IBIS

(I/O Buffer Information Specification)

Version 6.0

Ratified September 20, 2013

Contents

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 1993), IBIS Version 2.1 (ratified as ANSI/EIA-656 in December 1995), IBIS Version 3.2 (ratified as ANSI/EIA-656-A in October 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), and IBIS Version 6.0 (ratified on September 20, 2013).

The functionality of IBIS follows in Section 3.1 (formerly Section 3A) through Section 8. Sections 3.1 through 6 describe the format of the core functionality of IBIS Version 1.1 and the 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 format of IBIS Version 3.2. Along with Section 4 header information, electrical board descriptions must be contained in separate .ebd files.

Sections 10.1, 1.1, and 11 (formerly Sections 6C, 10, and 11, respectively) are new in IBIS Version 5.0 and contain reference and modeling information related to the algorithmic modeling interface (AMI) support, and EMI parameters. Sections 6.4 and 10.3 (formerly Sections 6D and 10A, respectively) are new in IBIS Version 5.1, to place test loads and data appropriately in the keyword hierarchy and to more fully describe algorithmic models, respectively. Section 10.5 is added in IBIS Version 6.0, to describe the keyword, AMI parameters, and data flow associated with repeaters. IBIS Version 6.0 also modifies the organization of the document.

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.

2 STATEMENT OF INTENT

In order to enable an industry standard method to electronically transport IBIS modeling data between semiconductor vendors, EDA tool vendors, and end customers, this template is proposed. The intention of this template 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 this template is to represent the current state of IBIS data, while allowing a growth path to more complex models/methods (when deemed appropriate). This would be accomplished by a revision of the base template, and possibly the addition of new keywords or categories.

Another goal of this template 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 template 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 6.0 of this electronic template 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.eda.org/ibis/.

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 modeling platforms develop support for revisions of the IBIS ASCII template, all previous revisions of the template 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 2.0 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

3 GENERAL SYNTAX RULES AND GUIDELINES

This section contains general syntax rules and guidelines for ASCII .ibs files:

1. The content of the files is case sensitive, except for reserved words and keywords.
2. The following words are reserved words and must not be used for any other purposes in the document:

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

3. To facilitate portability between operating systems, file names used in a .ibs file must only have lower case characters. File names should have a basename of no more than forty (40) characters followed by a period (“.”), followed by a file name extension of no more than three characters. The file name and extension must use characters from the set (space, “ ”, 0x20 is not included):

   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 _ ^ $ ~ ! # % & - { }) (@ ‘ `

   The file name and extension are recommended to be lower case on systems that support such names.
4. A line of the file may have at most 120 characters, followed by a line termination sequence. The line termination sequence must be one of the following two sequences: a linefeed character or a carriage return followed by 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 as desired.
6. Keywords must be enclosed in square brackets, “[]”, and must start in column 1 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.
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 factors are 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, it is allowed to use full abbreviations for the units, (e.g., pF, nH, mA, mOhm). 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 they 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 120 characters, which is illegal in .ibs 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, may be used in a .ibs file. The use of characters with codes greater than hexadecimal 07E is not allowed. Also, ASCII control characters (those numerically less than hexadecimal 20) are not allowed, except for tabs or in a line termination sequence. As mentioned in item 10 above, the use of tab characters is discouraged.

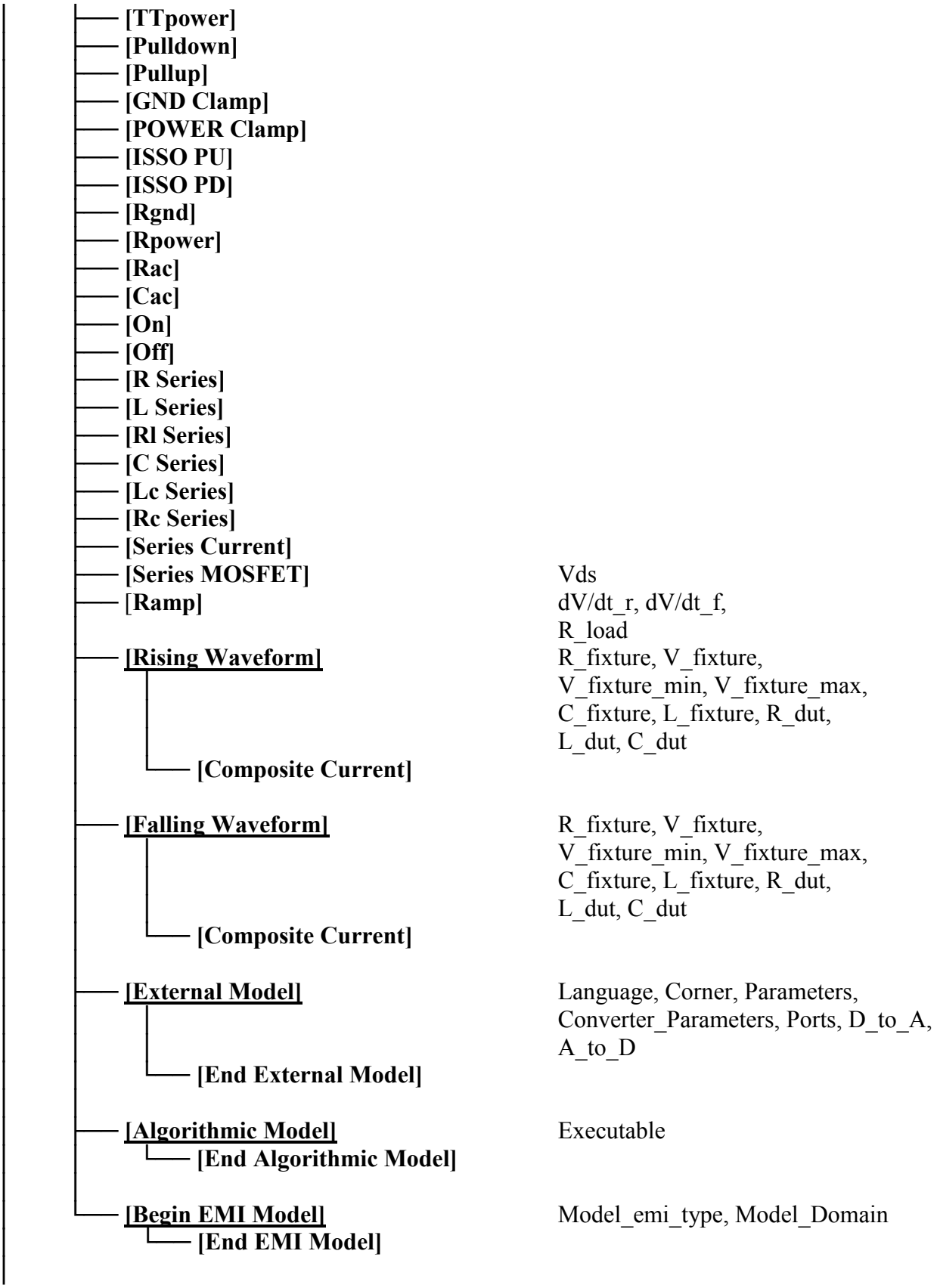
3.1 KEYWORD HIERARCHY

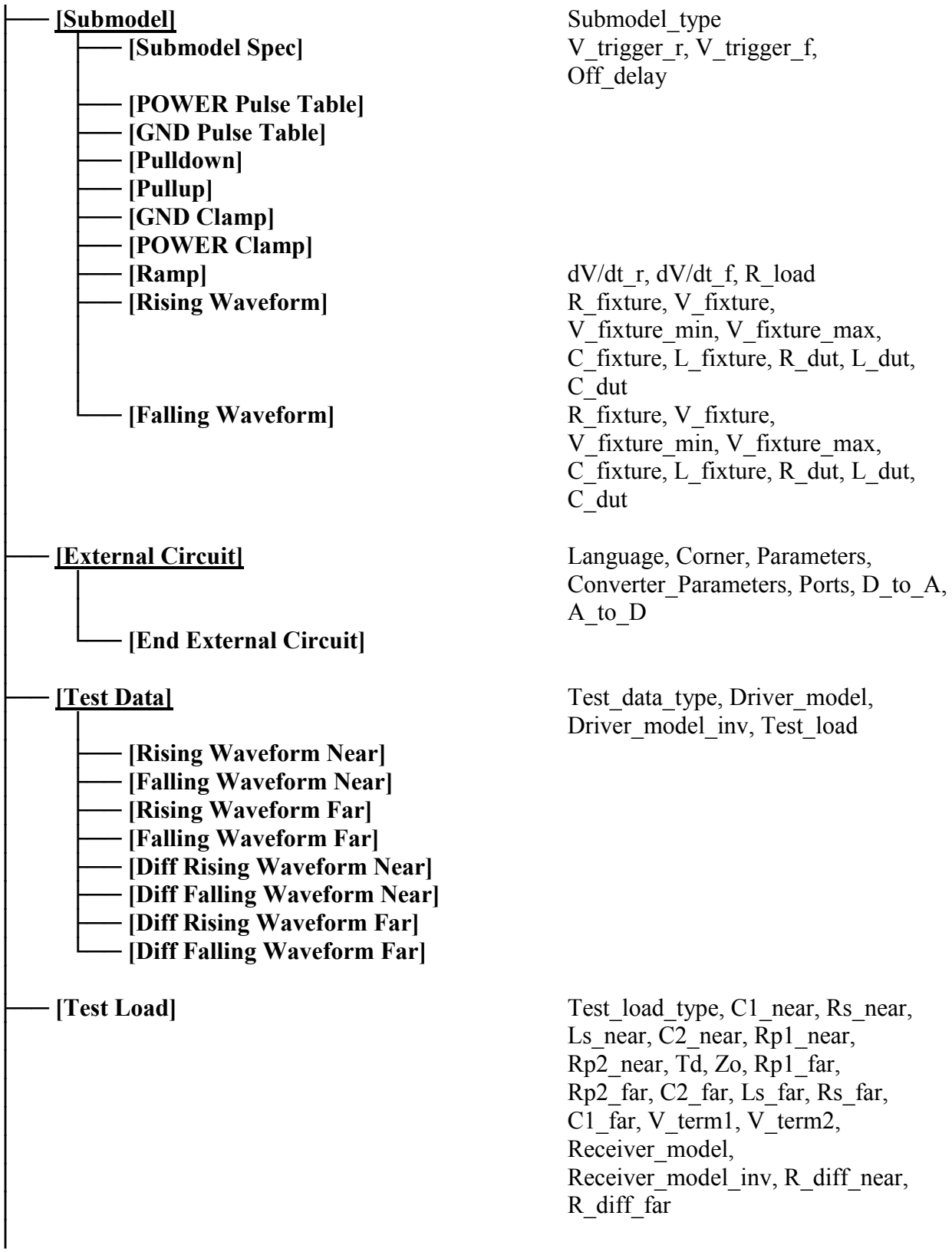
.ibs FILE

- 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
  - **[Package Model]**
    - **[Alternate Package Models]**
      - **[End Alternate Package Models]**
  - **[Pin Mapping]** — pulldown_ref, pullup_ref, gnd_clamp_ref, power_clamp_ref, ext_ref
  - **[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
    - **[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
  - **[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]
[External Model]
Language, Corner, Parameters,
Converter_Parameters, Ports, D_to_A,
A_to_D
[End External Model]
[Algorithmic Model]
Executable
[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
[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
  - **[Model Data]**
    - **[Resistance Matrix]** — Banded_matrix, Sparse_matrix, Full_matrix
      - **[Bandwidth]**
      - **[Row]**
    - **[Inductance Matrix]** — Banded_matrix, Sparse_matrix, Full_matrix
      - **[Bandwidth]**
      - **[Row]**
    - **[Capacitance Matrix]** — Banded_matrix, Sparse_matrix, Full_matrix
      - **[Bandwidth]**
      - **[Row]**
    - **[End Model Data]**
  - **[End Package Model]**
- **[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
- **[Model Data]**
  - **[Resistance Matrix]** — Banded_matrix, Sparse_matrix, Full_matrix
    - **[Bandwidth]**
    - **[Row]**
  - **[Inductance Matrix]** — Banded_matrix, Sparse_matrix, Full_matrix
    - **[Bandwidth]**
    - **[Row]**
  - **[Capacitance Matrix]** — Banded_matrix, Sparse_matrix, Full_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]

4 FILE HEADER INFORMATION

Keyword: **[IBIS Ver]**

Required: Yes

Description: Specifies the IBIS template version. 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 "|".

Example:

```
[IBIS Ver]  6.0   | Used for template variations
```

Keyword: **[Comment Char]**

Required: No

Description: Defines a new comment character to replace the default "|" (pipe) character, if desired.

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, as desired.

Example:

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

Keyword: **[File Name]**

Required: Yes

Description: Specifies the name of the .ibs file.

Usage Rules: The file name must conform to the rules in paragraph 3 of Section 3, "GENERAL SYNTAX RULES AND GUIDELINES". In addition, the file name must use the extension ".ibs", ".pkg", or ".ebd". The file name must be the actual name of the file.

Example:

```
[File Name]     ver5_1.ibs
```

Keyword: **[File Rev]**

Required: Yes

Description: Tracks the revision level of a particular .ibs 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 .ibs file variations
```

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

Required: No

Description: Optionally clarifies the 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]          September 20, 2013          | 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 2013, XYZ Corp., All Rights Reserved
```

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". The default location is at the "Pin".

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.

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] keyword for more complex package descriptions.

If defined, the [Package Model] 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: All pins on a component must be specified. The first column must contain the pin name. 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 must 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, or NC).

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]	5 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, "GENERAL SYNTAX RULES AND GUIDELINES".

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 globally 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: **[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. 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 must be given.

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]	5 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: **[Diff Pin]**

Required: No

Description: Associates differential pins and defines their differential receiver threshold voltage 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. All 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]	5 characters max
inv_pin	5 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]'s entries are. The simulator 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: **[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 for 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 an alphanumeric designator string 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 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 is omitted.

The column length limits are:

[Series Pin Mapping]	5 characters max
pin_2	5 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 and any other elements such as additional capacitance or clamping circuitry are defined by the model_name that is referenced in the [Pin] keyword. 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 is 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 120 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 can be 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 white space. 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 120-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 must match the 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.
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.

Model Type	Definition
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 not illegal to include the C_comp subparameter together with one or more of the remaining C_comp_* subparameters, but in that

case the simulator 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 simulator 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 simulator 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.

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.

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 the 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 are recognized only when the [Diff Pin] keyword connects the models. This applies for the true differential buffers in Section 6.3 and also for 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 the model. 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 1.

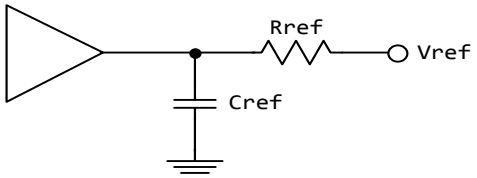


Figure 1 - Reference Load Connections

A single-ended or true differential buffer can have Rref_diff and Cref_diff (Figure 2).

