.B11                BA07
U3.B12                VSS          GND
U3.V3                 VDD          POWER        VDD1
U3.W1                 A07r                    | Net A07r Terminal
U3.W3                 VSS          GND
U4.K9                 VSS          GND
U4.M8                 BA07
U4.N9                 VDD          POWER        VDD1
U5.K9                 VSS          GND
U5.M8                 BA07
U5.N9                 VDD          POWER        VDD1
U7.K9                 VSS          GND
U7.M8                 BA07
U7.N9                 VDD          POWER        VDD1
U8.K9                 VSS          GND
U8.M8                 BA07
U8.N9                 VDD          POWER        VDD1
R123.1                A07                     | Net A07 Terminal
R123.2                A07r                    | Net A07r Terminal
RN13.2                VTT          POWER
RN13.7                BA07
[End Designator Pin List]

[Voltage List]
VDD   1.200   1.140   1.260
VSS   0.000   0.000   0.000
VTT   0.600   0.570   0.630
[End Voltage List]

[EMD Group]    Addr_07_Group_3
Addr_07_3         NA
RIGHT_SIDE_POWER  NA
[End EMD Group]
[End EMD]
```

```
[EMD Set]      Addr_07_3
[Manufacturer]  NoName Corp.
[EMD Model]     A07_3
File_IBIS-ISS   A07.iss       A07_3
Number_of_terminals = 3
1  Pin_I/O      pin_name      211
2  Pin_I/O      pin_name      R123.1  | Net A07 Terminals and Connection
                                      | Series Resistor is in two [EMD Model]s
3  Pin_Rail     signal_name   VSS
[End EMD Model]
|
[EMD Model]     A07R_3
File_IBIS-ISS   A07.iss       A07R_3
Number_of_terminals = 3
1  Pin_I/O      pin_name      R123.2  | Net A07r Terminal and Connection
2  Pin_I/O      pin_name      U3.W1   | Net A07r Terminal and Connection
3  Pin_Rail     signal_name   VSS
[End EMD Model]
|
[EMD Model]     BA07_3
File_IBIS-ISS   A07.iss      BA07_3
Number_of_terminals = 13
1  Pin_I/O      pin_name      U3.B11
2  Pin_Rail     signal_name   U3.VSS
3  Pin_I/O      pin_name      U4.M8
4  Pin_Rail     signal_name   U4.VSS
5  Pin_I/O      pin_name      U5.M8
6  Pin_Rail     signal_name   U5.VSS
7  Pin_I/O      pin_name      U7.M8
8  Pin_Rail     signal_name   U7.VSS
9  Pin_I/O      pin_name      U8.M8
10 Pin_Rail     signal_name   U8.VSS
11 Pin_I/O      pin_name      RN13.7
12 Pin_Rail     signal_name   RN13.VTT
13 Pin_Rail     signal_name   VSS
[End EMD Model]
[End EMD Set]

[EMD Set]      RIGHT_SIDE_POWER
[EMD Model]     RIGHT_SIDE_VDD1_VTT_VSS
File_IBIS-ISS   rdimm_power.iss RIGHT_SIDE_VDD1_VTT_VSS
Number_of_terminals = 14
1  Pin_Rail     bus_label     VDD1
2  Pin_Rail     signal_name   VSS
3  Pin_Rail     signal_name   VTT
4  Pin_Rail     bus_label     U3.VDD1
5  Pin_Rail     signal_name   U3.VSS
6  Pin_Rail     bus_label     U4.VDD1
7  Pin_Rail     signal_name   U4.VSS
8  Pin_Rail     bus_label     U5.VDD1
9  Pin_Rail     signal_name   U5.VSS
10 Pin_Rail     bus_label     U7.VDD1
11 Pin_Rail     signal_name   U7.VSS
12 Pin_Rail     bus_label     U8.VDD1
13 Pin_Rail     signal_name   U8.VSS
14 Pin_Rail     signal_name   RN13.VTT
```

