cadence®

Allegro SI Device Modeling Language User Guide

Product Version 23.1
September 2023

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

Introduction

Objective

This document provides a detailed description of the Device Modeling Language (DML). It covers the syntax and structure of DML files and their use within Allegro SI. It also provides examples of DML models, describes how to write DML macromodels, explains the usage of the `dmlcheck` utility, as well as provides examples of model translations. This document is available to all engineers as a single reference to learn the correct syntax and usage of DML.

Scope

This User Guide covers the Device Modeling Language across the product line, including `tlsim`, `DML parser`, `dmlcheck`, `ibis2signoise` and `quad2signoise`.

Other Reference Materials

The following items are additional references for DML:

- *Allegro PCB SI User Guide*
- "Hspice Simulator User Guide"
- IBIS Version 3.2 ANSI/EIA-656-A Homepage:

  http://www.eigroup.org/ibis/default.htm
- DML Library Files for Allegro SI:

  `<cds_install_dir>/share/pcb/signal/`

1

DML Syntax

What is DML?

The Device Modeling Language (DML) is a Cadence proprietary modeling language. All device models for Allegro SI are stored in DML format. You can use the DML format for many model types, such as IBIS and Spice. DML also includes a behavioral modeling syntax and extensions to the Spice syntax. DML files are ASCII files and may contain one or more Spice-like sub-circuits. In addition, IBIS models are translated into the DML format for simulation: an IBIS file. Although an IBIS file may contain only a single buffer model, it is not uncommon to see an IBIS model refer to a single IBIS file for a device (component), which would include a separate package model, the package parasitics model, and all of the buffer models for the device. A DML model refers to a single specific entity. That entity can be a PackageModel, a Cable model, an ESpice model, or a translated IBIS model. It should be noted that an IBIS model can contain a package model within one translated file.

A DML file is a file that contains one or more models written in the DML language. These model files are used in circuit simulation using analysis tools such as Allegro SI and SigXplorer. Models are developed in advance of simulation, and used to characterize manufactured components such as ICs, discrete components, and connectors. These device models are stored in DML files, which have a `.dml` extension. The simulator requires that simulation models be in the DML format for successful simulation.

DML files are traditionally associated with translated IBIS files. However, DML files are more than just translated IBIS files. They can be library files of discrete components, library files of package models to represent package parasitics, or they can be several DML files grouped together to form an index of library file (`<filename>.ndx`) content.

A DML file consists of keywords that have parameters and sub-parameters enclosed within pairs of parentheses. You can include comments in a DML file for documentation purposes.

Since DML files are ASCII text files, you can use a text editor to edit or create a DML file. Most of the time, you will use the Signal Analysis Library Browser and the Model Browser available from Allegro SI and SigXplorer to help automate the model creation and editing functions. You can use the Model Integrity tool for creating and editing DML files, and translating other model formats into DML.

The DML language is often confused with the Interconnect Description Language (IDL), ESpice and Spice. Table gives the terminology descriptions of Cadence languages.

Description of Cadence Language Terminologies

Language	Description
DML	Device Modeling Language is the file storage for any simulation model used in the simulator. It is an input to the circuit builder.
IDL	Interconnect Description Language is used for modeling traces/vias (`.iml` file extension).
ESPICE	Cadence's flavor of Spice and the `tlsim` net listing language. It is the output of the circuit builder and the input to Cadence's simulator.
KSPICE	Retired name of Cadence's flavor of Spice. It was replaced by ESPICE.

Syntax Rules and Guidelines

Listed below are the general rules and guidelines for DML files:

1. All contents of the files are case sensitive including component names and node names, EXCEPT scaling factors as listed in item 4 below.
2. To facilitate portability between operating systems, the file name and extension must use characters from the set (a space, which is hexadecimal 20 or 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 following characters are not recommended to be used as part of a DML filename:

   `^ $ ~ ! # % & - { }) (@ ' ``

   In general, the file name and extension are recommended to be lower case only.
3. Only ASCII characters, as defined in ANSI Standard X3.4-1986, may be used in a DML file. The use of characters with codes greater than hexadecimal 0x7E 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.
4. Valid scaling factors are:

T or t = tera	K or k = kilo	N or n = nano
G or g = giga	M or m = milli	P or p = pico
MEG or meg = mega	U or u = micro	F or 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. Note that spaces between the number and the scaling factor are NOT permitted. "`1.23 m`" will be read as 1.23e+00, while "`1.23m`" will be read as 1.23e-03. In addition, scientific notation IS allowed (i.e. `1.2345e-12`). In general, units should be omitted, since a unit can be confused with a scaling factor (for example, the values `2f`, `2Farads`, `2fF` will all be read as 2e-15). Characters after a scaling factor are ignored; `3pF`, `3pOhms` and `3pH` are equal. All three are read as 3e-12.

5. Anything following the comment character (`;` or `*` depending where it is used, see Comments) is ignored and considered a comment on that line.
6. Keywords, parameters, sub-parameters (See Keywords, Parameters and Sub-parameters) must be enclosed in parenthesis, `()`. A space or a new line is allowed immediately after the opening parenthesis '`(`' or immediately before the closing parenthesis '`)`'.
7. In an IBIS-based, translated DML model, 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. Similarly, the V-t data tables should use enough data points around sharply curved areas of the V-t curves. These data tables can be graphically viewed in Model Integrity; turn on data point viewing in SigWave to see the data points on the curves.
8. Currents are considered positive when their direction is into the component.
9. All temperatures are represented in degrees Celsius.

File Structure

The top 4 levels of a DML file are organized according to the file structure shown below. The parentheses of these different levels are shown in different colors; Black – Level 1, Red – Level 2, Blue – Level 3, Green – Level 4 and Gold – Level 5.