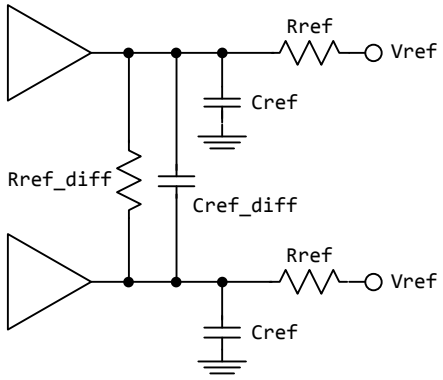


Figure 2 - Single-Ended or True Differential Buffer

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], [Rpower], [Rac], and [Cac] keywords. 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 any of [Rgnd], [Rpower], [Rac], and [Cac] keywords 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
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, C_ref_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] must follow all other 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 white space. All four columns are required under the [Model Spec] keyword. However, data is 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 specifications 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 3.

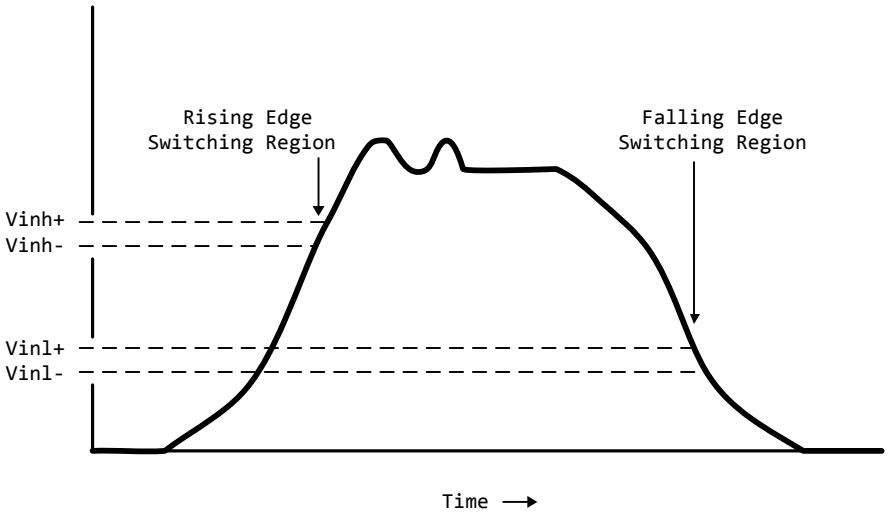


Figure 3 - Receiver Voltage with Hysteresis Thresholds

S_overshoot_high, S_overshoot_low rules:

The static overshoot subparameters provide the DC voltage values for 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 4.

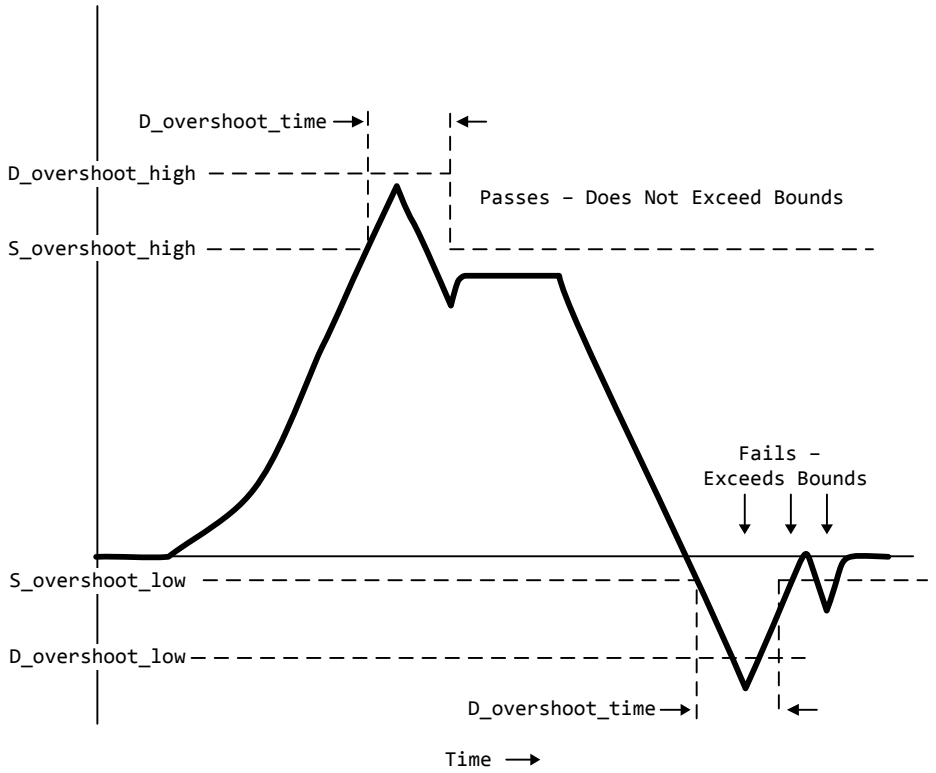


Figure 4 - Receiver Voltage with Static and Dynamic Overshoot Limits

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 5. 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 other static and dynamic overshoot parameters are optional.

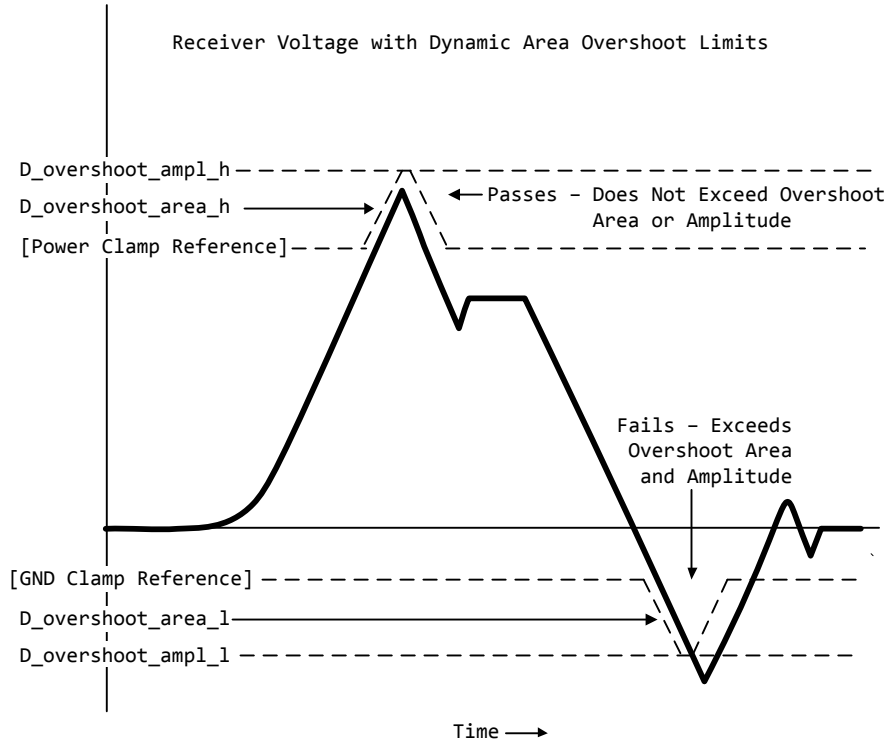


Figure 5 - Receiver Voltage with Dynamic Area Overshoot Limits

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 be below 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 6.

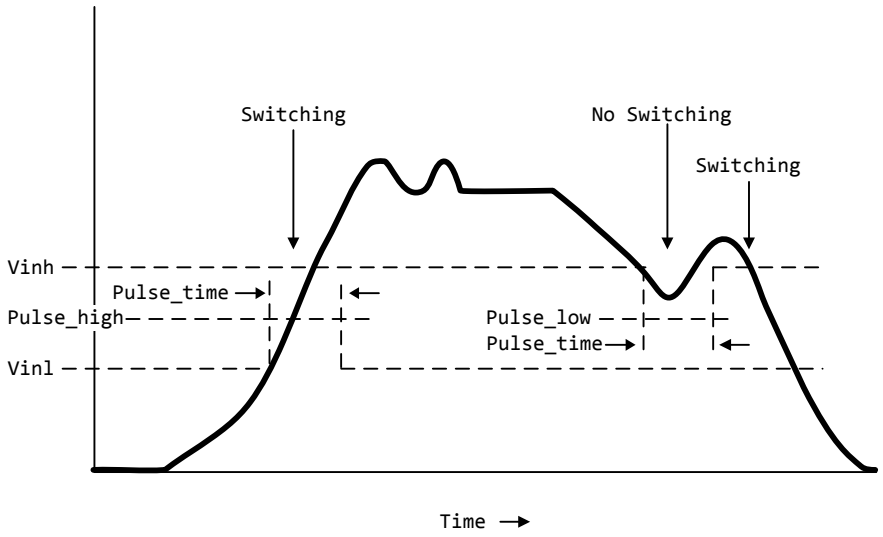


Figure 6 - Receiver Voltage with Pulse Immunity Thresholds

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 subparameters 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 IO circuit uses a simple weak tie-up or tie-down device (resistor or transistor) between the chip IO pad and a power supply, Weak_R stores the resistance of this device and Weak_V stores the voltage of the power supply to which the device is connected. A Weak_I stores the approximate current into the device and Weak_V stores 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 referenced 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, reaching 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 a receiver 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, reaching 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 a 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:

$$\frac{\Delta Vth}{\Delta Vsupply}$$

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 (=); white space around the equals sign is optional. The argument to the Reference_supply subparameter is separated from the subparameter by white space.

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 a model to add the functionality that is contained in the submodel or list of submodels in each line that follows. The first column contains the submodel name. 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) have to be modeled as scheduled models.

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 backwards 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

   (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 simulator 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 simulator 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 simulator 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 simulator 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 3 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 simulator pulse rising edge, and f, f1, and f2 are the numerical delay entries relative to internal simulator 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 simulator pulse and delays relative to it.

In Table 3, 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 is not supported.

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 section in 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.

Note: In a future release, the [Driver Schedule] keyword may be replaced by a newer method of specification that is consistent with some other planned extensions. However, the [Driver Schedule] syntax will continue to be supported.

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 preferred 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 is 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] 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: No

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 value or values override any other C_comp* representations. 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 simulator 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 simulator 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 simulator 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
|
```

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 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 white space. All three columns are required under these keywords. However, data is 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 degrees 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.

Example:

```
| 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 model

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 direction is 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 white space.

All four columns are required under these keywords. However, data is 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 is 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 is 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.

An IBIS syntax checking program shall test for non-monotonic data and provide a maximum of one warning per I-V table if non-monotonic data is found. For example:

> "Warning: Line 300, Pulldown I-V table for model DC040403 is non-monotonic! Most simulators will filter this data to remove the non-monotonic data."

It is also recognized that the data may be monotonic if currents from both the output stage and the clamp diode are added together as most simulators do. To limit the complexity of the IBIS syntax checking programs, such programs will conduct monotonicity testing only on one I-V table at a time.

It is intended that 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 simulator 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.

Example:

```
[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 is 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 7. 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 GND (typical) value as shown in Figure 7.

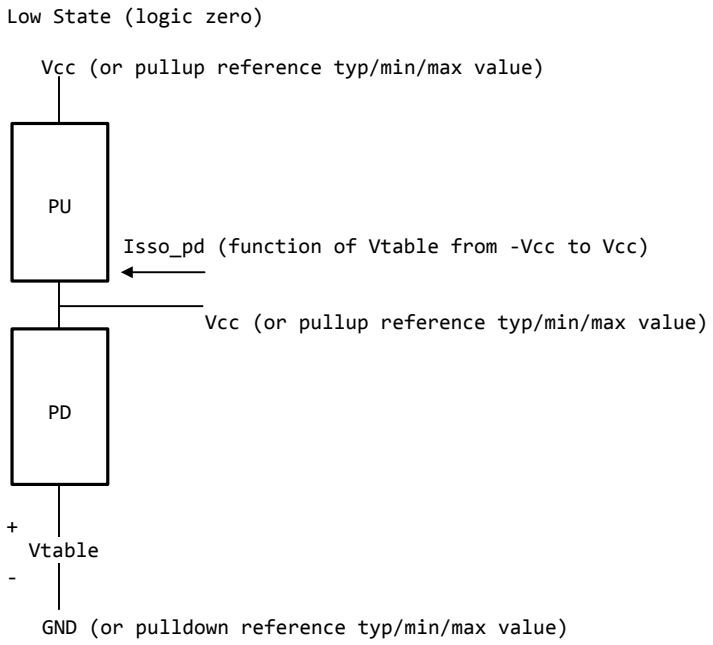


Figure 7 - Low State (Logic Zero) Isso_pd Data Collection

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 below. 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 8.

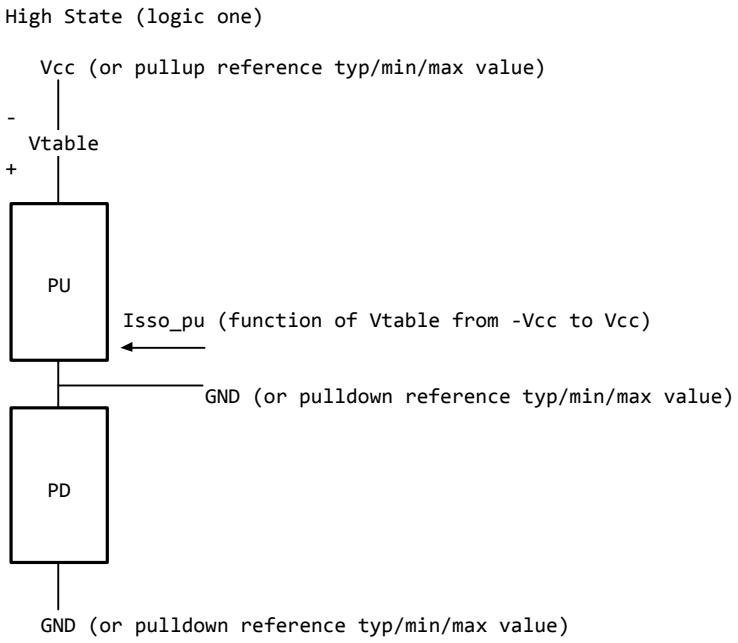


Figure 8 - High State (Logic One) Isso_pu Data Collection

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] selections.

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

Other Notes: Simulators 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 9.

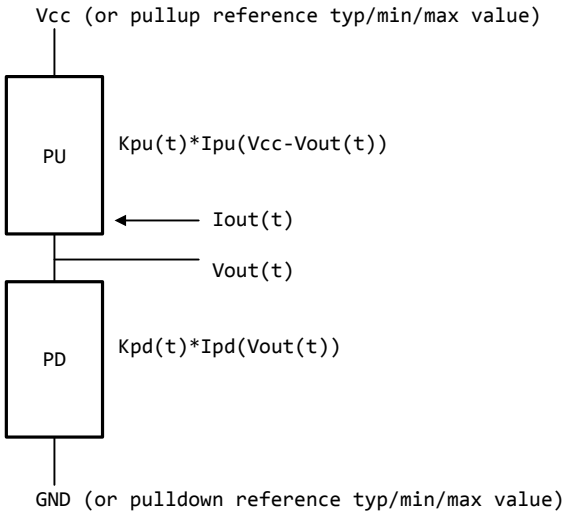


Figure 9 - Reference Data Collection

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 10.

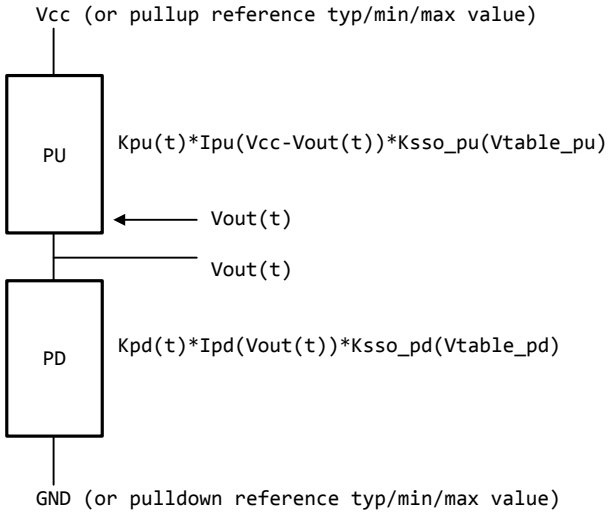


Figure 10 - Reference Data Collection with Supply Modulation

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 are the typ/min/max values for the corresponding typ/min/max columns.

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 3.

Table 3 – 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.

Example:

```
| 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 model

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 11.