```
[End EMD Model]
[End EMD Set]
|*****************************************************************************
```

13.6 CONNECTION RULES FOR EMD GROUP, EMD SET, AND EMD MODEL

At the [EMD Group] level, the connections between the referenced [EMD Set]s (and their encapsulated [EMD Model]s) are determined by the following rules:

1. I/O pins (Pin_I/O terminals by pin_name entries)
    a. Without Aggressor_Only:
        i. I/O terminals may exist with or without rail terminals
        ii. Within each [EMD Model], pin_name entries (as listed in the [EMD Pin List] keyword) shall be distinct for I/O pins
        iii. Within each [EMD Model], <designator>.<pin_name> entries (as listed in the [Designator Pin List] keyword) shall be distinct for I/O pins
        iv. At any one interface and for all [EMD Model]s referenced by all [EMD Set]s under an [EMD Group], no duplicate pin_name entries are permitted for I/O pins
        v. Electrical connections between I/O pins are based on the content of the referenced electrical models (*.iss or Touchstone files)
        vi. Net connections are indicated by identical signal_name entries available from the [EMD Pin List] and/or [Designator Pin List] entries. In Example 1, Pin_I/O pin_name 211 and Pin_I/O pin_name U3.W1 are considered connected through the IBIS-ISS subcircuit because they both share the same signal_name, A07
        vii. The logical and electrical connections can span several interfaces. In Example 1, Pin_I/O pin_name U3.B11, Pin_I/O pin_name U4.M8, etc. share the same signal_name BA07 and are therefore in the same net
    b. With Aggressor_Only:
        i. I/O terminals may exist with or without rail terminals
        ii. To permit selection of nets within an [EMD Group], identical pin_name entries are permitted in <u>different</u> [EMD Model] keywords if there is no overlap of pin_name entries at the same interface <u>without</u> Aggressor_Only. For example, “Pin_I/O pin_name 211” and “Pin_I/O pin_name 211 Aggressor_Only” can exist under different [EMD Model] keywords but will not be used together in simulation
        iii. The complete I/O net for a given signal_name entry is deemed Aggressor_Only if one or more of the pin_names in the net has an Aggressor_Only column entry
        iv. At least one net shall exist without Aggressor_Only
2. Rail (Pin_Rail terminals) connections by pin_name, signal_name, bus_label

a. Within an [EMD Group] and for all referenced [EMD Set] keywords and their encapsulated [EMD Model] keywords, identically named rail terminals shall be considered connected based on these rules:
    i. Rail terminals may exist with or without I/O terminals
    ii. At an EMD Pin List interface, identical Pin_Rail pin_name, bus_label or signal_name entries in different [EMD Model]s shall be considered shorted
    iii. At a Designator Pin List interface, identical Pin_Rail pin_name, bus_label, or signal_name entries in different [EMD Model]s shall be considered shorted
    iv. For each [EMD Model] and at any one interface, there shall not be any overlap of Pin_Rail pin_name, bus_label and signal_name entries:
        1. A pin_name entry shall not overlap with a bus_label entry
        2. A pin_name entry shall not overlap with a signal_name entry
        3. A bus_label entry shall not overlap with a signal_name entry
    v. For all [EMD Model]s and at any one interface, where Pin_Rail pin_name, bus_label and/or signal_name entries in different [EMD Model]s overlap:
        1. A pin_name entry shall be shorted with a corresponding bus_label entry
        2. A pin_name entry shall be shorted with a corresponding signal_name entry
        3. A bus_label entry shall be shorted with a corresponding signal_name entry

b. Within an [EMD Group] and for all referenced [EMD Set] keywords and their encapsulated [EMD Model] keywords, Pin_Rail terminals are considered merged into a single terminal across designator interfaces (not the EMD interface) based on these rules:
    i. Pin_Rail signal_name *.<signal_name> shorts all connections with signal_name <signal_name> for all designator interfaces (not the EMD interface)
    ii. Pin_Rail bus_label *.<bus_label> shorts all connections with bus_label <bus_label> for all designator interfaces (not the EMD interface)
    iii. No corresponding rule for pin_name entries exists since connected rail pin_names can differ at different interfaces

c. Simulator Global Reference:
    i. Terminal_type A_gnd can be used for a simulator global reference in any EMD Model
    ii. All simulator global references are shorted

13.7 ADDITIONAL EMD MODEL EXAMPLES

Examples:

The example below for a simplified DIMM includes pins at the EMD interface and at the designator interfaces of two memory components. Three EMD Groups provide EMD Model options including one option with no crosstalk and two options with crosstalk included. The EMD Groups with crosstalk included show use of IBIS-ISS or Touchstone files, and the rail connections are modeled in separate EMD Sets that are included in each EMD Group. The rail terminals are connected by either bus_label or signal_name. Bus_labels are used to split the VDD rail into VDD1 and VDD2 buses. While only one VSS rail is shown, separate VSS rails could exist (for example, VSS1 and VSS2) and would be included by using bus_label syntax.