Note: This is best viewed using Adobe® Acrobat® Reader® as color codes are used here to distinguish the beginning and ending of the different keywords in the DML file structure.

```
("filename.dml"                ; Level 1.The name of this DML file.
  (<ModelTypeCategory>         ; Level 2. A Top-Level DML keyword
                               ; (See ModelTypeCategory Keywords on page 18).
                               ; The parentheses and contents of
                               ; <ModelTypeCategory> should be indented to
                               ; the right of the first parenthesis of the
                               ; DML filename (or "filename.dml" in this
                               ; case) for readability.
                               ; All the model descriptions under a
                               ; specific <ModelTypeCategory> must be
                               ; specified within the
                               ; <ModelTypeCategory>'s parentheses.
    ("modelname"               ; Level 3. "modelname" is a token
                               ; See Token on page 21) and in this case, the
                               ; name that the user and tools will know
                               ; the model by.
                               ; The parentheses and contents of
                               ; "modelname" should be indented to the
                               ; right of the first parenthesis of the
                               ; <ModelTypeCategory> for readability.
                               ; All the model sub-parameter descriptions
                               ; of a specific "modelname" must be
                               ; specified within its parentheses.
     (<model parameters>       ; Level 4. The "modelname" 's parameters
                               ; are specified here within the parentheses
                               ; of "modelname". These parameters are
                               ; indented to the right of the first
                               ; parenthesis of "modelname" for
                               ; readability. Note that the parameters are
                               ; different for each model type.
       (<model sub-parameters> ; Level 5. The "modelname" 's sub-
                               ; parameters are specified here within the
                               ; parentheses of <model sub-parameters>.
                               ; These parameters are indented to the
                               ; right of the first parenthesis of
                               ; <model sub-parameters> for readability.
                               ; Note that the sub-parameters are
                               ; different for each model type.
        )                      ; End of <model sub-parameters description.
     )                         ; End of <model parameters> description.
```

```
    )                           ; End of the "modelname" model description.
    (<other models>             ; Level 3. More models can be specified
                                ; here within the parentheses of this
                                ; <ModelTypeCategory>. These other models'
                                ; parentheses and contents should align
                                ; with the "modelname" model for
    )                           ; readability.
  )                             ; End of <ModelTypeCategory>.
  (<other ModelTypeCategory>    ; Level 2. More <ModelTypeCategory>
                                ; keywords may be specified here within the
                                ; parentheses of the DML filename or
                                ; "filename.dml" in this case.
                                ; These other <ModelTypeCategory> keywords'
                                ; parentheses and contents should align to
                                ; the first <ModelTypeCategory> for
  )                             ; readability.
)                               ; End of the DML filename's ("filename.dml"
                                ; in this example) description.
```

Following is an example of a DML file in order to better illustrate the DML file structure. There are more complete examples of DML files in the appendices.

Example of DML file structure:

```
("DMLFileStructureExample1.dml"     ; This is the DML filename
  (PackagedDevice                   ; Top-level DML keyword declaration
    ("resistor_47Ohm"               ; ModelName  = resistor_47Ohm model
      (PinConnections               ; Single/Common pins are declared here
        (1 2 )                      ; Shows connectivity of this
        (2 1 )                      ; resistor - bi-directional
      )                             ; End of PinConnections definition
      (ESpice "                     * ESpice device model description
        .subckt resistor47Ohm 1 2   * resistor_47Ohm sub-circuit
        R1 1 2 47                   * definition
        .ends resistor47Ohm         * End of resistor_47Ohm sub-circuit definition
      ")                            ; End of ESpice device model description
      (Manufacturer "Signetics")    ; Manufacturer of this resistor
    )                               ; End of  resistor_47Ohm model description
    ("capacitor_20pF"               ; ModelName  = capacitor_20pF model
                                    ; description
      (PinConnections               ; Single/Common pins are declared here
        (1 2 )                      ; Shows connectivity of this
        (2 1 )                      ; capacitor - bi-directional
      )                             ; End of PinConnections definition
      (ESpice "                     * ESpice device model description
        .subckt capacitor20pF 1 2   * capacitor_20pF sub-circuit
        C1 1 2 20p                  * definition
        .ends capacitor20pF         * End of capacitor_20pF sub-circuit definition
      ")                            ; End of ESpice device model description
      (Manufacturer "Signetics")    ; Manufacturer of this capacitor
```

```
    )                              ; End of  capacitor_20pF model description
  )                                ; End of PackagedDevice declaration
)                                  ; End of DMLFileStructureExample1.dml file
```

Comments

A comment consists of all characters in a DML code line that follows a semi-colon. However, the comment character inside the two double-quotes after the SubCircuits keyword is an asterisk (*). Note that empty lines between DML code lines do not need to contain a comment character (; or *). You can also place comments after a left parenthesis or even after a left parenthesis and keyword before the right parenthesis. Semi-colons that are placed within double-quotes for a token (See Token) are considered as semi-colons and therefore part of the token and NOT to mark the beginning of a comment. Following are a few examples that display part of a DML file that has comments as described above.

Example of the correct syntax usage for comments:

```
...
( ; This is a comment line.
  MacroModel
    (Parameters      ; This is also a comment. Parameters keyword is still valid.
      (Buffers
        (BUFF CDSDefaultIO) ) )
      (NumberOfTerminals 8)

; This is a comment line too.
                                                    ;This is NOT a comment line as you need to insert a
semicolon at the beginning of this new line.

    (PinTerminalsMap
      (4 "Data    ; This is NOT a comment line but part of this token,
                  ; since it's inside the quotes.
      ")
      (5 "Enable")
      (8 "Clock" ))
    (SubCircuits "

    * This is a comment line as well.

  .subckt generic_ecl pwr out gd in en pwrcl gndcl
  + ibis_file=ibis_models.inc BUFF=CDSDefaultOutput
  ...
  ")
...
```

ModelTypeCategory Keywords

Table below contains the usage description of the 7 top-level DML keywords that are used to specify the model type.

ModelTypeCategory Keywords

Keyword	Description
PackagedDevice	Defines component models.
PackageModel	Describes package parasitics which may apply to many parts.
BoardModel	Describes an interconnect as a Spice sub-circuit model net by net.
IbisIOCell	Describes I/O buffer model information, also known as a buffer. Does not include any package parasitics.
AnalogOutput	Describes a buffer by an analog waveform.
DesignLink	Describes connectivity between modules.
Cable	Defines interconnects using RLGC matrices; also used by System Configurations to represent cables.

Optional Keywords

Table below contains the usage description of the optional DML keywords, which you can define within any top-level DML keywords (<*ModelTypeCategory*>) as listed in Table . However, the Copyright, Disclaimer, and Notes keywords are recommended and you can place these under the DML filename or at the same level as any top-level DML keywords. The ModelDate, ModelSource, ModelVersion, and Manufacturer keywords are best placed under a specific model name as shown in the example below. You should not define these optional DML keywords below Level 4 as described in File Structure, as it will be tagged as an error in `dmlcheck`. These keywords are not generated by Allegro SI but can only be included in a DML file if it is manually edited.

Optional DML Keywords

Option	Description
Copyright	Any copyright information about the DML file.
Disclaimer	Any disclaimer information about the DML file.
LibraryVersion[1]	The internal version of the DML library used by the software.
Manufacturer	Manufacturer of a specific device in a model.
ModelDate	The date that the model was first created.
ModelSource	The originator and source information of the DML model.

Optional DML Keywords

Option	Description
ModelVersion	The version of the model to keep track of revision history of the DML file.
Notes	Any special notes or comments related to the DML file.

1. Required only for the software to uprev the DML file's libraries to the current version of the software that is being used, if necessary. This keyword is usually generated by Allegro SI and therefore should not be manually edited. LibraryVersion should not be confused with the PSD Library database.