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 white space. All three columns are required under these keywords. However, data is 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. When [Rgnd], [Rpower], or [Rac] and [Cac] are specified, the Model_type must be Terminator.

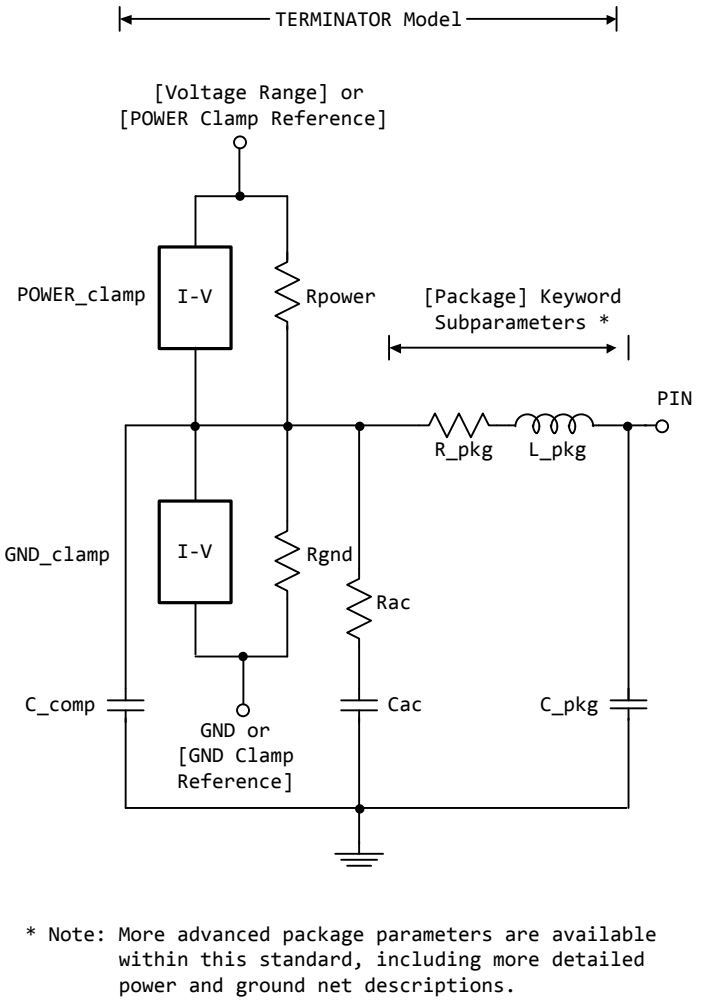


Figure 11 - [Rgnd], [Rpower], [Rac], [Cac] in Relation to Package and Buffer Data

Example:

```
| 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.

Example:

```
[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 model

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 white space. All three columns are required under these keywords. However, data is 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 keyword 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 model is shown in Figure 12.

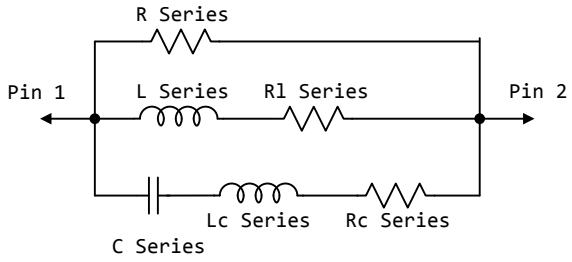


Figure 12 - Series Element Associations

[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.

Example:

```
| 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 model

Description: The data points under this keyword define the I-V tables for 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 white space.

All four columns are required under these keywords. However, data is 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 13.

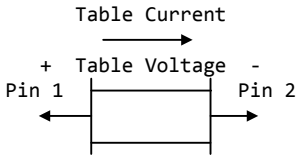


Figure 13 - [Series Current] Voltage Polarity and Current Direction

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 14.

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 white space.

All four columns are required under this keyword. However, data is 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.

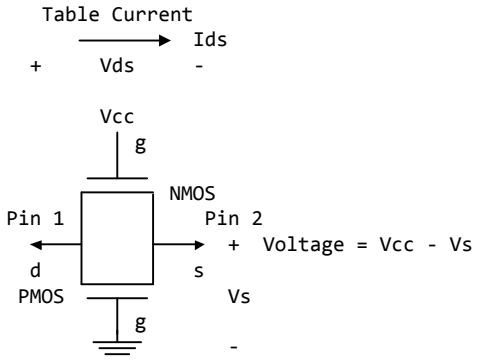


Figure 14 - [Series MOSFET] Voltage Polarities and Current Direction

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 entries are actually Vgs values of the NMOS device and Vcc - Vgs values of the PMOS device, if present. The polarity conventions are identical with those used for other tables that are referenced to power rails. Thus, the voltage column can be viewed as a table defining the source voltages Vs 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 a 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 that would be embedded in the analysis process 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 is 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 simulator 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 time is defined as the time it takes the output to go from 20% to 80% of its final value. The ramp rate is defined as:

$$\frac{\text{—}}{\text{———————————}}$$

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 white space. All four columns are required. However, data is only required in the typical column. If minimum or maximum data is 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 taken 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 15 shows the interconnection of these elements.

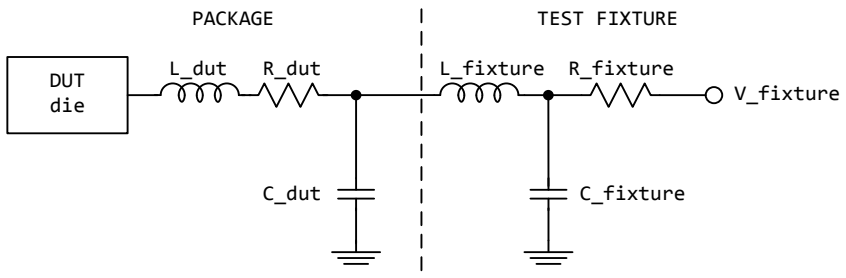


Figure 15 - [Rising Waveform] and [Falling Waveform] Fixtures

NOTE: The use of L_dut, R_dut, and C_dut is strongly discouraged in developing waveform data from simulation models. Some simulators 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 simulators.

Example:

```
[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 white space. All four columns are required. However, data is only required in the typical column. If minimum or maximum data is 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 16 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 pin terminal are used. The absolute GND is the reference for the V_fixture voltage and the package model equivalent network. It can also serve as a reference for C_comp, unless C_comp is optionally split into component attached to the other reference voltages.

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.

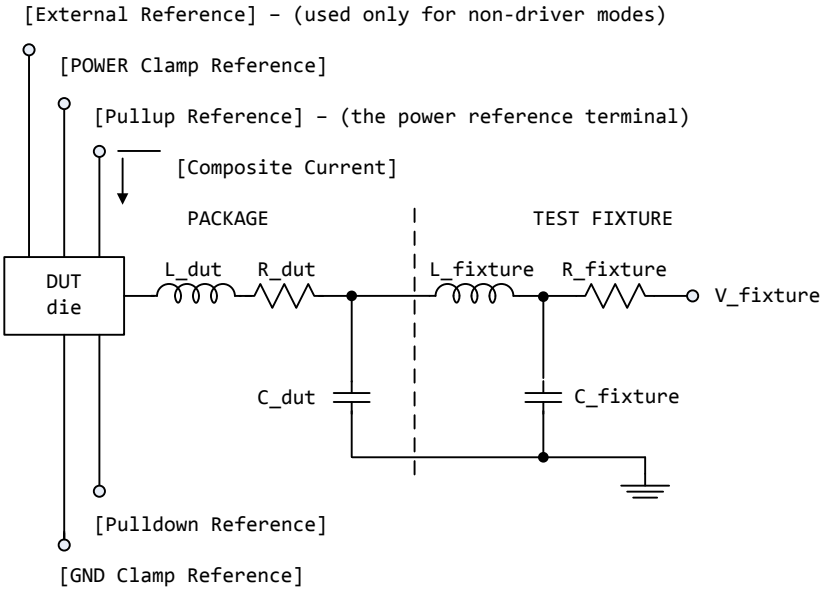


Figure 16 - [External Reference] - (used only for non-driver modes)

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

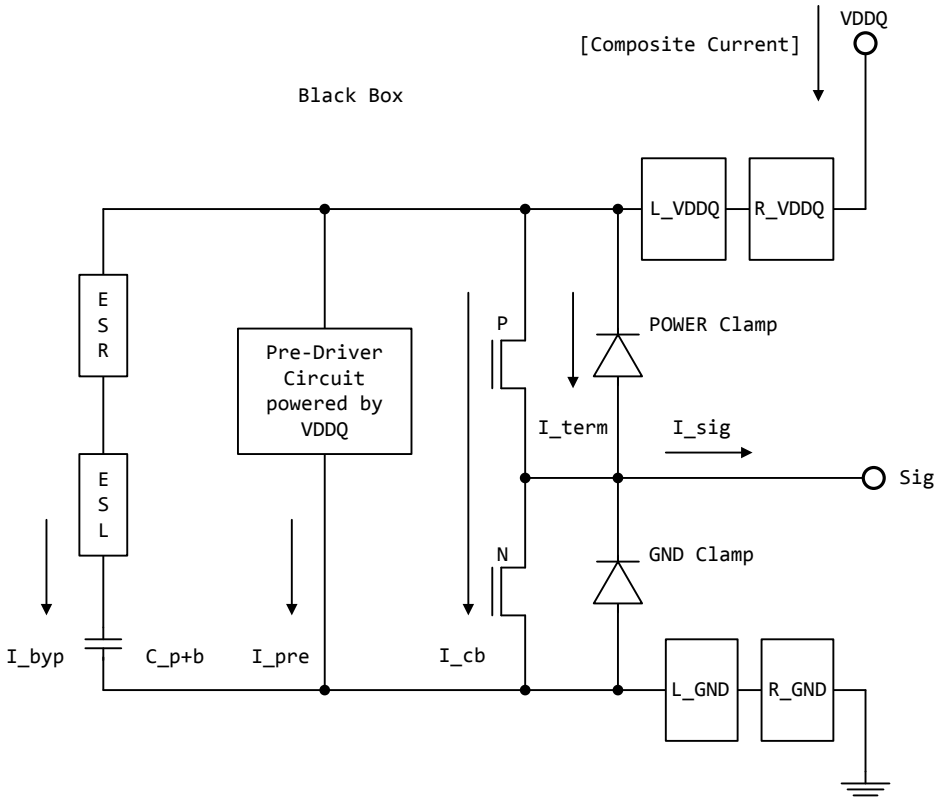


Figure 17 - [Composite Current] Internal Current Paths

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 what 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.
```

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 of 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]

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 must match the 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.

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 are 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], and [Falling Waveform].

The following list of additional keywords can be used:
[Submodel Spec], [GND Pulse Table], and [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 be must be placed on a single line and must be separated by at least one white space. All four columns are required under the [Submodel Spec] keyword. However, data is 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 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.

Example:

```
| 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. An example for a complete dynamic clamp model is provided 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 white space. All four columns are required. However, data is only required in the typical column. If minimum or maximum data is not available, use the reserved word "NA". Time values must increase as one parses down the table. The waveform table can contain of 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 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 die passes the trigger voltage. At that time, [POWER Pulse Table] operation starts. The [GND Pulse Table] dependency is shown in Figure 18.

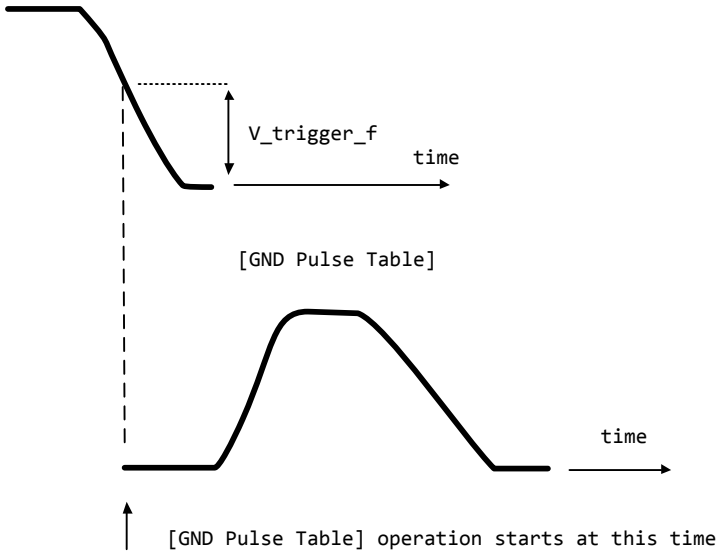


Figure 18 – [GND Pulse Table] Waveforms at Die

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.