```
[Begin EMD] DIMM
[Number of EMD Pins] 7
[EMD Pin List] signal_name  signal_type  bus_label
A1             DQ0
A2             DQ1
A3             DQ2
A4             DQ3
P1             VDD          POWER        VDD1
P2             VDD          POWER        VDD2
G1             VSS          GND
[End EMD Pin List]

[EMD Parts]
ACME_MEM  mem.ibs  MEMx4
[End EMD Parts]

[EMD Designator List]
U1  ACME_MEM
U2  ACME_MEM
[End EMD Designator List]

[Designator Pin List] signal_name  signal_type  bus_label
U1.1                  VDD          POWER        VDD1
U1.2                  VDD          POWER        VDD2
U1.3                  VSS          GND
U1.4                  VSS          GND
U1.5                  DQ0
U1.6                  DQ1
U1.7                  DQ2
U1.8                  DQ3
|
U2.1                  VDD          POWER        VDD1
U2.2                  VDD          POWER        VDD2
U2.3                  VSS          GND
U2.4                  VSS          GND
U2.5                  DQ0
U2.6                  DQ1
U2.7                  DQ2
U2.8                  DQ3
[End Designator Pin List]
```

```
| EMD Group has no crosstalk modeled and includes the
| rails in the same IBIS-ISS subcircuit
[EMD Group]  All_DQs_No_Coupling_Rails
All_DQs_Uncoupled     NA
[End EMD Group]

| EMD Group models crosstalk with IBIS-ISS subcircuits
[EMD Group]  All_DQs_Aggressor_Options_ISS
All_DQs_Crosstalk_ISS NA
Rails_ISS             NA
[End EMD Group]

| EMD Group models crosstalk with Touchstone files
[EMD Group]  All_DQs_Aggressor_Options_TS
All_DQs_Crosstalk_TS  NA
Rails_TS              NA
[End EMD Group]

[End EMD]             | End of [Begin EMD]