Note: Although these keywords are optional, it is recommended that these keywords are always included when creating new DML files in order to maintain consistency as well as to provide information about the DML file to other users.

Example of usage of the optional DML keywords:

```
("Optional_Keywords_Example.dml"
     (Notes "This model needs to be verified using dmlcheck before usage")
     (Disclaimer "This information is for modeling purposes only, and is
                  not guaranteed")
     (Copyright "Copyright 2002, Cadence Design Systems, All Rights
                  Reserved")
(PackagedDevice
   ("ABT125"
     (ModelVersion "1.1")
     (ModelSource "Created by John Smith using data from lab
                  measurements at Intel")
     (ModelDate "March 8, 2002")
     (Manufacturer "Texas Instruments")
    ...
  (LibraryVersion "136.2") )
```

Token

A token is a string of characters enclosed in double-quotes ("*<token>*") after a left parenthesis. For example, ("20 Pin Connector"). The tokens following the first one, up to the balancing right parenthesis, usually represent the value of the data. Tokens are part of keywords, parameters, and sub-parameters as described in Keywords, Parameters and Sub-parameters. Following are the important rules of a token:

- The first token of a DML file is always the name of the DML file. For example:

```
("DMLfilename.dml"
```

- If a token contains only alphabetic, numeric, and underscore characters, the double-quotes may be omitted. For example, (20_Pin_Connector) and ("20 Pin Connector") are both acceptable tokens. Note that although the usage of double-quotes with tokens is optional in some cases, it is recommended that they are used with every token.
- Within double-quotes, a token may span multiple lines, with the line endings retained as part of the token. There can be multiple values for a token, implementing a data hierarchy. For example:

```
(data
   "1.432p 0.705p
   1.142p 0.505p
   0.852p 0.305p
   0.562p 0.305p
   1.142p 0.705p
   1.432p")
```

- Semi-colons within double-quotes are semi-colons, not the beginning of a comment. (See the example in Comments).
- The first token after each left parenthesis is the name of data. For example, (WireNumber "53") has WireNumber as the name of the data, 53.
- The order of the tokens does not matter. They can be in any order as long as the required tokens are present. For example:

```
(Rise
      (minimum "4.75")
      (typical "5.0")
      (maximum "5.25") )
```

or

```
(Rise
      (maximum "5.25")
      (minimum "4.75")
      (typical "5.0") )
```

are both equivalent definitions of the rise time regardless of the order of their minimum, maximum and typical declarations.

Keywords, Parameters and Sub-parameters

This section describes the meanings of keywords, parameters and sub-parameters, which are used throughout this document to describe DML. The definitions of these words were created for the purpose of documentation.

Keywords are tokens that are used to categorize the different types of DML models. In this document, keywords only refer to the 7 top-level DML keywords (ModelTypeCategory keywords), namely, PackagedDevice, PackageModel, BoardModel, IbisIOCell, AnalogOutput, DesignLink and Cable, as described in Table , EXCEPT the optional keywords, as described in Table . These optional keywords can also be parameters and sub-parameters depending on where they are defined but they are referred to as keywords in this document. A keyword can have a parameter defined directly under it but not just a sub-parameter without a parameter above it. Referring to File Structure, keywords are specified at level 2, which is within the red parentheses.

Parameters are tokens that are defined within the parentheses of keywords. For example, IbisDevice and ESpice are parameters of the keyword, PackagedDevice, and RLGC and CircuitModels are two of the parameters of the keyword, PackageModel. A parameter can have sub-parameters defined under it but not any keywords. Referring to File Structure, parameters are specified at level 4, which is within the green parentheses.

Sub-parameters are tokens are defined within the parentheses of parameters. For example, BufferDelay is a sub-parameter of the parameter, IbisPinMap, and LaunchDelay is the sub-parameter of the parameter, DiffPair. A Sub-parameter can have another sub-parameter defined under it but not any parameters or keywords. For example, the sub-parameter BufferDelay has sub-parameters, Rise and Fall defined under it. Referring to File Structure, sub-parameters are specified at levels 5 (and above), which is within the gold parentheses.

Following is an example to show the difference between keywords, parameters and sub-parameters:

```
(PackagedDevice                    ; Keyword (Level 2)
 (p14u2                            ; Model name (Level 3)
  (DiffPair                        ; Parameter (Level 4)
   (25                             ; Sub-parameter (Level 5)
    (LogicThresholds               ; Sub-parameter (Level 6)
     (Input                        ; Sub-parameter (Level 7)
      (High                        ; Sub-parameter (Level 8)
       (minimum 0)                 ; Sub-parameter and value (Level 9)
       (typical 60mv)              ; Sub-parameter and value (Level 9)
```

```
      (maximum 200mV) )           ; Sub-parameter and value (Level 9)
...
```

Note: Some parameters can be sub-parameters of other parameters depending on where it is defined. For example, RLGC is a parameter when it is defined under the PackageModel keyword but it is a sub-parameter when it is defined under the Connections parameter under the DesignLink keyword.

2

DML Models

Model Overview

The DML Models chapter is organized into seven sections which correspond to each of the seven DML *ModelTypeCategory* keywords. Each section generally contains the following information:

- A general description of the DML *ModelTypeCategory* keyword
- A listing of required and optional parameters and sub-parameters for the DML *ModelTypeCategory* keyword
- Descriptions for each of the parameters and sub-parameters
- A “How to Create” sub-section

PackagedDevice

PackagedDevice is a DML keyword that is used to describe component models. There are two parameters for the PackagedDevice keyword, namely IbisDevice and ESpiceDevice (ESpice). IbisDevice models are used for modeling active components and ESpiceDevice models are used for modeling passive components. IbisDevice allows definitions of DML compatible IBIS keywords for a device and ESpice allows definitions of ESpice sub-circuits for a device. The ESpice parameter is used to indicate that an ESpice sub-circuit follows whereas the IbisDevice parameter is used to indicate that additional IBIS-related sub-parameters follow. However, unlike the ESpice parameter, IbisDevice is not used as a parameter under the PackagedDevice keyword but instead it refers to IbisDevice models. See IbisDevice, to find out more about the IbisDevice parameters.

IbisDevice

An IbisDevice model in its simplest form is just a mapping of buffers to pins. It usually consists of IOCell models that define the electrical operation of each of the pins that make up the IBIS device models. However, an IbisDevice model for a connector has package parasitics but no IOCell models. Refer to IbisIOCell, to find out more about IOCells. An IbisDevice can include a PackageModel to describe the parasitics of the package or the IbisDevice model itself can include pin specific parasitics to account for delay. Refer to Pin/Package Parasitics Definition for an IbisDevice, to learn more about describing pin parasitics for an IbisDevice.