Example:

```
| 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 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 voltage 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 4 through Table 7 summarizes the bus hold initializations and switching transitions:

Table 4 – 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
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 5 – 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 6 – Bus Hold with Off_Delay (Requires Either [Pullup] or [Pulldown] Only) - Initialization

[Pullup] or [Pulldown] Table	Initial Bus Hold Submodel State (Off Mode)
[Pullup]	low
[Pulldown]	high

Table 7 – Bus Hold with Off_Delay (Requires Either [Pullup] or [Pulldown] Only) - 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
Note: 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.			

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] keyword, 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 8 through Table 10.

Table 8 – Fall Back, Initial State

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

Table 9 – 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 10 – 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

INTRODUCTION

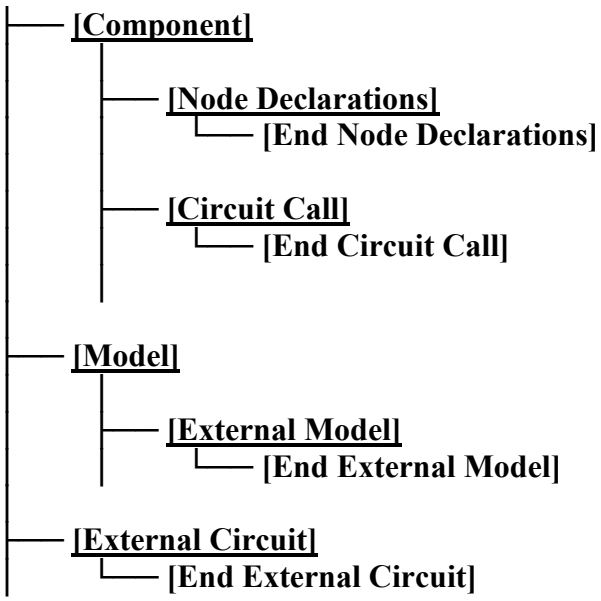
The SPICE, IBIS-ISS, VHDL-AMS and Verilog-AMS languages are supported by IBIS. This chapter describes how models written in these languages can be referenced and used by .ibs files.

Table 11 shows the keywords used by the language extensions within the IBIS framework.

Table 11 – 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:



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.

Overview:

The four keyword pairs discussed in this chapter 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 simulator is expected to carry out the same type of connections and measurements as is 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 the various [External Circuit]s and connecting them together. 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.

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

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 example, the [Voltage Reference] keyword implies the existence of power supply rails which are connected to the power supply ports of the circuit described by the [Model] keyword.

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 12.

Table 12 – 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

Port	Name	Description
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	Global reference voltage port
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 universal 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 reference 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 19 and Figure 20.

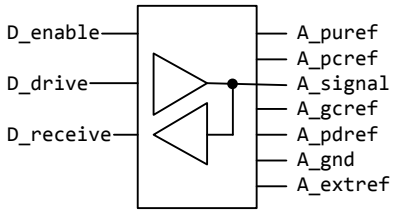


Figure 19 - Port Names for I/O Buffer

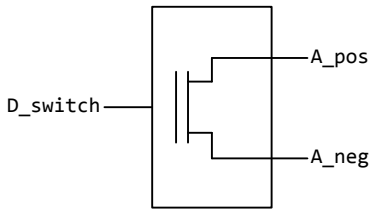


Figure 20 - Port Names for Series Switch

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 21 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.

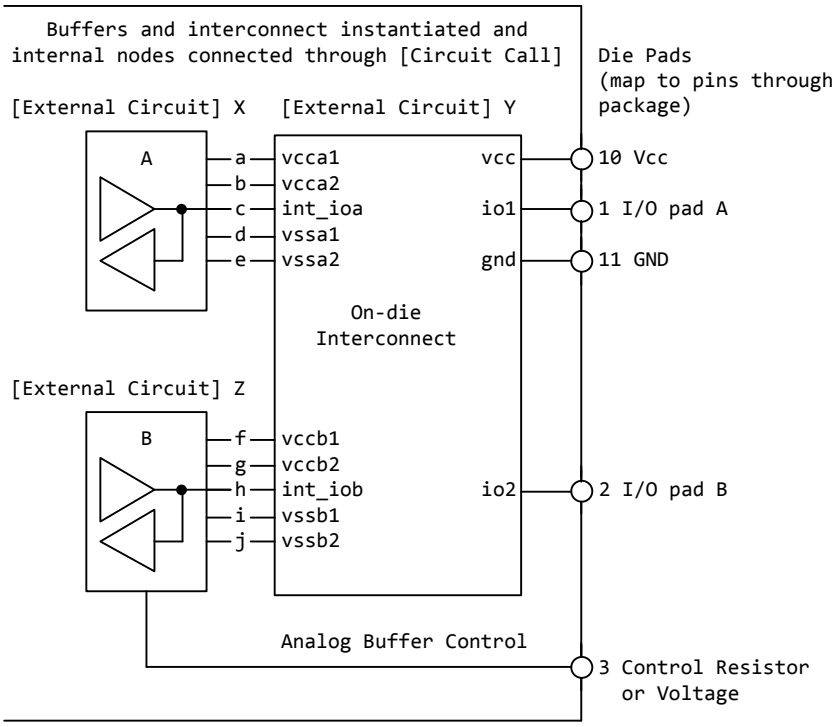


Figure 21 - Example Showing [External Circuit] Ports

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 22 and Figure 23 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.

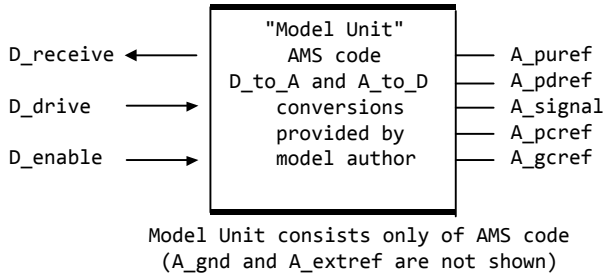


Figure 22 - AMS Model Unit, Using an I/O Buffer as an Example

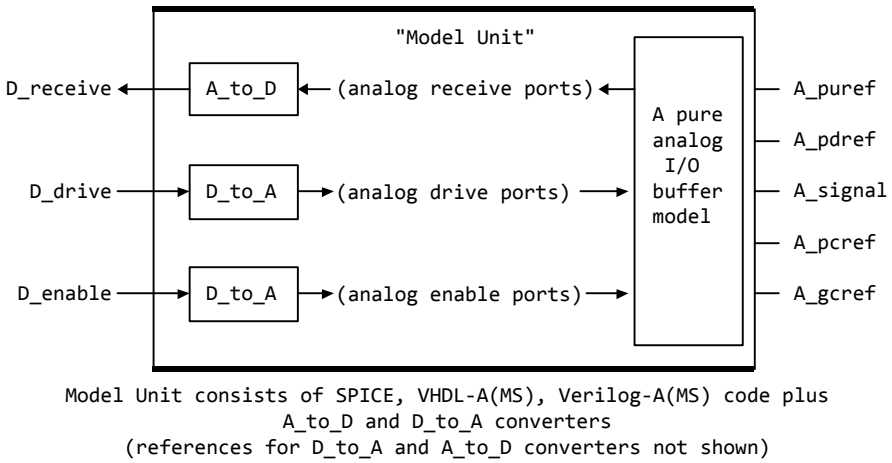


Figure 23 - An Analog-Only Model Unit, Using an I/O Buffer as an Example

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]'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 simulators. 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 [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_name circuit_name

The corner_name entry is "Typ", "Min", or "Max". The file_name entry points to the referenced file in the same directory as the .ibs file.

Up to three Corner lines are permitted. A "Typ" line is required. If "Min" and/or "Max" data is 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_name and circuit_name entries. However, the total number of characters in each Corner line must comply with the rules in Section 3. Furthermore, lower-case file_name entries are recommended to avoid possible conflicts with file naming conventions under different operating systems. Case differences between otherwise identical file_name 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 must 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 must 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 must begin with a file name, followed by an open parentheses and a the tree root name, a new open parentheses 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 files referenced must be located in the same directory as the .ibs file containing the reference. The file names of parameter files must follow the rules for file names given in Section 3, “GENERAL SYNTAX RULES AND GUIDELINES”. Parameter files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

When the extension of the external parameter’s file name ends with “.ami”:

a) only Usage In or Usage Info are allowed for parameters which are to be passed into models instantiated by the [External Model] or the [External Circuit] keywords

When the extension of the external parameter’s file name does not end with “.ami”:

a) the parameter tree must not contain the Reserved_Parameters branch but must contain the Model_Specific branch
b) only Usage Info is allowed

Note that in the case when a parameter is located in an .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 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 must contain one parameter name per line, which must 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 must begin with a file name, followed by an open parentheses and a the tree root name, a new open parentheses 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 files referenced must be located in the same directory as the .ibs file containing the reference. The file names of parameter files must follow the rules for file names given in Section 3, "GENERAL SYNTAX RULES AND GUIDELINES". Parameter files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

When the extension of the external parameter's file name ends with ".ami":

a) only Usage In or Usage Info are allowed for parameters which are to be passed into models instantiated by the [External Model] or the [External Circuit] keywords

When the extension of the external parameter's file name does not end with ".ami":

a) the parameter tree must not contain the Reserved_Parameters branch but must contain the Model_Specific branch
b) only Usage Info is allowed

Note that in the case when a parameter is located in an .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].

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 models 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-pad A_signal response). If at-pad 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.

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 24. 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:

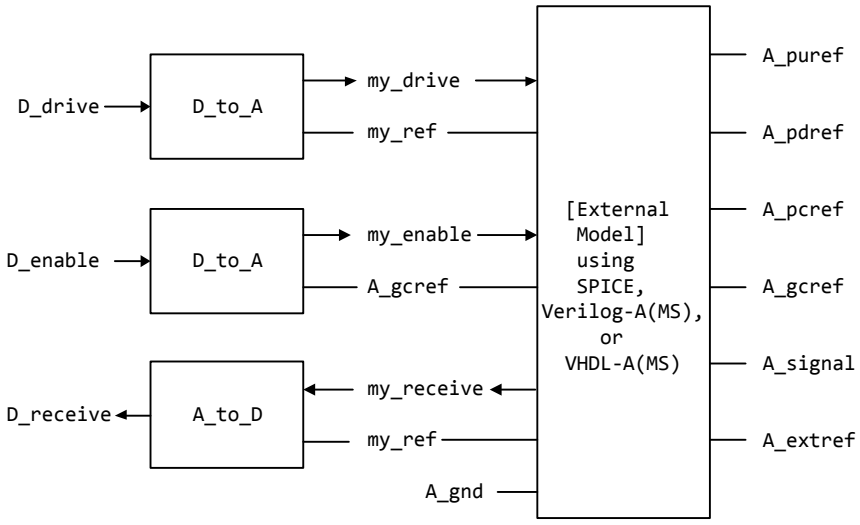


Figure 24 - Example of an [External Model] I/O Buffer Using SPICE, Verilog-A(MS), or VHDL-A(MS)

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.

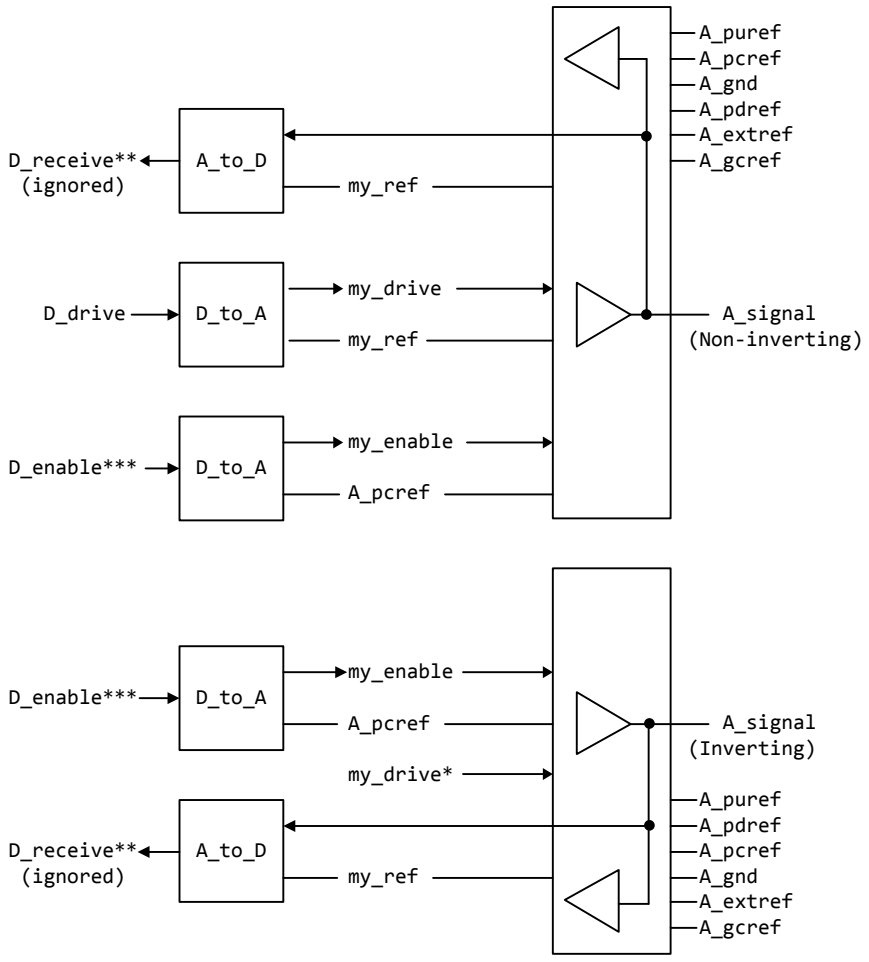
Users should note that, in pseudo-differential buffers, only one formal signal port is used to stimulate the two [External Model] digital inputs (D_drive). One of these inputs will reflect the timing and polarity of the formal signal port named by the user, while the other input is inverted and (potentially) 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 25.



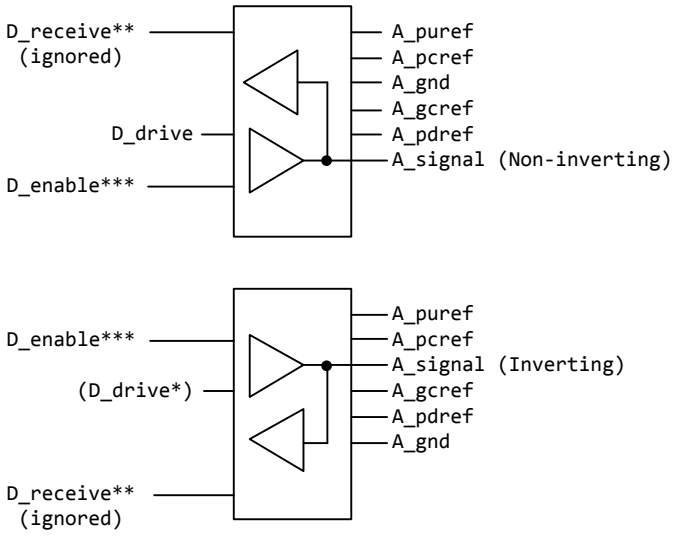
* This signal is automatically created, by inverting and delaying D_drive based on the information in [Diff Pin].

** 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 25 -Example SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) Implementation

Figure 26 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 26 - Example *-AMS Implementation

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 27.

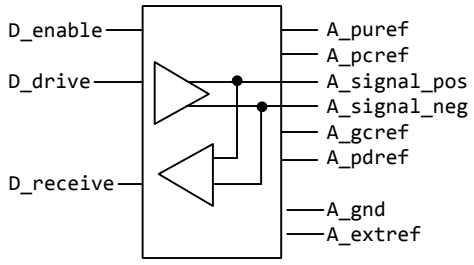


Figure 27 - Port Names for True Differential I/O Buffer

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 28.

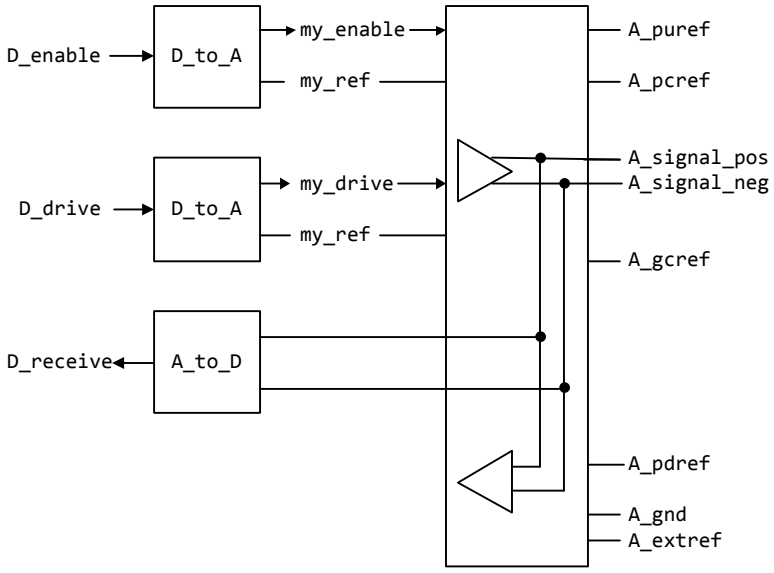


Figure 28 - Example SPICE, IBIS-ISS, Verilog-A(MS) or VHDL-A(MS) Implementation of a True Differential Buffer

If at-pad 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-pad A_signal response). If at-pad 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 user-defined analog signal port would be named in the A_to_D line under port1. 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.

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 13 and Table 14 below define which reserved port names are required for various Model_types.

Table 13 – 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			

Model_type	D_drive	D_enable	D_receive	A_signal	D_switch	A_pos	A_neg
Output*, Open*	X			X			
Input			X	X			
Terminator				X			
Series						X	X
Series_switch					X	X	X

Table 14 – 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(TreeRootName(Model_Specific(TstoneFile)))
Parameters  C1_value
Parameters  R1_value = paramfile.par(TreeRootName(Model_Specific(R1)))
|
| List of converter parameters
Converter_Parameters  MyVlow  = 0.0
Converter_Parameters  MyVHigh  = 3.3
Converter_Parameters  MyVinl  =
paramfile.par(TreeRootName(Model_Specific(Vinl)))
Converter_Parameters  MyVinh  =
paramfile.par(TreeRootName(Model_Specific(Vinh)))
Converter_Parameters   MyTrise = paramfile.par(TreeRootName(Model_Specific(Trf)))
Converter_Parameters   MyTfall = paramfile.par(TreeRootName(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_name circuit_name

The corner_name entry is “Typ”, “Min”, or “Max”. The file_name entry points to the referenced file in the same directory as the .ibs file.

Up to three Corner lines are permitted. A “Typ” line is required. If “Min” and/or “Max” data is 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_name and circuit_name entries. However, the total number of characters in each Corner line must comply with Section 3. Furthermore, lower-case file_name entries are recommended to avoid possible conflicts with file naming conventions under different operating systems. Case differences between otherwise identical file_name 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 must 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 must 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 must begin with a file name, followed by an open parentheses and a the tree root name, a new open parentheses 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 files referenced must be located in the same directory as the .ibs file containing the reference. The file names of parameter files must follow the rules for file names given in Section 3, "GENERAL SYNTAX RULES AND GUIDELINES". Parameter files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

When the extension of the external parameter's file name ends with ".ami":

a) only Usage In or Usage Info are allowed for parameters which are to be passed into models instantiated by the [External Model] or the [External Circuit] keywords

When the extension of the external parameter's file name does not end with ".ami":

a) the parameter tree must not contain the Reserved_Parameters branch but must contain the Model_Specific branch
b) only Usage Info is allowed

Note that in the case when a parameter is located in an .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 must contain one parameter name per line, which must 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 must begin with a file name, followed by an open parentheses and a the tree root name, a new open parentheses 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 files referenced must be located in the same directory as the .ibs file containing the reference. The file names of parameter files must follow the rules for file names given in Section 3, “GENERAL SYNTAX RULES AND GUIDELINES”. Parameter files may only contain parameter trees using the tree syntax described in IBIS in Section 10.3 with the following exceptions and additions:

When the extension of the external parameter’s file name ends with “.ami”:

a) only Usage In or Usage Info are allowed for parameters which are to be passed into models instantiated by the [External Model] or the [External Circuit] keywords

When the extension of the external parameter’s file name does not end with “.ami”:

a) the parameter tree must not contain the Reserved_Parameters branch but must contain the Model_Specific branch
b) only Usage Info is allowed

Note that in the case when a parameter is located in an .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].

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.

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 port names in 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. he 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 A_signal response). If native IBIS 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 analog 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 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_name       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_name       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(TreeRootName(Model_Specific(TstoneFile)))
Parameters  C1_value
Parameters  R1_value = paramfile.par(TreeRootName(Model_Specific(R1)))
|
Converter_Parameters  MyVlow  = 0.0
Converter_Parameters  MyVHigh  = 3.3
Converter_Parameters  MyVinl  =
paramfile.par(TreeRootName(Model_Specific(Vinl)))
Converter_Parameters  MyVinh  =
paramfile.par(TreeRootName(Model_Specific(Vinh)))
Converter_Parameters   MyTrise = paramfile.par(TreeRootName(Model_Specific(Trf)))
Converter_Parameters   MyTfall = paramfile.par(TreeRootName(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_name       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_name     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_name   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_name   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_name  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_name 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 white space. 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
|
| Die pads:
pad_2a pad_2b pad_4 pad_11   | List of die pads
|
[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, 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 simulator. 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.

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, die pad, 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.

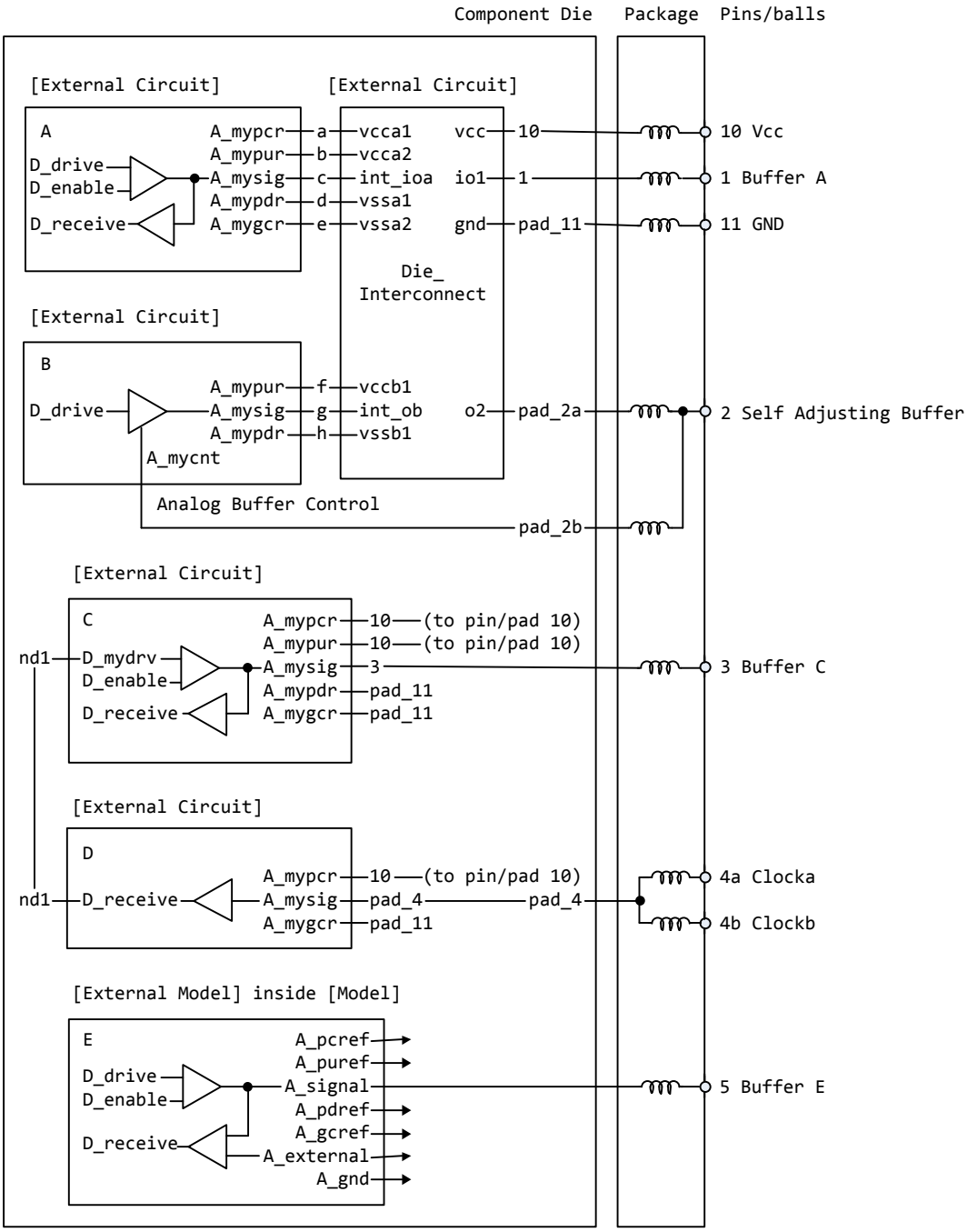
Please note that a pin name in the second argument does not mean that the connection is made directly to the pin. Since native IBIS does not have a mechanism to declare die pads explicitly, connections to die pads are made through their corresponding pin names (listed under the [Pin] keyword). This convention must only be used with native IBIS package models where a one-to-one path between the die pads and pins is assumed. When a package model other than native IBIS is used with a [Component], the second argument of Port_map must have a die pad or die node name. These names are matched to the corresponding port name of the non-native package model by name (not by position). In this case, the package model may have an arbitrary circuit topology between the die pads and the pins. A one-to-one mapping is not required.

Examples:

NOTE REGARDING THIS EXAMPLE:

The pad_* to pin connections in Figure 29 and in the example lines with the comment, "explicit pad connection", are shown for reference. The connection syntax has not yet been defined. Therefore, the connections for pad_* to pin are not supported in this specification.

For the examples below please refer to Figure 29 and the example provided for the [Node Declarations] keyword.



Notes:
1) The ports of the [External Model] E are automatically connected by the tool, taking the [Pin Mapping] keyword into consideration, if exists.
2) The package model shown in this drawing assumes the capabilities of a non-native IBIS package model are available to the model author.

Figure 29 - 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       pad_2b   | Port to explicit 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       pad_11   | Port to explicit pad connection
Port_map   A_mygcr       pad_11   | Port to explicit 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 4a
|
| mapping  port          pad/node
|
Port_map   A_my_pcref    10       | Port to implicit pad connection
Port_map   A_my_signal   pad_4    | Port to explicit pad connection
Port_map   A_my_gcref    pad_11   | Port to explicit 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"
|
| mapping  port          pad/node
|
Port_map   vcc           10       | Port to implicit pad connection
Port_map   gnd           pad_11   | Port to explicit pad connection
Port_map   io1           1        | Port to implicit pad connection
Port_map   o2            pad_2a   | Port to explicit 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

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 simulator implementation.

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 simulator 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.

Example:

```
[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 30 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.

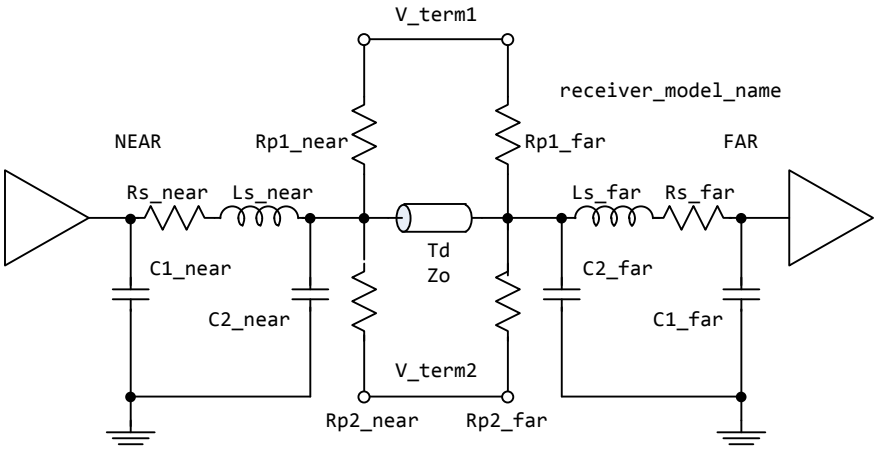


Figure 30 - [Test Load] Elements and Placement

If the Td subparameter is present, then the Zo subparameter must also be present. If the Td subparameter is not present, then the simulator 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.

Example:

```
[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

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 <package_file_name>.pkg file or can exist in the .ibs files between the [Define Package Model] ... [End Package Model] keywords for each package model that is defined. For reference, these keywords are listed in Table 15. Full descriptions follow. EDA tools that do not support these keywords will ignore all entries between the [Define Package Model] and [End Package Model] keywords.

Table 15 – 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)
[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)
Note 1 Required when the [Define Package Model] keyword is used Note 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]	

Keyword	Notes
keywords are mutually exclusive. Note 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 globally 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:

<filename>.pkg.

The <filename> provided must adhere to the rules given in Section 3, "GENERAL SYNTAX RULES AND GUIDELINES". Use the “.pkg” extension to identify files containing package models. The .pkg file must 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 in length. 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]  Quality Semiconductors Ltd.
```

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 must be less than 60 characters in length, must 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 bondwire, the connection between the bondwire 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: Tells the parser how many pins to expect.

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: Tells the parser the set of names that are used for the package pins and also 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 there are pins (as given by the preceding [Number Of Pins] keyword). Pin names cannot exceed 5 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.

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 (=); white space 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 indentation of the Fork and Endfork subparameters are for readability
| are not required.
|
A1 Len=0 L=2.3n /         | bondwire
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 /        | bondwire
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: **[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 is 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.

Sub-Params: Banded_matrix, Sparse_matrix, or Full_matrix

Description: The subparameters mark the beginning of a matrix, and specify how the matrix data is formatted. See Figure 31.

Usage Rules: For each matrix keyword, use only one of the subparameters. 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 (rule #11 in Section 3, "GENERAL SYNTAX RULES AND GUIDELINES"). The corresponding voltage drops are to be measured with the current pointing "in" to the "+" sign and "out" of the "-" sign.

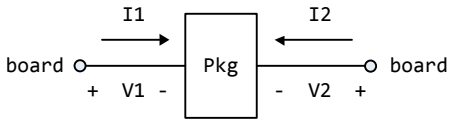


Figure 31 - Package Matrix Voltage Polarities and Current Directions

It is important to observe this convention in order to get the correct signs for the mutual inductances and resistances.

Example:

```
[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.

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. 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.

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 120 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 is 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 120-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]      6.0
[File Name]     example.pkg
[File Rev]      0.1
[Date]          September 20, 2013
[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]          Quality Semiconductors Ltd.
[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

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 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 or included in any .ibs or .pkg file. Electrical Board Descriptions are stored in a file whose name looks like <filename>.ebd, where <filename> must conform to the naming rules given in Section 3 of this specification. The .ebd extension is mandatory.

Contents:

An .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.

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 in length, 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] Quality SIMM Corp.
```

Keyword: **[Number Of Pins]**

Required: Yes

Description: Tells the parser the number of pins to expect. Pins are any externally accessible electrical connection to the component.