|************************** EMD Sets ************************

[EMD Set]       All_DQs_Uncoupled
[Manufacturer]  NoName Corp.
[EMD Model]     DQ0_3
File_IBIS-ISS   DQ.iss        DQ
Number_of_terminals = 20
1  Pin_I/O      pin_name      A1        | DQ0
2  Pin_I/O      pin_name      A2        | DQ1
3  Pin_I/O      pin_name      A3        | DQ2
4  Pin_I/O      pin_name      A4        | DQ3
5  Pin_Rail     signal_name   VDD       | EMD Pins P1 and P2
6  Pin_Rail     signal_name   VSS       | EMD Pin G1
|
7  Pin_I/O      pin_name      U1.5      | DQ0
8  Pin_I/O      pin_name      U1.6      | DQ1
9  Pin_I/O      pin_name      U1.7      | DQ2
10 Pin_I/O      pin_name      U1.8      | DQ3
11 Pin_Rail     bus_label     U1.VDD1   | U1 Pin 1
12 Pin_Rail     bus_label     U1.VDD2   | U1 Pin 2
13 Pin_Rail     signal_name   U1.VSS    | U1 Pins 3 and 4
|
14 Pin_I/O      pin_name      U2.5      | DQ0
15 Pin_I/O      pin_name      U2.6      | DQ1
16 Pin_I/O      pin_name      U2.7      | DQ2
17 Pin_I/O      pin_name      U2.8      | DQ3
18 Pin_Rail     bus_label     U2.VDD1   | U2 Pin 1
19 Pin_Rail     bus_label     U2.VDD2   | U2 Pin 2
20 Pin_Rail     signal_name   U2.VSS    | U2 Pins 3 and 4
[End EMD Model]
[End EMD Set]

[EMD Set]       All_DQs_Crosstalk_ISS
| EMD Model includes all crosstalk contributions for DQ1.
| Crosstalk contributions are incomplete for other nets
| marked as Aggressor_Only.
```

```
[Manufacturer]  NoName Corp.
[EMD Model]     DQ1_Victim
File_IBIS-ISS   DQ.iss        DQ1_Victim
Number_of_terminals = 15
1  Pin_I/O      pin_name      A1        Aggressor_Only  | DQ0
2  Pin_I/O      pin_name      A2                        | DQ1
3  Pin_I/O      pin_name      A3        Aggressor_Only  | DQ2
4  Pin_I/O      pin_name      A4        Aggressor_Only  | DQ3
5  Pin_Rail     signal_name   VSS
|
6  Pin_I/O      pin_name      U1.5      | DQ0
7  Pin_I/O      pin_name      U1.6      | DQ1
8  Pin_I/O      pin_name      U1.7      | DQ2
9  Pin_I/O      pin_name      U1.8      | DQ3
10 Pin_Rail     signal_name   U1.VSS    | U1 Pins 3 and 4
|
11 Pin_I/O      pin_name      U2.5      | DQ0
12 Pin_I/O      pin_name      U2.6      | DQ1
13 Pin_I/O      pin_name      U2.7      | DQ2
14 Pin_I/O      pin_name      U2.8      | DQ3
15 Pin_Rail     signal_name   U2.VSS    | U2 Pins 3 and 4
[End EMD Model]

| EMD Model includes all crosstalk contributions for DQ2.
| Crosstalk contributions are incomplete for other nets
| marked as Aggressor_Only.
[EMD Model]     DQ2_Victim
File_IBIS-ISS   DQ.iss        DQ2_Victim
Number_of_terminals = 15
1  Pin_I/O      pin_name      A1        Aggressor_Only  | DQ0
2  Pin_I/O      pin_name      A2        Aggressor_Only  | DQ1
3  Pin_I/O      pin_name      A3                        | DQ2
4  Pin_I/O      pin_name      A4        Aggressor_Only  | DQ3
5  Pin_Rail     signal_name   VSS
|
6  Pin_I/O      pin_name      U1.5      | DQ0
7  Pin_I/O      pin_name      U1.6      | DQ1
8  Pin_I/O      pin_name      U1.7      | DQ2
9  Pin_I/O      pin_name      U1.8      | DQ3
10 Pin_Rail     signal_name   U1.VSS    | U1 Pins 3 and 4
|
11 Pin_I/O      pin_name      U2.5      | DQ0
12 Pin_I/O      pin_name      U2.6      | DQ1
13 Pin_I/O      pin_name      U2.7      | DQ2
14 Pin_I/O      pin_name      U2.8      | DQ3
15 Pin_Rail     signal_name   U2.VSS    | U2 Pins 3 and 4
[End EMD Model]
[End EMD Set]