Note: An IbisDevice is not the same as an IBIS device. IbisDevice is the name for Cadence’s devices and IBIS device is simply the name for IBIS devices.

An IbisDevice model can contain both component and buffer properties. Component-level properties are found in the `Pinlist` section of an IBIS file. Buffer-level properties are found

in the `Model` section for the buffer and are discussed in more detail in IbisIOCell. In a DML model, the section for a pin appears to contain both component and some buffer properties.

The following tables show IbisDevice models parameters. The first table shows the required parameters for IbisDevice models as well as the required and optional sub-parameters. The second table shows the optional parameters for IbisDevice models and the respective required and optional sub-parameters. Some of the examples that are used in this sub-section are excerpts from a larger DML file called `cds_samples.dml` which is located in the *<cds_install_dir>/share/pcb/signal/* directory.

Ibis Device Required Parameters and Their Respective Required and Optional Sub-Parameters

Required Parameters	Sub-parameters	
	Required	**Optional**
IbisPinMap	Signal	BufferDelay
	bus[1]	ground_clamp_bus
	signal_model	power_clamp_bus
	ground_bus	ground_pin
	power_bus	power_pin
	WireNumber[2]	DelayMeasurementFixture
	<R, L, C>[3]	MeasurementNodes
PackageModel[4]	*<name of PackageModel>*	-

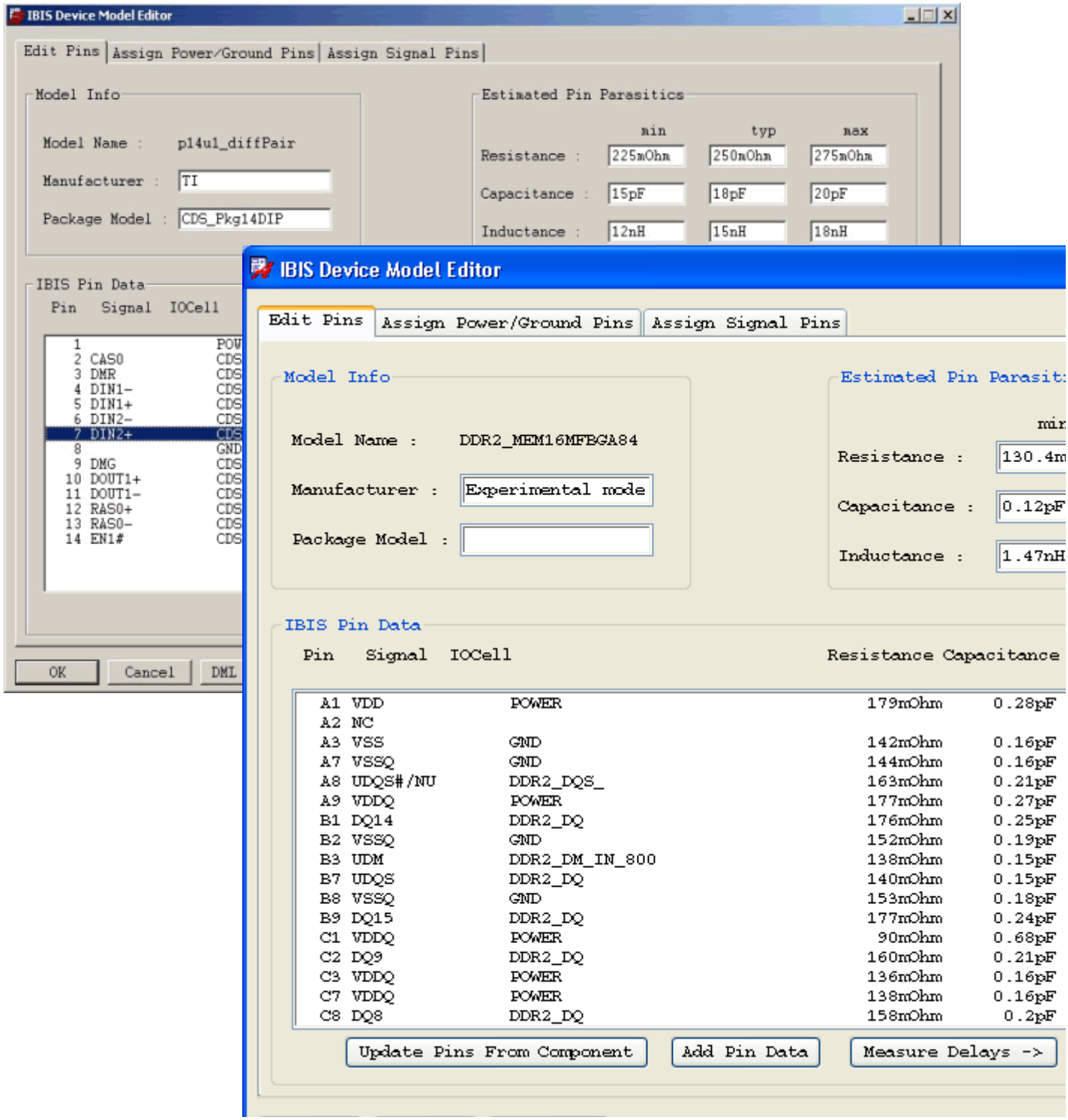
1. The bus sub-parameter is only required for pins that are connected to a bus. Otherwise, the signal sub-parameter can be used instead.
2. The WireNumber sub-parameter is only required when a PackageModel exists for the device and the pin names are not numeric.
3. The R, L and C sub-parameters are only required if this method of defining pin parasitics is desired (See Pin/Package Parasitics Definition for an IbisDevice).
4. The PackageModel parameter is only required if this method is being used to define package parasitics (See Pin/Package Parasitics Definition for an IbisDevice).

IbisDevice Optional Parameters and Their Respective Required and Optional Sub-parameters

Optional Parameters	Sub-parameters	
	Required	**Optional**
`EstimatedPinParasitics`	<R, L, C>	-
`DiffPair`	`InversePin`	`LaunchDelay`
	`LogicThresholds`	
`ModelSelector`	*<name of model>*	-
`MeasurementLocation`	-	`SiLocation`

Figure 2-1 shows the IBIS Device Model Editor window for a model example (`p14u1_diffPair`) from Allegro SI. The user inputs for the *Edit Pins* tab of this form are *Manufacturer*, *Package Model*, and *Estimated Pin Parasitics* (which includes the minimum, typical, and maximum values for *Resistance*, *Capacitance*, and *Inductance*). See Appendix A, "DML File of GUI Example (Figures 2-1 and 2-2) for IbisDevice," for an ASCII version of this model example.