Usage Rules: The field must be a positive decimal integer. Note: The simulator 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: Tells the parser the pin names of the user accessible pins. It also informs the parser 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 must be as many pin_name/signal_name rows as there are pins given by the preceding [Number Of Pins] keyword. Pin names must be the alphanumeric external pin names of the part. The pin names cannot exceed eight characters in length. Any pin associated with a signal name that begins with "GND" or "POWER" will be interpreted as connecting to the boards ground or power plane. In

addition, NC is a legal signal name and indicates that the Pin is a “no connect”. As per the IBIS standard “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 simulator.

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 reference_designator.pin
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 (".") . 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 MUST 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 (=); white space 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 white space.

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 32):

```
|
[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
```

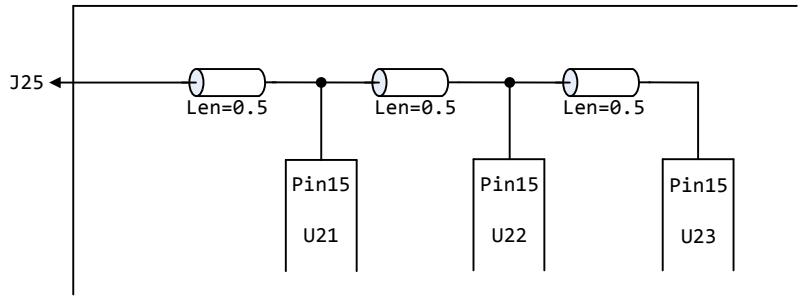


Figure 32 - SIMM Package Path Example

A Description Using The Fork and Endfork Subparameters (see Figure 33):

```
|
[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
|
```

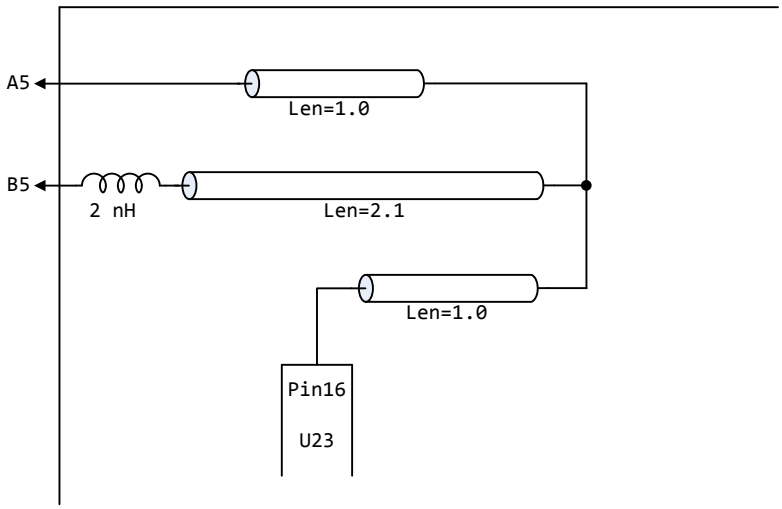


Figure 33 - Fork and Endfork in [Path Description]

A Description Including a Discrete Series Element (see Figure 34):

```
|
[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
```

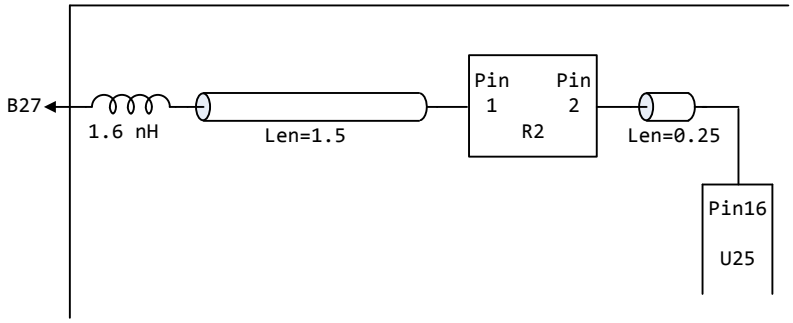


Figure 34 – Discrete Series Element in [Path Description]

A path including series passive components (C17, R21) between branches forming a differential termination (see Figure 35):

```
[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
```

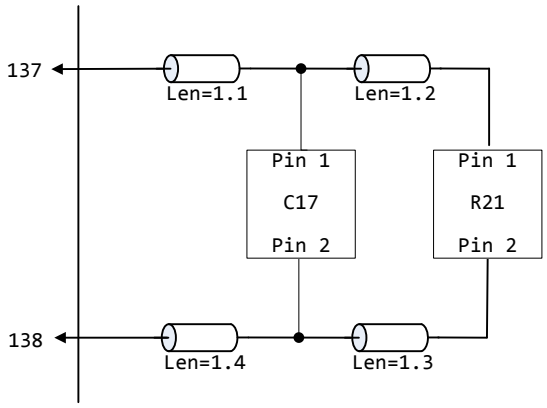


Figure 35 – Series Passive Components as Differential Termination

Two paths connected by series resistors (R8, R9) used as differential termination between components:

```
[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
```

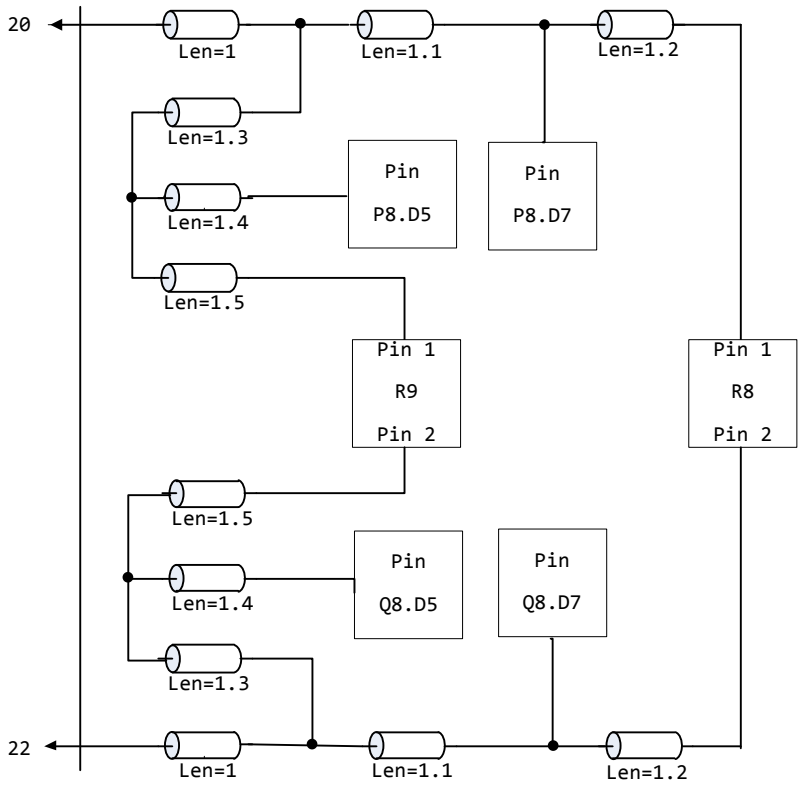


Figure 36 – Paths Connected by Series Resistors as Differential Terminators

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 must 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 name 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 white space. By default the .ibs or .ebd files are assumed to exist in the same directory as the calling .ebd file. 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 name   Component name
u23        pp100.ibs   Processor
u24        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 interconnect model description.

Optionally, a comment may be added after the [End Electrical Description] keyword to clarify which board model has ended.

Example:

```
[End Board Description]        | End: 16Meg X 8 SIMM Module
```

Keyword: **[End]**

Required: Yes

Description: Defines the end of the .ibs, .pkg, or .ebd file.

Example:

```
[End]
```

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 fab 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 16.

Table 16 – 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 simulator 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 load subparameter to the [Ramp] specification.

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 simulator 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 derate 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 37.

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 - 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).

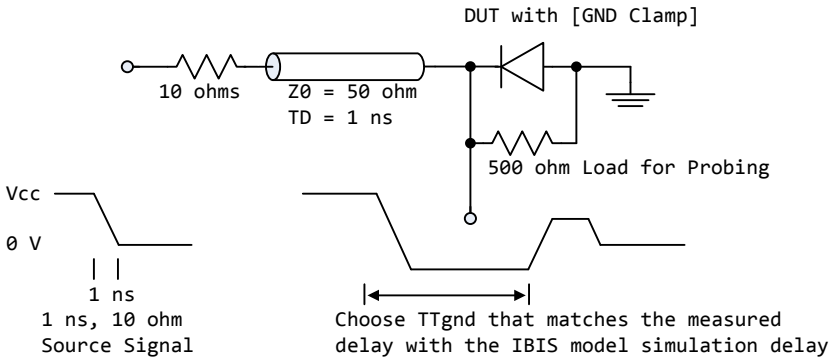


Figure 37 - Example of TTgnd Extraction Setup

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 38. 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.

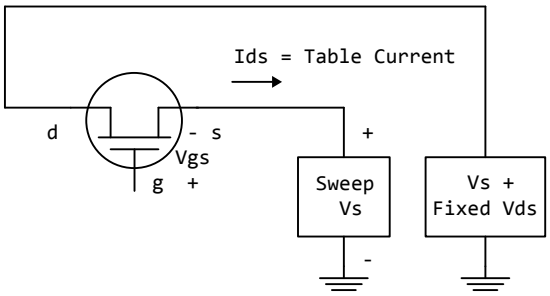


Figure 38 - Example of Series MOSFET Table Extraction

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)

INTRODUCTION

Algorithmic modeling of advanced Serializer-Deserializer (SERDES) devices is supported by IBIS, through the Algorithmic Modeling Interface (AMI). 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 are 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
- a 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

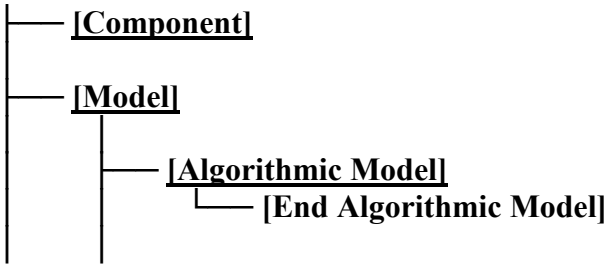
Both of these files are provided by the SERDES device vendor.

This section defines how the components of an algorithmic model are specified in an .ibs file. 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 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:



KEYWORD DEFINITIONS

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

Required: No

Description: Used to reference an executable model file and accompanying parameter definition file. This executable model file encapsulates signal processing functions, while the 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 with 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

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 File_Name 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 white spaces, 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 must 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 operating system and platform.

Multiple occurrences, without duplication, of Executable are permitted to allow for providing executable model files for as many combinations of operating system platforms and compilers for the same algorithmic model.

The File_Name provides the name of the executable model file. The executable model file should be in the same directory as the.ibs file.

The Parameter_File entry provides the name of the parameter definition file, which shall have an extension of .ami. This must be an external file and should reside in the same directory as the .ibs file and the executable model file. See Section 10.3 for details.

Examples:

Example of Receiver Model in [Algorithmic Model]:

```
[Algorithmic Model]
|
Executable Windows_VisualStudio_32 example_rx.dll example_rx_params.ami
|
[End Algorithmic Model]
```

Example of Transmitter Model in [Algorithmic Model]:

```
[Algorithmic Model]
|
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]
```

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 executable model file of a Serializer-Deserializer (SERDES) transmitter or receiver contains up to three functions: "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_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. If they are not supplied in the executable model file named by the Executable sub-parameter, then they shall be ignored. This is acceptable so long as:

1. The entire functionality of the model is supplied in the executable model file.
2. All termination actions required by the model are included in the executable model file.

The three functions can be included in the executable model file in one of the following two 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.

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 two 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.

10.2.2.1 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 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 in 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.

10.2.2.2 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 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 by the EDA tool. For example, if one million bits are to be simulated, there can be 1000 segments of 1000 bits each, i.e., one time segment comprises 1000 bits. 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 in 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 in 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.

10.2.2.3 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.

10.2.2.3.1 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.

10.2.2.3.2 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.

Under certain circumstances, for example when the Rx AMI_Init function includes an optimization algorithm, the impulse response presented to the Rx AMI_Init function must include the Tx equalization effects for the optimization to work correctly. However, when the Tx AMI model contains an AMI_GetWave function that performs a similar or better equalization than the Tx

AMI_Init function, there is a possibility for "double-counting" the equalization effects in the Tx executable model file. To allow for such models to work correctly, the EDA tool can operate in one of several ways, two of which are documented here:

- not utilize the Tx AMI_GetWave functionality, by treating the Tx AMI model as if the Tx GetWave_Exists was False.
- use deconvolution to obtain the impulse response of the Rx filter. Since the AMI_Init function contains a linear and time invariant algorithm, the Rx equalization can be represented as an impulse response. Since the output of the Rx AMI_Init function (output of Step 3) is an impulse response modified by the Rx 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 can be obtained by deconvolving the output of Step 3 with the input presented to Step 3.

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 output of Step 5 is convolved with the output of Step 1 and the Impulse Response of the Rx filter by the EDA tool and the result is passed on to Step 8. (The Impulse Response of the Rx filter may be obtained by deconvolving the output of Step 3 by the input of Step 3).

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 ticks, 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. Splitting up the full analog waveform into multiple calls reduces the memory requirements when doing long simulations, and allows AMI_GetWave to return model status every so many bits. 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.

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 bit_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 trough 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.

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 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 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 bit_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.

Example:

```
Sample_interval = (lowest_bit_time / 64)
```

bit_time

bit_time is the bit time or unit interval (UI) of the current data, e.g., 100 ps, 200 ps etc. The executable model file may use this information along with the impulse_matrix to initialize the filter coefficients. The unit for bit_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. All the input from the parameter definition file is passed to the algorithmic model using a string that has been formatted as using 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, even if there are no parameters to pass to the algorithmic model.

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 white space characters.
3. Parameter name/value pairs are always enclosed in parentheses, with the value separated from the name by white space.
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 must be a parameter group.
8. White space 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 white space.

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.