[EMD Set]       Rails_ISS
[Manufacturer]  NoName Corp.
[EMD Model]     Power_Rails
File_IBIS-ISS   Power_Rails.iss  Rails
Number_of_terminals = 8
1  Pin_Rail     signal_name   VDD       | EMD Pins P1 and P2
2  Pin_Rail     signal_name   VSS       | EMD Pin G1
```

```
|
3  Pin_Rail     bus_label     U1.VDD1   | U1 Pin 1
4  Pin_Rail     bus_label     U1.VDD2   | U1 Pin 2
5  Pin_Rail     signal_name   U1.VSS    | U1 Pins 3 and 4
|
6  Pin_Rail     bus_label     U2.VDD1   | U2 Pin 1
7  Pin_Rail     bus_label     U2.VDD2   | U2 Pin 2
8  Pin_Rail     signal_name   U2.VSS    | U2 Pins 3 and 4
[End EMD Model]
[End EMD Set]

[EMD Set]       All_DQs_Crosstalk_TS
| EMD Model includes all crosstalk contributions for DQ1.
| Crosstalk contributions are incomplete for other nets
| marked as Aggressor_Only.
[Manufacturer]  NoName Corp.
[EMD Model]     DQ1_Victim
File_TS         DQ1_Victim.ts
Unused_port_termination       Reference
Number_of_terminals = 25
1  Pin_I/O      pin_name      A1        Aggressor_Only  | DQ0
3  Pin_I/O      pin_name      A2                        | DQ1
5  Pin_I/O      pin_name      A3        Aggressor_Only  | DQ2
7  Pin_I/O      pin_name      A4        Aggressor_Only  | DQ3
|
9  Pin_I/O      pin_name      U1.5      | DQ0
11 Pin_I/O      pin_name      U1.6      | DQ1
13 Pin_I/O      pin_name      U1.7      | DQ2
15 Pin_I/O      pin_name      U1.8      | DQ3
|
17 Pin_I/O      pin_name      U2.5      | DQ0
19 Pin_I/O      pin_name      U2.6      | DQ1
21 Pin_I/O      pin_name      U2.7      | DQ2
23 Pin_I/O      pin_name      U2.8      | DQ3
|
25 A_gnd        | Reference for all ports
[End EMD Model]

| EMD Model includes all crosstalk contributions for DQ2.
| Crosstalk contributions are incomplete for other nets
| marked as Aggressor_Only.
[EMD Model]     DQ2_Victim
File_TS         DQ2_Victim.ts
Unused_port_termination       Reference
Number_of_terminals = 25
1  Pin_I/O      pin_name      A1        Aggressor_Only  | DQ0
3  Pin_I/O      pin_name      A2        Aggressor_Only  | DQ1
5  Pin_I/O      pin_name      A3                        | DQ2
7  Pin_I/O      pin_name      A4        Aggressor_Only  | DQ3
|
9  Pin_I/O      pin_name      U1.5      | DQ0
11 Pin_I/O      pin_name      U1.6      | DQ1
13 Pin_I/O      pin_name      U1.7      | DQ2
15 Pin_I/O      pin_name      U1.8      | DQ3
|
17 Pin_I/O      pin_name      U2.5      | DQ0
```

```
19 Pin_I/O      pin_name      U2.6      | DQ1
21 Pin_I/O      pin_name      U2.7      | DQ2
23 Pin_I/O      pin_name      U2.8      | DQ3
|
25 A_gnd        | Reference for all ports
[End EMD Model]
[End EMD Set]