Figure 2-1 The IBIS Device Model Editor

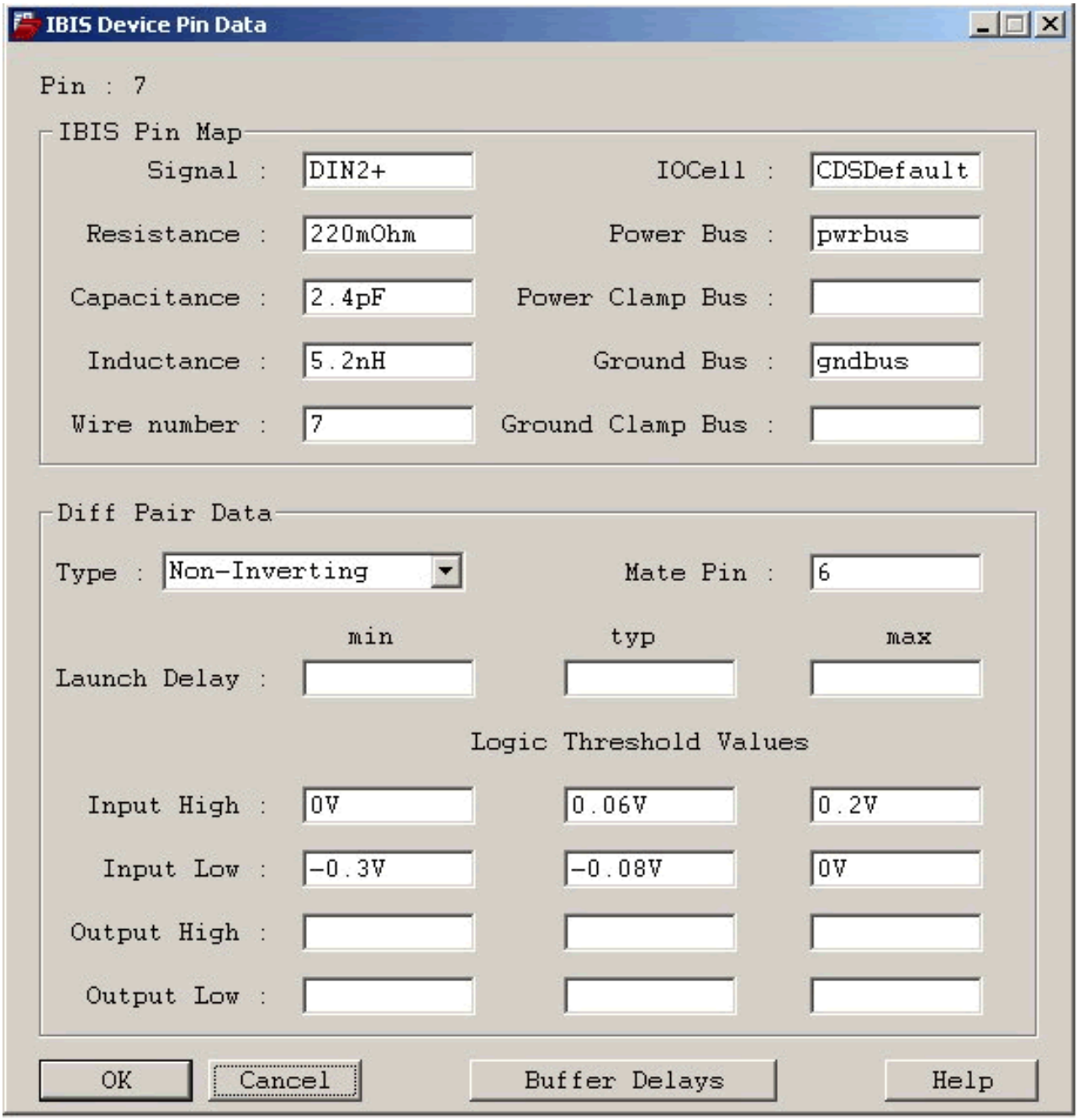


To open IBIS Device Model Editor:

1. In SRECCTRAQuest, choose *Analyze – Model Assignment.*
2. In the Signal Model Assignment form, select the IBIS device model from the list.
3. Click *Edit Model*).

Figure 2-2 shows the IbisDevice Pin Data window for the same model example (`p14u1_diffPair`) shown in Figure 2-1. This figure shows the pin data of pin 7 (highlighted in Figure 2-1) of the `p14u1_diffPair` model. The user inputs for the IBIS Device Pin Data window are allowed under the *Ibis Pin Map* and *Diff Pair Data* sections depending on the model.

Figure 2-2 IBIS Device Pin Data window



Note: You can open the IBIS Device Pin Data window by double clicking on a pin entry from the *Ibis Pin Data* section in the bottom part of the IBIS Device Model Editor in order to edit the pin data for a specific pin.

Pin/Package Parasitics Definition for an IbisDevice

Choosing the method to describe pin/package parasitics can control which keywords, parameters and sub-parameters appear in a DML model. Figure 2-1 on page 29, which shows the IbisDevice Model Editor window for a `p14u1_diffPair` model, is a typical example that shows all three methods of defining package parasitics. The three ways (plus one special case, *None*) of defining pin parasitics for an IbisDevice are described below. Note that these items are listed in decreasing order of precedence in DML from top to bottom.

1. PackageModel – Highest order of precedence for pin parasitics. When `PackageModel` is specified, all other pin parasitics definitions (numbers 2 through 4) are ignored. (See PackageModel for more details). Referring to Figure 2-1 on page 29, the name of the `PackageModel` (if any) is specified at the top left hand corner under the Model Info section. A pin/package parasitics can be defined in two ways:

   - CircuitModels describe pin parasitics using a Spice sub-circuit definition
   - RLGC Matrix represents pin parasitic matrices at different frequencies

   An example of the pin parasitics of `PackageModel` can be found in the PackageModel section.

2. Pin Specific Values – Specified as `R`, `L`, and `C` parameters in `PackagedDevice` by pin number in a parameter form. If a `PackageModel` is defined, then the `R`, `L`, and `C` values of each pin defined under the `PackagedDevice` model is ignored. The *Ibis Pin Data* section of the IbisDevice Model Editor form as shown in Figure 2-1 on page 29 displays the pin specific parasitic values for the `p14u1_diffPair` model. Referring to Figure 2-2 on page 31, the pin parasitics values for the *Resistance*, *Inductance* and *Capacitance* of a specific pin can be specified under the *Ibis Pin Map* section. Here's an example of an `IbisPinMap` definition, which includes pin parasitic parameters.

```
(PackagedDevice
  ("p14u1"                    ; ModelName - p14u1
   (IbisPinMap
     (9                       ; Definition of pin 9 of this device
       (signal "DMG2" )
       (signal_model "CDSDefaultIO" )
       (ground_bus "gndbus" )
       (power_bus "pwrbus" )
       (WireNumber "9" )
     (R "200m" )              ; Parasitic resistance of pin 9
     (L "4.9n" )              ; Parasitic inductance of pin 9
     (C "2.2p" )              ; Parasitic capacitance of pin 9
  ...
```

   Note: The parameters and sub-parameters of IbisPinMap are explained in more detail in IbisDevice Sub-parameters Descriptions.