While the AMI_parameters_out 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 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_handle

Used to point to local storage for the algorithmic block being modeled and shall be passed back during the AMI_GetWave calls. e.g., a code snippet may look like the following:

```
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_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_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 trough 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 simulator such as SPICE.

It is assumed that the electrical interface to either the driver or the receiver is differential. Therefore, the sample values are assumed to be differential voltages centered nominally around zero volts. The algorithmic model's logic threshold may be non-zero, for example to model the differential offset of a receiver; 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"). 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. The clock times are referenced to the start of the simulation (the first AMI_GetWave call). The clock_times vector is allocated by the EDA tool and is guaranteed to be greater than the number of clocks expected during the AMI_GetWave call. The clock times are exactly bit_time/2 before the input data signal is sampled. The algorithmic model will return non-negative clock_times values, and place -1 after the last valid clock tick in the clock_times vector during each AMI_GetWave call. If there are no valid clock ticks for the duration of an AMI_GetWave call, a single entry of -1 will be returned in the clock_times vector. The units of clock_times are seconds.

The clock ticks represented by clock times 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-strict-monotonic behavior of clock times (including two identical values) should be considered by EDA tool as an algorithmic model failure.

Each valid value in the clock_times vector shall be used to sample the output waveform by adding to it bit_time/2, regardless 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 insure that the calculations used always maintain full double-precision floating point accuracy across multi-million bit simulations.

Although clock_times will generally be related to the unit interval for the primary SerDes channel being simulated, there is no requirement that there be any relationship between the clock ticks generated by clock_times and the actual waveform returned in the primary channel. It is possible for the CDR to go out of lock, resulting in clock ticks that have no definite relationship to the output wave. It is possible for the CDR to be suppressed for an undefined number of bits until the output of the 1st clock tick. In the case of a receiver without a CDR, it is possible for only -1 to ever be output during all AMI_GetWave calls.

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.

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

INTRODUCTION

The information provided in this section is applicable to the content of the AMI parameter definition file (.ami file; hereinafter, parameter definition file). Note that the rules described below deviate from the rules for .ibs files

PARAMETER DEFINITION FILE ORGANIZATION

The parameter definition file is organized as a "tree", with "leaves" and "branches" off a single "root" identified by a unique 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 parameter definition file shall be organized in the following way:

```
(my_AMIname                    | Root name given to the Parameter 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 parameter file
```

General Rules:

- The content of the 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 parameter definition file is not limited to a specific number of characters.
- The root name in the file may contain an arbitrary string and does not need to match the file name.
- A white space in the parameter definition file may be one or more space, tab, and/or line termination characters.

- The “Reserved_Parameter” section is required while the “Model_Specific” section is optional.
- For AMI_Version 5.1 and above, the Reserved_Parameter branch shall appear before the Model_Specific branch. Branches may be in any order in the 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 Parameter definition file.
- Scientific and floating point notation is permitted.
- Note that Description is considered a leaf that may be optionally used within the individual “Reserved_Parameters” or “Model_Specfiic” 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.

PARAMETER RULES SUMMARY

The features of a model described in a 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 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 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> selection 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.

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 model

Info
Information for user or EDA tool

InOut

Parameter value is a required input to the AMI model. The AMI model may return a different value.

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 quotes ("). As defined in ANSI Standard X3.4-1986, the allowable ASCII characters consist of hexadecimal 20, 21, 23 to 7E, and the ASCII control characters 09 (HT), 0A (LF), and 0D (CR) for defining tabs and line termination sequences. The double quote character 22 (") 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 data rate frequency, for example 1 UI of a channel operating at 10 Gb/s is 100 ps. UI parameter values are in units of UI (bit time). The parameter may take on either floating point or integer values.

Format <data_format> <data> or **<data_format>**<data>:

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. Valid entries for the <data_format> and <data> fields are:

Value <value>
Single value data. 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 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

List <default value> <value> <value> <value> ... <value>
This defines a discrete set of values from which the user may select one value

List_Tip <default_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. 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>
where min <= typ <= max and delta is always positive. After expansion, the expanded values of the parameter are typ + N*delta where N is any positive or negative integer value provided by the EDA tool during the expansion process so that: min <= expanded values <= max

Steps <typ> <min> <max> <# steps>
Treat 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 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 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 file 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 file.

Examples:

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 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 and random jitter 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 120 characters per line.

COMBINATION AND CORNER RULES

For Usage Out parameters, ({Format} <data_format> <data>) may be ignored by the EDA tool, except when <data_format> is Table where at least a one-row Table is required in <data> to serve as a template for single and multi-row tables.

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 AMI parameters defined as Format Corner, the EDA tool will pick one of the three supplied values (<typ value>, <slow value>, <fast value>) in the 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 Types “Range”, “Increment” and “Steps” <min value>, <max value> does not imply slow and fast corners, and 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 types “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.

Table 17 summarizes the relationships between the different Format and Data Types for Reserved or Model Specific Parameters.

Table 17 – 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

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 below) is formatted the same way as the tree data structure in the parameter definition file with the following exceptions.

The EDA tool shall process the content of the 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 white space, a value and a closing parenthesis ")"

The executable model shall generate a parameter string that is consistent with the content of the 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 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 parameter definition file. For Usage Out, the value 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 parameter definition file and passed into the Algorithmic Model which may return a new value in the AMI parameter leaves after some processing.

GENERAL RESERVED PARAMETERS

The parameter definition file shall have a branch with the heading “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

Descriptors:

Usage:	Info
Type:	String
Format:	Value

Default: <string_literal>
Description: <string>

Definition: Tells EDA tool what version of the AMI modeling language is supported.

Usage Rules: AMI_Version is required in the 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 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 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 and "6.0" for AMI models written in compliance with the IBIS Version 6.0 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.

Examples:

```
(AMI_Version (Usage Info) (Type String) (Value "5.1")
      (Description "Valid for AMI_Version 5.1 and above")
)
(AMI_Version (Usage Info) (Type String) (Default "6.0")
      (Description "Valid for AMI_Version 6.0")
)
```

Parameter: **Init_Returns_Impulse**

Required: Yes

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells 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

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells 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: **Use_Init_Output**

Required: No, and legal only before AMI_Version 5.1

Descriptors:

Usage: Info
Type: Boolean
Format: Value
Default: <Boolean_literal>
Description: <string>

Definition: Tells 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:

Examples:

```
(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

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

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 bits 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")
)
```

Table 18 – General Rules and Allowable Usage for General Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage			
	Required	Default	Info	In	Out	InOut
AMI_Version[1]	Yes	--	X			
GetWave_Exists	Yes	--	X			
Ignore_Bits	No	0	X			
Init_Returns_Impulse	Yes	--	X			
Max_Init_Aggressors	No	0	X			
Use_Init_Output[2]	No	True	X			

1) Required for AMI_Version 5.1 and later, and illegal before AMI_Version 5.1
2) Illegal for AMI_Version 5.1 and later

Table 19 – Allowable Data Types for General Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
AMI_Version[1]				X	
GetWave_Exists					X
Ignore_Bits			X		
Init_Returns_Impulse					X
Max_Init_Aggressors			X		
Use_Init_Output[2]					X

1) Required for AMI_Version 5.1 and later, and illegal before AMI_Version 5.1
2) Illegal for AMI_Version 5.1 and later

Table 20 – Allowable Data Formats for General Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
AMI_Version[1]	X									
GetWave_Exists	X									
Ignore_Bits	X									
Init_Returns_Impulse	X									
Max_Init_Aggressors	X									
Use_Init_Output[2]	X									

1) Required for AMI_Version 5.1 and later, and illegal before AMI_Version 5.1
2) Illegal for AMI_Version 5.1 and later

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)
      )
    )
  )
)
```

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                            | Parameter Definition File name
  (Description "Sample AMI File")
  (Reserved_Parameters                  | Required heading
    (AMI_Version (Usage Info) (Type String) (Value "6.0")
      (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
```

10.4 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

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 IBIS-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 must be expressed relative to the location of the .ami file in which the Supporting_Files parameter is found. (The AMI executable models and the parameter files are all required to be in the same directory as the .ibs file in which they are declared). 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

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 or simulator 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

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 a unique alphanumeric identifier. The algorithmic model is responsible for using DLL_ID as the base 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:

Other Notes:

Example:

```
DLL_ID (Usage In) (Type String) (Value "placeholder")
       (Description "Unique base name for each AMI model instance and run"))
```

Tables summarizing the reserved parameters for supporting files are shown below.

Table 21 – General Rules and Allowable Usage for Supporting Files Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage			
	Required	Default	Info	In	Out	InOut
DLL_ID	No	No DLL_ID		X		
DLL_Path	No	No DLL_Path		X		
Supporting_Files	No	None	X			

Table 22 – 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 23 – Allowable Data Formats for Supporting Files Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table

Reserved Parameter	Data Format									
DLL_ID	X									
DLL_Path	X									
Supporting_Files										X

10.5 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 the values returned by the AMI_Init function. 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.

Tx-only Reserved Parameters

These Reserved Parameters only apply to Tx algorithmic models. There parameters are optional. If these parameters are not specified, the values default to no jitter specified in the model ("0" jitter).

Parameter: **Tx_Jitter**

Required: No

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Gaussian, Dual-Dirac, DjRj, Table
Default:	(Illegal)
Description:	<string>

Definition: Tells 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 bit 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

Descriptors:

Usage:	Info, Out
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.

$$Time(n) = n * bit\_time + Tx\_DCD * (-1.0)^n$$

Where:

- n*bit_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:

Examples:

```
(Tx_DCD (Usage Info) (Type Float)
      (Range 2e-12 1e-12 3e-12))
```

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). This data is used by the simulator 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.

Parameter: **Tx_Rj**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage:	Info, Out
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:

$$Time(n) = n * bit\_time + Tx\_Rj * gaussian\_rand()$$

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

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The worst case half the peak to peak variation at the transmitter implemented by the EDA tool 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 * bit_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

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half 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 * bit_time + Tx_Sj * sin((n * bit_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

Descriptors:

Usage:	Info, Out
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 * bit_time + Tx_Sj * sin((n * bit_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."))
```

Rx-only Reserved Parameters

These Reserved Parameters only apply to Rx algorithmic models. These parameters are optional. If the parameters are not specified, the values default to "0".

Parameter: **Rx_Clock_PDF**

Required: No

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Gaussian, Dual-Dirac, DjRj, Table
Default:	(Illegal)
Description:	<string>

Definition: Tells 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 bit 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, 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_Receiver_Sensitivity**

Required: No

Descriptors:

Usage:	Info, Out
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 to ensure proper sampling of the equalized signal.

Usage Rules: Entries are assumed to be in units of volts.

Other Notes:

Examples:

In the example below, 100 mV (above +100 mV or below -100 mV is needed to ensure the signal is sampled correctly).

```
(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 0.0 0.1 -0.1))
```

The following optional Reserved Parameters are used to specify characteristics of the receiver's recovered clock. This data is used by the simulator 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 platform should NOT apply these jitter parameters again to the Rx clock_times. These parameters are provided by the model creator to the EDA platform 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 platform is responsible to apply these jitters to the Rx output."

"Rx_Clock_Recovery_Mean" is an AMI parameter of Type either Float or UI, Format either Value, List, Range, Corner, Increment, or Steps, and Usage Info which defines a static offset, in seconds or UI, between the recovered clock and the point half way between the PDF medians of consecutive eye zero crossings.

Parameter: **Rx_Clock_Recovery_Mean**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage: Info, Out
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 eye zero crossings. Entries are assumed to be in units of seconds when declared as Type Float.

Usage Rules:

Other Notes: Time is calculated as follows:

actual_time = ideal_time + Rx_Clock_Recovery_Mean

Where ideal_time is half way between the median of the eye crossing 0.0 on both sides of the eye.

Examples:

```
(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

Descriptors:

Usage: Info, Out
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:

*actual_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

Descriptors:

Usage: Info, Out
Type: Float, UI
Format: Value, List, Range, Corner, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: The worst case half the 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:

*actual_time = ideal_time + 2.0 * Rx_Clock_Recovery_Dj * rand()*

Example:

```
(Rx_Clock_Recovery_Dj (Usage Info) (Value 0.1) (Type UI)
         (Description "Tx Bounded Jitter in UI."))
```

Parameter: **Rx_Clock_Recovery_Sj**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage: Info, Out
Type: Float, UI
Format: Value, List, Range, Corner, Increment, Steps
Default: <numeric_literal>
Description: <string>

Definition: Half 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:

*actual_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

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half 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:

$$actual\_time = ideal\_time + Rx\_Clock\_Recovery\_DCD * (-1.0)^n$$

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 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:

$$\begin{aligned} actual\_time = ideal\_time &+ 2.0 * Rx\_Clock\_Recovery\_Dj * rand() \\ &+ Rx\_Clock\_Recovery\_Sj * sin(Pi * rand()) \\ &+ Rx\_Clock\_Recovery\_DCD * (-1.0)^n \\ &+ <deterministic\ contribution\ from\ Rx\_Clock\_PDF> \end{aligned}$$

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. This data is used by the simulator 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_Rj**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage:	Info, Out
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

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The worst case half 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:

*actual_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 "Tx Bounded Jitter in UI."))
```

Parameter: **Rx_Sj**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half 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:

$$actual\_time = time + Rx\_Sj * sin(Pi * rand())$$

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."))
```

Parameter: **Rx_DCD**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage:	Info, Out
Type:	Float, UI
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: Half 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:

$$actual\_time = time + Rx\_DCD * (-1.0)^n$$

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 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:

$$actual\_time = time + 2.0 * Rx\_Dj * rand() + Rx\_Sj * sin(Pi * rand()) + Rx\_DCD * (-1.0)^n$$

The following optional Reserved Parameter is used to modify the statistics associated with the data input to the receiver's sampling latch (a.k.a. 'slicer'). This data is used by the simulator when post-processing the results from the model; the budget values specified by this parameter are not passed directly to the model itself.

Parameter: **Rx_Noise**

Required: No, and illegal before AMI_Version 6.0

Descriptors:

Usage:	Info, Out
Type:	Float
Format:	Value, List, Range, Corner, Increment, Steps
Default:	<numeric_literal>
Description:	<string>

Definition: The standard deviation, in Volts, of a 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: Time is calculated as follows:

$$wave(t) = wave(t) + Rx\_Noise * gaussian\_rand()$$

Where wave(t) is the waveform returned by Rx AMI_GetWave.

Example:

```
(Rx_Noise (Usage Info) (Value 0.010) (Type Float)
        (Description "Rx amplitude noise at sampling latch in Volts."))
```

Note:
The "Rx_Clock_Recovery Parameters" (Rx_Clock_PDF, Rx_Clock_Recovery_Mean, Rx_Clock_Recovery_Rj, Rx_Clock_Recovery_Dj, Rx_Clock_Recovery_Sj and Rx_Clock_Recovery_DCD, should be used by the simulator when analyzing the output of Rx AMI_Init (for statistical analysis) or Rx AMI_GetWave (time domain) when Rx AMI_GetWave does not return clock_times. When Rx AMI_GetWave returns clock_times, the simulator should not use the "Rx_Clock_Recovery Parameters".

Note:
The "Rx Jitter Parameters" (Rx_Rj, Rx_Dj, Rx_Sj and Rx_DCD, should be used by the simulator when analyzing the output of either Rx AMI_Init (for statistical analysis) or Rx AMI_GetWave (for time domain analysis).

Tables summarizing the rules for the jitter, noise, and sensitivity reserved parameters are shown below.

Table 24 – Allowable Data Types for Jitter and Noise Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage			
	Required	Default	Info	In	Out	InOut
Rx_Clock_PDF	No	Clock Centered	X		X	
Rx_Clock_Recovery_DCD	No	0	X		X	
Rx_Clock_Recovery_Dj	No	0	X		X	
Rx_Clock_Recovery_Mean	No	0	X		X	
Rx_Clock_Recovery_Rj	No	0	X		X	
Rx_Clock_Recovery_Sj	No	0	X		X	
Rx_DCD	No	0	X		X	
Rx_Dj	No	0	X		X	
Rx_Noise	No	0	X		X	
Rx_Receiver_Sensitivity	No	0	X		X	
Rx_Rj	No	0	X		X	
Rx_Sj	No	0	X		X	
Tx_DCD	No	0	X		X	
Tx_Dj	No	0	X		X	

Reserved Parameter	General Rules		Allowable Usage			
Tx_Jitter	No	No Jitter	X		X	
Tx_Rj	No	0	X		X	
Tx_Sj	No	0	X		X	
Tx_Sj_Frequency	No	Undefined	X		X	

Table 25 – 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	X				
Rx_Receiver_Sensitivity	X				
Rx_Rj	X	X			
Rx_Sj	X	X			
Tx_DCD	X	X			
Tx_Dj		X			
Tx_Jitter		X			
Tx_Rj		X			
Tx_Sj		X			
Tx_Sj_Frequency	X				

Table 26 – 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				

Reserved Parameter	Data Format									
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	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.

10.6 REPEATERS

A Repeater is a type of device that is placed in the middle of the channel to compensate 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 30.

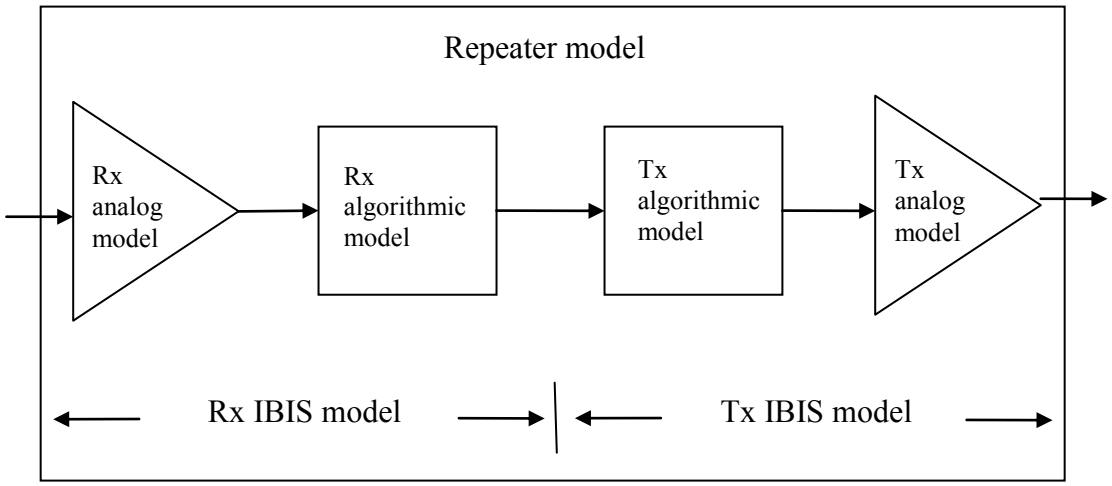


Figure 39 – Repeater model

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.

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]	5 characters max
tx_non_inv_pin	5 characters max

Example:

```
[Repeater Pin]  tx_non_inv_pin
3               11
```

AMI Reserved Parameters:

Parameter: **Repeater_Type**

Required: No, and illegal before AMI_Version 6.0

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"))
```

Tables summarizing the reserved parameters for Repeaters are shown below.

Table 27 – General Rules and Allowable Usage for Repeater Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage			
	Required	Default	Info	In	Out	InOut
Repeater_Type	No (Required with [Repeater Pin])	None	X			

Table 28 – Allowable Data Types for Repeater Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
Repeater_Type				X	

Table 29 – 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 simulation platform shall generate a digital input to the Retimer Tx by sampling the Rx AMI_GetWave output waveform ½ UI after each clock tick, The digital stimulus shall have values of -½ and +½.

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.

The time domain simulation flow for a Repeater link shown in Figure 40 is defined below.

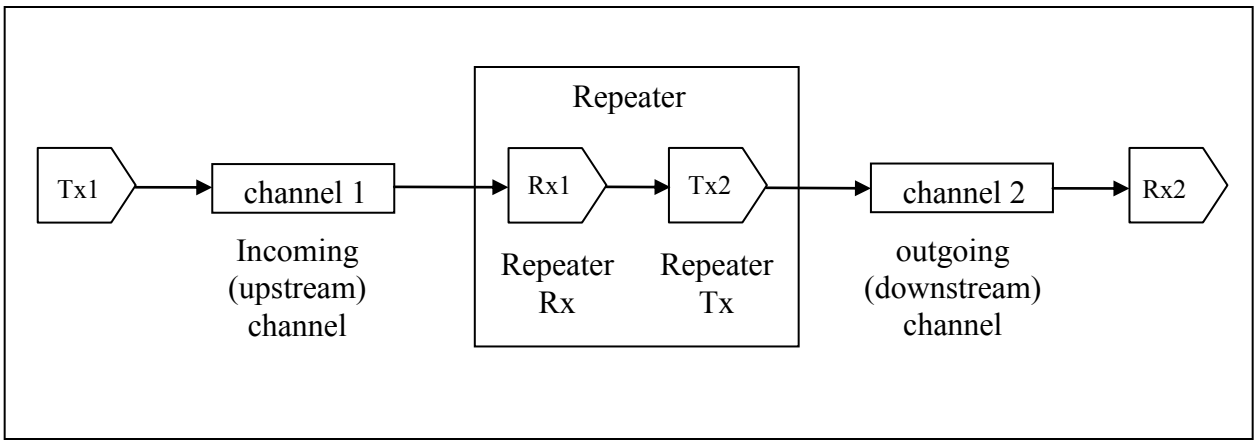


Figure 40 - Repeater link

Here Tx1 denotes the Repeater upstream channel (channel 1) Tx AMI model (including analog and algorithmic models), Rx1 the Repeater Rx AMI model (including analog and algorithmic models), Tx2 the Repeater Tx AMI model (including analog and algorithmic models) and Rx2 the Repeater downstream channel (channel 2) Rx AMI model (including analog and algorithmic models).

Step 1. The simulation platform 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 simulation platform 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 simulation platform performs 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 8a. Redriver: The simulation platform uses the signal waveform at the output end of Rx1's algorithmic model in step 7, regardless whether Rx1's AMI_GetWave exists or not, as the stimulus of Tx2's algorithmic model, regardless whether Tx2's AMI_GetWave exists or not, and performs simulation on the downstream channel, which consists of Tx2, physical channel 2 and Rx2, according to the AMI flow defined in the spec for channels without Redrivers.

Step 8b. Retimer: The simulation platform samples the output waveform of Retimer Rx AMI_GetWave at ½ UI after each clock tick returned by the function, generates a digital stimulus

as the input to Tx2's algorithmic model, regardless whether Tx2's AMI_GetWave exists or not, and performs simulation on the downstream channel, which consists of Tx2, physical channel 2 and Rx2, according to the AMI flow defined in the spec for channels without Redriver. 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_Reciver_Sensitivity, the logic level is unchanged from the previous bit. The digital stimulus have values of -½ volt for logic 0 and +½ volt for logic 1.

Step 9. The simulation platform 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 a Redriver model, jitter parameters and the Rx_Noise parameter specified in Redriver .ami files are ignored by the simulation platform. Since the Retimer output signal is driven by a digital stimulus as described above in step 8b, jitter and noise parameters specified in Retimer .ami files are applied according to the specification for channels without Repeaters.

The statistical simulation flow for a Repeater link shown in Fig. 2 is defined below.

Step 1. The simulation platform 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 simulation platform 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 7a. Redriver: The simulation platform convolves impulse responses returned by Rx1's AMI_Init in step 3 and by Rx2's AMI_Init in step 6 to obtained the full channel impulse response and uses it to perform statistical simulation.

Step 7b. Retimer: The simulation platform uses the impulse responses returned by Rx1's AMI_Init in step 3 to perform a statistical simulation of channel 1. The simulation platform uses the impulse responses returned by Rx2's AMI_Init in step 6 to perform a statistical simulation of channel 2.

IBIS does not prohibit the use of multiple Repeaters, or a mixture of Redrivers and Retimers, cascaded in a channel.

Example:

```
[IBIS Ver]   6.0
```

```
[File Name]  Redriver.ibs
[Component]  Redriver
…
[Pin]    signal_name              model_name        R_pin L_pin  C_pin
1p       Redriver_Rx_1p           Redriver_Rx
1n       Redriver_Rx_1n           Redriver_Rx
2p       Redriver_Tx_2p           Redriver_Tx
2n       Redriver_Tx_2n           Redriver_Tx

[Diff_Pin] inv_pin vdiff tdelay_typ tdelay_min tdelay_max
1p       1n           NA    NA    NA    NA
2p       2n           NA    NA    NA    NA

[Repeater Pin]
1p 2p

[Model] Redriver_Rx
Model_type Input
…
[Algorithmic Model]
Executable Windows_VisualStudio10.0.30319_32 Redriver_Rx_32.dll Redriver_Rx.ami
Executable Windows_VisualStudio10.0.30319_64 Redriver_Rx_64.dll Redriver_Rx.ami
Executable Linux_gcc4.6.1_32 Redriver_Rx_32.so Redriver_Rx.ami
Executable Linux_gcc4.6.1_64 Redriver_Rx_64.so Redriver_Rx.ami
[End Algorithmic Model]

[Model] Redriver_Tx
Model_type Output
…
[Algorithmic Model]
Executable Windows_VisualStudio10.0.30319_32 Redriver_Tx_32.dll Redriver_Tx.ami
Executable Windows_VisualStudio10.0.30319_64 Redriver_Tx_64.dll Redriver_Tx.ami
Executable Linux_gcc4.6.1_32 Redriver_output_32.so Redriver_Tx.ami
Executable Linux_gcc4.6.1_64 Redriver_output_64.so Redriver_Tx.ami
[End Algorithmic Model]

[End]
```

10.7 RESERVED PARAMETER AND DATA TYPE RULE SUMMARY TABLES

The tables below summarize the valid combinations of AMI Reserved Parameters, defaults, data Types and data Formats.

Table 30 – General Rules and Allowable Usage for Reserved Parameters

Reserved Parameter	General Rules		Allowable Usage			
	Required	Default	Info	In	Out	InOut
AMI_Version[1]	Yes	--	X			
DLL_ID[3]	No	No DLL_ID		X		
DLL_Path[3]	No	No DLL_Path		X		
GetWave_Exists	Yes	--	X			
Ignore_Bits	No	0	X			
Init_Returns_Impulse	Yes	--	X			
Max_Init_Aggressors	No	0	X			
Repeater_Type[3]	No (Required with [Repeater Pin])	None	X			
Rx_Clock_PDF	No	Clock Centered	X		X	
Rx_Clock_Recovery_DCD[3]	No	0	X		X	
Rx_Clock_Recovery_Dj[3]	No	0	X		X	
Rx_Clock_Recovery_Mean[3]	No	0	X		X	
Rx_Clock_Recovery_Rj[3]	No	0	X		X	
Rx_Clock_Recovery_Sj[3]	No	0	X		X	
Rx_DCD[3]	No	0	X		X	
Rx_Dj[3]	No	0	X		X	
Rx_Noise[3]	No	0	X		X	
Rx_Receiver_Sensitivity	No	0	X		X	
Rx_Rj[3]	No	0	X		X	
Rx_Sj[3]	No	0	X		X	
Supporting_Files[3]	No	None	X			
Tx_DCD	No	0	X		X	

Reserved Parameter	General Rules		Allowable Usage			
Tx_Dj[3]	No	0	X		X	
Tx_Jitter	No	No Jitter	X		X	
Tx_Rj[3]	No	0	X		X	
Tx_Sj[3]	No	0	X		X	
Tx_Sj_Frequency[3]	No	Undefined	X		X	
Use_Init_Output[2]	No	True	X			

1) Required for AMI_Version 5.1 and later, and illegal before AMI_Version 5.1
2) Illegal for AMI_Version 5.1 and later
3) Illegal before AMI_Version 6.0

Table 31 – Allowable Data Types for Reserved Parameters

Reserved Parameter	Data Type				
	Float	UI	Integer	String	Boolean
AMI_Version[1]				X	
DLL_ID[3]				X	
DLL_Path[3]				X	
GetWave_Exists					X
Ignore_Bits[2]			X		
Init_Returns_Impulse					X
Max_Init_Aggressors			X		
Repeater_Type[3]				X	
Rx_Clock_PDF	X	X			
Rx_Clock_Recovery_DCD[3]	X	X			
Rx_Clock_Recovery_Dj[3]	X	X			
Rx_Clock_Recovery_Mean[3]	X	X			
Rx_Clock_Recovery_Rj[3]	X	X			
Rx_Clock_Recovery_Sj[3]	X	X			
Rx_DCD[3]	X	X			
Rx_Dj[3]	X	X			
Rx_Noise[3]	X				
Rx_Receiver_Sensitivity	X				
Rx_Rj[3]	X	X			
Rx_Sj[3]	X	X			

Reserved Parameter	Data Type				
Supporting_Files[3]				X	
Tx_DCD	X	X			
Tx_Dj[3]		X			
Tx_Jitter		X			
Tx_Rj[3]		X			
Tx_Sj[3]		X			
Tx_Sj_Frequency[3]	X				
Use_Init_Output[2]					X

1) Required for AMI_Version 5.1 and later, and illegal before AMI_Version 5.1
2) Illegal for AMI_Version 5.1 and later
3) Illegal before AMI_Version 6.0

Table 32 – Allowable Data Formats for Reserved Parameters

Reserved Parameter	Data Format									
	Value	Range	Corner	List	Increment	Steps	Gaussian	Dual-Dirac	DjRj	Table
AMI_Version[1]	X									
DLL_ID[3]	X									
DLL_Path[3]	X									
GetWave_Exists	X									
Ignore_Bits	X									
Init_Returns_Impulse	X									
Max_Init_Aggressors	X									
Repeater_Type[3]	X									
Rx_Clock_PDF							X	X	X	X
Rx_Clock_Recovery_DCD[3]	X	X	X	X	X	X				
Rx_Clock_Recovery_Dj[3]	X	X	X	X	X	X				
Rx_Clock_Recovery_Mean[3]	X	X	X	X	X	X				
Rx_Clock_Recovery_Rj[3]	X	X	X	X	X	X				
Rx_Clock_Recovery_Sj[3]	X	X	X	X	X	X				
Rx_DCD[3]	X	X	X	X	X	X				
Rx_Dj[3]	X	X	X	X	X	X				
Rx_Noise[3]	X	X	X	X	X	X				
Rx_Receiver_Sensitivity	X	X	X	X	X	X				

Reserved Parameter	Data Format									
Rx_Rj[3]	X	X	X	X	X	X				
Rx_Sj[3]	X	X	X	X	X	X				
Supporting_Files[3]										X
Tx_DCD	X	X	X	X	X	X				
Tx_Dj[3]	X	X	X	X	X	X				
Tx_Jitter							X	X	X	X
Tx_Rj[3]	X	X	X	X	X	X				
Tx_Sj[3]	X	X	X	X	X	X				
Tx_Sj_Frequency[3]	X	X	X	X	X	X				
Use_Init_Output[2]	X									

1) Required for AMI_Version 5.1 and later, and illegal before AMI_Version 5.1
2) Illegal for AMI_Version 5.1 and later
3) Illegal before AMI_Version 6.0

Table 33 summarizes the relationships between the different Format and Data Types for Reserved or Model Specific Parameters.

Table 33 – 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

11 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 databooks. 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.

The subparameter takes one argument: the heatsink capacitance

The syntax for Heatsink_cap 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	5 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]
```