[EMD Set]       Rails_TS
[Manufacturer]  NoName Corp.
[EMD Model]     Power_Rails
File_TS         Power_Rails_TS.s8p
Number_of_terminals = 9
1  Pin_Rail     signal_name   VDD       | EMD Pins P1 and P2
2  Pin_Rail     signal_name   VSS       | EMD Pin G1
|
3  Pin_Rail     bus_label     U1.VDD1   | U1 Pin 1
4  Pin_Rail     bus_label     U1.VDD2   | U1 Pin 2
5  Pin_Rail     signal_name   U1.VSS    | U1 Pins 3 and 4
|
6  Pin_Rail     bus_label     U2.VDD1   | U2 Pin 1
7  Pin_Rail     bus_label     U2.VDD2   | U2 Pin 2
8  Pin_Rail     signal_name   U2.VSS    | U2 Pins 3 and 4
|
9  A_gnd        | Reference for all ports
[End EMD Model]
[End EMD Set]
```

Keyword: **[End EMD Model]**

Required: Yes

Description: Marks the end of an EMD Model.

Usage Rules: This keyword must come at the end of each complete electrical EMD Model.

Example:

```
[End EMD Model]
```

14 EMI PARAMETERS

There are two sections here: one for a [Component] and one for a [Model].

This section describes the structure of the EMI parameters under a top-level [Component] keyword. It is used to describe the EMI parameters associated with a [Component]. The parameters shall be surrounded by the [Begin EMI Component] and [End EMI Component] keywords.

The following keywords are defined:

[Begin EMI Component]
[End EMI Component]
[Pin EMI]
[Pin Domain EMI]

The following subparameters are defined:

Domain
Cpd
C_Heatsink_gnd
C_Heatsink_float

Keyword: **[Begin EMI Component]**

Required: No

Description: Marks the beginning of the Component EMI parameters.

Sub-Params: Domain, Cpd, C_Heatsink_gnd, C_Heatsink_float

Domain indicates whether the component is digital, analog, or part digital part analog. Analog circuits are more susceptible to low-level noise. Analog circuits operate at very low signal levels (mV or uV) and can contain high gain amplifiers. In contrast, digital circuits operate at relatively large signal levels (compared to analog circuits).

The syntax for Domain is:

Domain Domain_value

where Domain_value is an enumerated argument, and is one of:

Digital, Analog, Digital_analog

This subparameter is optional. If not entered, the default is Digital.

Cpd is the power dissipation capacitance parameter. Cpd (Power Dissipation Capacitance) is the internal parasitic capacitance (e.g., gate-to-source and gate-to-drain capacitance) plus the equivalent capacitance associated with the through currents when both transistors (n-channel and p-channel) are momentarily conducting.

Cpd is typically for CMOS devices, and helps provide a more accurate estimation of the power bus current, and therefore the noise voltage on the power bus. If the high frequency noise on the power bus (due to switching of digital circuits) is known, then the radiation can be calculated.

Sometimes Iccd (dynamic power supply current) is found in data books. It is normally given for FACT families. Iccd is specified in units of mA/MHz.

Cpd can be calculated from Iccd by the equation:

Cpd (nF) = Iccd (mA/MHz) / Vcc (V).

The syntax for Cpd is:

Cpd = capacitance_value

The units of capacitance_value are farads.

This subparameter is optional. If not entered, the default is 0.0 F.

C_Heatsink_float and C_Heatsink_gnd define the heatsink capacitance and connection conditions. C_Heatsink_float indicates that the heatsink is floating, and C_Heatsink_gnd indicates that the heatsink is grounded.

Internal currents inside a (high speed) IC can be closely coupled onto a heatsink. As the heatsink is physically much larger than the IC silicon chip and bond wires, it is a more efficient radiator. Knowing the capacitance of the heatsink, the radiated electric field can be estimated.

Only one of these subparameters can be defined. It is not legal to define both. It is legal to omit both. In this case it means that a heatsink is not present.

Each of these subparameters takes one argument: the heatsink capacitance.

The syntax for the heatsink capacitance subparameters is:

C_Heatsink_float = capacitance_value
C_Heatsink_gnd = capacitance_value

The units for capacitance_value are farads.

This subparameter is optional. If not entered, the default is that the component does not have a heatsink.

Keyword: **[End EMI Component]**

Required: No

Description: Marks the end of the Component EMI parameters.

Example:

```
[Begin EMI Component]
Domain           Digital
Cpd            = 6.4pF
C_Heatsink_gnd = 3.4pF
[End EMI Component]
```

Keyword: **[Pin EMI]**

Required: No

Description: Specifies the EMI parameters for a Pin.

Sub-Params: domain_name, clock_div

Usage Rules: Each line must contain three columns. The first column shall contain the pin name. This pin name shall match a pin name in the [Pin] keyword (the pin name is the first column in the [Pin] record).

The second column is the domain name. This specifies the clock domain for that pin. This is used by [Pin Domain EMI]. The field should be set to NA if unused.

The default for domain_name is that the percentage of power used is 100%.

The third column is the clock division. This is the ratio of the frequency at this pin to the reference pin. The reference pin is always set to "1.0". The ratio is a floating-point number. The choice of the reference pin does not matter as this information is pin to pin ratios. It is suggested that the pin with the maximum frequency is chosen as the reference.

The field should be set to NA if unused.

The default for clock_div is 1.0.

Column length limits are:

pin_name	20 characters max
domain_name	20 characters max
clock_div	5 characters max

It is not a requirement to specify every pin. An undefined pin will default to 100% power usage for domain_name, and 1.0 for clock_div.

Keyword: **[Pin Domain EMI]**

Required: No

Description: Specifies the percentage of power used in each clock domain.

Sub-Params: percentage

Usage Rules: Each line must contain two columns. The first column must contain the domain_name. This name must match a domain name in the [Pin EMI] keyword (the domain name is the second column in that record).

The percentage represents a user-definable percentage of the power used by that domain. It is an integer in the range 0 < percentage =< 100.

Column length limits are:

domain_name	20 characters max
percentage	5 characters max

Example:

```
[Begin EMI Component]
Domain          Digital
Cpd           = 6.4pF
|
[Pin EMI]   domain_name    clock_div
 4          MEM            0.5
 5          MEM            0.5
```

```
 7          NA             0.5           | domain_name defaults to 100%
 8          RIOG           NA            | clock_div defaults to 1.0
14          CPU            1.0
15          RIOG           0.5
|

[Pin Domain EMI]   percentage
 CPU               40
 MEM               30
 RIOG              30
|
[End EMI Component]
```

This section describes the structure of the EMI parameters under a top-level [Model] keyword. It is used to describe the EMI parameters associated with a [Model]. The parameters must be surrounded by the [Begin EMI Model] and [End EMI Model] keywords.

The following keywords are defined:

[Begin EMI Model]
[End EMI Model]

The following subparameters are defined:

Model_emi_type
Model_Domain

Keyword: **[Begin EMI Model]**

Required: No

Description: Marks the beginning of the Model EMI parameters.

Sub-Params: Model_emi_type, Domain

Model_emi_type indicates whether the model (for this pin) is a ferrite or not.

The syntax for Model_emi_type is:

Model_emi_type Model_emi_type_value

where Model_emi_type_value is an enumerated argument, and is one of:

Ferrite, Not_a_ferrite

If not entered (the default), the model is Not_a_ferrite.

Model_Domain indicates whether the model is digital or analog. This is only used if the [Component EMI] Domain is set to Digital_analog. If the [Component EMI] Domain is set to anything else, Model_Domain is ignored.

The syntax for Domain is:

Model_Domain Domain_value

where Domain_value is one of:

Digital, Analog

If not entered, the default is to use the [Component EMI] Domain setting and its default.

Keyword: **[End EMI Model]**

Required: No

Description: Marks the end of the Model EMI parameters.

Example:

```
[Begin EMI Model]
Domain          Analog
Model_emi_type  Ferrite
[End EMI Model]
```