3. EstimatedPinParasitics – Only used for pins that are not otherwise specified in the `IbisPinMap` or as part of a `PackageModel`. These values are for use only when the specific data is not available. This information is not required. In other words, the `R`, `L`, and `C` values are defaulted to the EstimatedPinParasitics if they are not specified in `IbisPinMap` and `PackageModel`. Referring to Figure 2-1 on page 29, the values of the `EstimatedPinParasitics` (if any), which includes the minimum, typical, and maximum values for the *Resistance*, *Inductance*, and *Capacitance*, are specified at the top right hand corner under the `EstimatedPinParasitics` section. Here's an example of `EstimatedPinParasitics` of `PackagedDevice`.

```
(PackagedDevice
  ("p14u1"                    ; ModelName - p14u1
    (EstimatedPinParasitics
      (R                      ; Parasitic resistance definition
        (minimum "0.225")     ; Minimum parasitic resistance
        (typical "0.25")      ; Typical parasitic resistance
        (maximum "0.275") ) ; Maximum parasitic resistance
        (maximum "2e-11") ); Maximum parasitic capacitance
      (L                ; Parasitic inductance definition
        (minimum "1.2e-8"); Minimum parasitic inductance
        (typical "1.5e-8"); Typical parasitic inductance
        (maximum "1.8e-8") ); Maximum parasitic inductance
      (C                ; Parasitic capacitance definition
        (minimum "1.5e-11"); Minimum  parasitic capacitance
        (typical "1.8e-11") ) ); Typical  parasitic capacitance
      ...
```

4. Special case, *None* – No pin parasitics specified in the DML file, in the Create IBIS Device Model or Ibis Device Pin Data windows. Small default values will be used instead in this case. These default values are as follows: *Resistance* = 1.0e-6, *Capacitance* = 1.0e-15, and *Inductance* = 1.0e-6.

IbisDevice Sub-parameters Descriptions

This section contains descriptions and examples of sub-parameters of IbisDevice.

- IbisPinMap is a required parameter for IbisDevice. It is a representation of the pin map section of an IBIS model. It is where each significant pin of a component is described in terms of its assigned buffer model at a minimum. Pins that are not listed in the `IbisPinMap` are assumed as not connected. Here's an example of `IbisPinMap`:

```
(PackagedDevice
 (IbisPinMap
  (5                                  ; Definition of pin 5
    (signal "RAS0")                   ; See signal below
    (signal_model "CDSDefaultIO")     ; See signal_model below
    (ground_bus "gndbus")             ; See ground_bus below
    (ground_clamp_bus "gndbus")       ; See ground_clamp_bus below
    (power_bus "pwrbus")              ; See power_bus below
    (power_clamp_bus "pwrbus")        ; See power_clamp_bus below
    (R "200m")                        ; See R below
    (L "5n")                          ; See L below
    (C "2p")                          ; See C below
    (WireNumber "5")                  ; See WireNumber below
    (BufferDelay                      ; See BufferDelay below
     ("CDSDefaultIO"                  ; <buffer_delay_signal_model>
      (Rise                           ; See Rise below
        (minimum "6.91381e-10")
        (typical "1.37573e-09")
        (maximum "1.80825e-09") )
      (Fall                           ; See Fall below
        (minimum "4.57694e-10")
        (typical "9.45875e-10")
        (maximum "1.23181e-09") ) ) ) )
    (DelayMeasurementFixture
      (Threshold                      ; See Threshold below
       (minimum "2.5")
       (typical "2.5")
       (maximum "2.5") )
         (C "0p")                     ; See C below
         (R "500")                    ; See R below
         (V "0") )                    ; See V below
    (MeasurementNodes                 ; See MeasurementNodes below
      (v_pin)                         ; <node_name>
      (g_pin) )                       ; <node_name>
 (8
   (bus "gndbus")                     ; See bus below
   (signal_model "ground")
   (R "150m")
   (L "7n")
   (C "3p")
   (WireNumber "8") )
 (1
```

```
   (bus "pwrbus")
   (signal_model "POWER")
   (R "175m")
   (L "8n")
   (C "2p")
   (WireNumber "1") )
 (6                                    ; Start of new pin, 6 definition
   ...
```

Note: The `R`, `L` and `C` values are ignored if `PackageModel` is used to define them. However, the `R`, `L` and `C` values are defaulted to the `EstimatedPinParasitics` if there are not specified in `IbisPinMap` and `PackageModel`.

- signal

  `signal` is a required sub-parameter for `IbisPinMap` *except* for pins that are connected to a bus in which case the sub-parameter bus (see bus) must be used instead as shown in the example above. `signal` is used as a reference and it is typically the name of the signal on a data-sheet and/or schematic diagram.

- signal_model

  `signal_model` is a required sub-parameter for `IbisPinMap`. It is used to refer to an IOCell model that must be specified in the appropriate section of the same or another library file. Some of the default IOCell models that you can use are `CDSDefaultOutput`, `CDSDefaultInput`, `CDSDefaultTristate` and `CDSDefaultIO`. Note that if the sub-parameter `BufferDelay` (see BufferDelay) is defined for a pin, the `signal_model` and the *<buffer_delay_signal_model>* MUST match as shown in the example above.

- ground_bus

  `ground_bus` is a required sub-parameter for `IbisPinMap`. It is used to define the ground bus that the IOCell's ground signal is connected to. Note that ground bus labeling is used during SSN simulation.

- ground_clamp_bus

  `ground_clamp_bus` is an optional sub-parameter for `IbisPinMap`. It is used to specify any different ground bus connections other than the one specified in the `ground_bus` sub-parameter. If it is not present, the ground bus connection is defaulted to the `ground_bus` value.

- power_bus

  `power_bus` is a required sub-parameter for `IbisPinMap`. It is used to define the power bus that the IOCell's power signal is connected to. Note that power bus labeling is used during SSN simulation.

- power_clamp_bus

  `power_clamp_bus` is an optional sub-parameter for `IbisPinMap`. It is used to specify any different power bus connections other than the one specified in the `power_bus` sub-parameter. If it is not present, the power bus connection is defaulted to the `power_bus` value.

- R

  `R` is an optional sub-parameter for `IbisPinMap`. It is used to define the pin specific parasitic resistance value. See Pin/Package Parasitics Definition for an IbisDevice, for more information on defining pin specific parasitics.

- L

  `L` is an optional sub-parameter for `IbisPinMap`. It is used to define the pin specific parasitic inductance value. See Pin/Package Parasitics Definition for an IbisDevice, for more information on defining pin specific parasitics.

- C

  `C` is an optional sub-parameter for IbisPinMap. It is used to define the pin specific parasitic capacitance value. See Pin/Package Parasitics Definition for an IbisDevice, for more information on defining pin specific parasitics.

- WireNumber

  `WireNumber` is a required sub-parameter for `IbisPinMap` only when a `PackageModel` exists for the device and the pin names are not numeric (i.e. A1, B5, OE). `WireNumber` maps a pin to an index in the RLGC matrix under the `PackageModel` keyword (see RLGC for more information).

- BufferDelay

  `BufferDelay` is an optional sub-parameter of `IbisPinMap`. It provides standard delays for each output pin, measured with the IOCell driving the test fixture defined by the `DelayMeasurementFixture` parameter (see DelayMeasurementFixture). This is the time it takes for the IOCell itself to reach the timing threshold (Vmeas) voltage (see Threshold below). When this delay is subtracted from the simulated delays, the result is purely interconnect delay. This value will influence the First Switch Delay and the Final Settle Delay.

  - Rise

    `Rise` is a required parameter for `BufferDelay`. The rising buffer delay time is defined as the time it takes the rising waveform to go from simulation time zero to the simulation time when it crosses the threshold of the buffer delay measurement. You can specify the `minimum`, `typical`, and `maximum` values

for `Rise` but only the `typical` value is required. The `minimum` values are used for Fast mode simulations, `typical` for Typical mode, and `maximum` for Slow mode. This is the default definition that can be changed in the simulation.

- Fall

  `Fall` is a required parameter for `BufferDelay`. The Fall time is defined as the time it takes the falling waveform output to go from simulation time zero to the simulation time when it crosses the threshold of the buffer delay measurement. You can specify the `minimum`, `typical`, and `maximum` values for `Fall` but only the `typical` value is required. The `minimum` values are used for Fast mode simulations, `typical` for Typical mode, and `maximum` for Slow mode. This is the default definition that can be changed in the simulation.

- DelayMeasurementFixture

  `DelayMeasurementFixture` is an optional sub-parameter for `IbisPinMap`. It is used for a standard test load to exercise the output IOCells in order to determine the time it takes for the buffer itself to rise or fall. The rise and fall data are placed under the `BufferDelay` sub-parameter. `DelayMeasurementFixture` requires the `BufferDelay` sub-parameter in an IOCell model in order for it to work properly.

  - Threshold

    `Threshold` is a required sub-parameter for `DelayMeasurementFixture`. `Threshold` is the voltage level that the semiconductor vendor uses for timing measurements in the datasheet for the I/O. The IBIS keyword for `Threshold` is `Vmeas`.

  - C

    `C` is an optional sub-parameter for `DelayMeasurementFixture`. It corresponds to the timing specification test load capacitance value that the semiconductor vendor uses when specifying the propagation delay or the output switching time of the model. The IBIS keyword for `C` is `Cref`.

  - R

    `R` is an optional sub-parameter for `DelayMeasurementFixture`. It corresponds to the timing specification test load resistance value that the semiconductor vendor uses when specifying the propagation delay or the output switching time of the model. The IBIS keyword for `R` is `Rref`.

  - V

    `V` is an optional sub-parameter for `DelayMeasurementFixture`. It is used to define the timing specification test load voltage value. The IBIS keyword for `V` is `Vref`.

- MeasurementNodes

  `MeasurementNodes` is an optional sub-parameter of `IbisPinMap`. It is used to specify the nodes where the Signal Integrity and Timing measurements are made during simulations for a component.

- bus

  `bus` is only a required sub-parameter for `IbisPinMap` when more than one pin in a component is connected to the same signal or power/ground supply nodes. Otherwise, the `signal` sub-parameter should be used instead. The `bus` sub-parameter indicates what bus a pin is tied to. A signal pin should never be tied directly to a bus, however this is not explicitly forbidden. Every bus used must have at least one pin assigned to it with the bus sub-parameter. Otherwise, the bus just floats and is not attached to any pin.

- ground_pin

  `ground_pin` is an optional sub-parameter for IbisPinMap. It refers to the ground pin definition of a device. The `ground_pin` sub-parameter is very similar to the `ground_bus` sub-parameter with the exception that it is used to define the ground pin (as opposed to the ground bus) that the IOCell's ground signal is connected to. If present they are used and result in more detailed modeling of the package characteristics. An example of usage for `ground_pin` is not shown above. In practice, the `ground_bus` sub-parameter is used more often.

- power_pin

  `power_pin` is an optional sub-parameter for `IbisPinMap`. It refers to the power pin definition of a device. The power_pin sub-parameter is very similar to the power_bus sub-parameter with the exception that it is used to define the power pin (as opposed to the power bus) that the IOCell's power signal is connected to. If present they are used and result in more detailed modeling of the package characteristics. An example of usage for `power_pin` is not shown above. In practice, the `power_bus` sub-parameter is used more often.

- PackageModel

  `PackageModel` is only a required parameter for IbisDevice if this method is being used to define package parasitics (See Pin/Package Parasitics Definition for an IbisDevice). Here's an example of `PackageModel`:

```
(PackagedDevice
  ("p14u1"              ; ModelName - p14u1
```

```
   (IbisPinMap
...
)
(PackageModel "CDS_Pkg14DIP") ) )
```

- EstimatedPinParasitics

  `EstimatedPinParasitics` is an optional parameter for IbisDevice. It is used to define pin parasitics for a device. Here is an example:

```
(PackagedDevice
  ("p8u10"                          ; ModelName - p8u10
   (EstimatedPinParasitics
     (R                             ; See R below
       (minimum "0.1")
       (typical "0.2")
       (maximum "0.3") )
     (L                             ; See L below
       (minimum "4.5e-6")
       (typical "5.5e-6")
       (maximum "6.5e-6") )
     (C                             ; See C below
       (minimum "2.2e-9")
       (typical "2.4e-9")
       (maximum "2.6e-9") ) )
...
```

  - R

    `R` is a required sub-parameter for `EstimatedPinParasitics`. It is used to define the pin/package parasitic resistance of a component. You can specify the `minimum`, `typical`, and `maximum` values for `R` but only the `typical` value is required.

  - L

    `L` is a required sub-parameter for `EstimatedPinParasitics`. It is used to define the pin/package parasitic inductance of a component. You can specify the `minimum`, `typical`, and `maximum` values for `L` but only the `typical` value is required.

  - C

    `C` is a required sub-parameter for `EstimatedPinParasitics`. It is used to define the pin/package parasitic capacitance of a component. You can specify the `minimum`, `typical`, and `maximum` values for `C` but only the `typical` value is required.

- DiffPair

  `DiffPair` is an optional parameter for IbisDevice. It is used to define differential pairs according to its electrical characteristics (that is, one positive input and one negative input makes a differential pair of inputs). You can define DiffPair nets by using properties

in the Allegro database (*physical* differential pairs) or by using IbisDevice assignments (*electrical* differential pairs). Both pins and nets are pulled into the differential simulation. The noise immunity for a set of differential pair drivers and receivers is calculated from the driver output voltages and the differential pair logic thresholds. Here's an example of DiffPair:

```
(PackagedDevice
  ("p14u1"                              ; ModelName - p14u1
    (IbisPinMap
      (11
        (signal "DOUT1-")
        (signal_model "CDSDefaultOutput")
        (WireNumber "11")
        (ground_bus "gndbus")
        (power_bus "pwrbus")
        (R "190m")
        (L "4.5n")
        (C "2p")
    (BufferDelay
      (CDSDefaultOutput
        (Rise
          (minimum "6.91344e-10")
          (typical "1.3757e-09")
          (maximum "1.80821e-09") )
        (Fall
          (minimum "4.57657e-10")
        (typical "9.4584e-10")
        (maximum "1.23178e-09") ) ) ) )
      (10
        (signal "DOUT1+")
        (signal_model "CDSDefaultOutput")
        (WireNumber "10")
        (ground_bus "gndbus")
        (power_bus "pwrbus")
        (R "190m")
        (L "4.5n")
        (C "2p")
        (BufferDelay
          (CDSDefaultOutput
            (Rise
              (minimum "6.91279e-10")
              (typical "1.37564e-09")
              (maximum "1.80815e-09") )
            (Fall
              (minimum "4.57593e-10")
              (typical "9.4578e-10")
              (maximum "1.23172e-09") ) ) ) )
      (4
        ...
        )
      (5
```

```
        ...
        )
        ...
        )
      (DiffPair
        (10
          (InversePin "11")                    ; See InversePin below
          (LogicThresholds                     ; See LogicThresholds below
           (Output                             ; See Output below
            (High
              (minimum "4.5")
              (typical "4.8")
              (maximum "5.1") )
            (Low
              (minimum "-5.2")
              (typical "-4.8")
              (maximum "-4.5") ) ) )
           (LaunchDelay                        ; See LaunchDelay below
              (minimum "-1n")
              (typical "0.3n")
              (maximum "1n") ) )
        (5
          (InversePin "4")
          (LogicThresholds
           (Input                              ; See Input below
            (High                              ; See High below
              (minimum "0")
              (typical "60m")
              (maximum "200m") )
            (Low                               ; See Low below
              (minimum "-300m")
              (typical "-80m")
              (maximum "0") ) ) )
           (ThermalShift                       ; See ThermalShift below
            (High
              (typical "0.0015") )
            (Low
              (typical "0.0006") ) ) )
...
) )
```

- InversePin

  `InversePin` is a required sub-parameter of `DiffPair`. It is used to define one pin of the differential pair as the logical inverse of the other. Usually, `InversePin` is used to define the negative pin of the differential pair. The IOCell assigned to the positive pin is taken to be non-inverting even if the IOCell data claims it is inverting. The InversePin is taken to be inverting even if the IOCell data claims it is inverting.

- LogicThresholds

`LogicThresholds` is a required sub-parameter for `DiffPair`. It is used for noise margin and delay calculations.

- Input

  `Input` is a required sub-parameter for `LogicThresholds`. It used to define the high and low threshold voltages for the input of a buffer. Logic threshold voltages can be used for transition and timing constraint checks.

  High

  `High` is a required sub-parameter for `Input`. It specifies the `minimum`, `typical`, and `maximum` logic high input threshold voltage of a buffer. Only the `typical` value is required for `High`. You can specify the `minimum` and `maximum` values but they are optional. The IBIS keyword for `High` is `vinh`.

  Low

  `Low` is a required sub-parameter for `Input`. It specifies the `minimum`, `typical`, and `maximum` logic low input threshold voltage of a buffer. As with `High`, only the `typical` value is required for `Low`. You can specify the `minimum` and `maximum` values but they are optional. The IBIS keyword for `Low` is `vinl`.

- Output

  `Output` is a required sub-parameter for `LogicThresholds`. It is used to define the high and low voltage values of the output of a buffer. The `High` and `Low` sub-parameters for `Output` are similar to the `High` and `Low` sub-parameters that are defined under the `Input` sub-parameter (see High and Low) The only difference is that there are no corresponding IBIS keywords for `High` and `Low` under the `Output` keyword. In other words, the IBIS keywords, `vinh` and `vinl` only correspond to `High` and `Low`, respectively, under the keyword `Input`. As with the `High` and `Low` sub-parameters under the `Input` sub-parameter, the `High` and `Low` sub-parameters under the `Output` sub-parameter only require you to define their respective `typical` values while definitions for their `minimum` and `maximum` values are optional.

- ThermalShift

  `ThermalShift` is an optional sub-parameter for LogicThresholds. It is used to adjust the `LogicThresholds` Input values based on simulation die temperature. Only the `typical` value is required for `ThermalShift`. You can specify the `minimum` and `maximum` values but they are optional. `ThermalShift` does not have a corresponding IBIS keyword.

```
Vhi-adj = Vhi + (Tempco * (comp-junc-temp - VIReferenceTemperature) )
```

The units of this formula are mV and degrees Celsius.

- LaunchDelay

  `LaunchDelay` is an optional sub-parameter for `DiffPair`. It is used to define the time difference between the signals generated by the primary and inverse pins:

  ```
  LaunchDelay = "primary_signal_start" - "inverse_signal_start"
  ```

  A positive launch delay means that the switching of the primary pin is after the switching of the inverse pin, with a delay `LaunchDelay`. Note that `LaunchDelay` is only applicable for differential pairs.

■ ModelSelector

- `ModelSelector`

  `ModelSelector` is an optional parameter for IbisDevice. It is used to define programmable pins that usually have multiple models. The ModelSelector section has a table of symbolic IOCell names that can map to any one of several actual IOCell names. It contains a list of *<symbolic_IOCell_model_names>* corresponding to a specific IOCell (see example below). Within this list, each *<symbolic_IOCell_model_name>* has an (*IOCell_name "description string"*) pair. The *symbolic_IOCell_model_name* is used in the `IbisPinMap` section. The `IOCell_name` is the name of a buffer model that you can select (or assign). The description string is used by tools outside of DML, such as the Allegro SI Signal Model Assignment dialog for interactively assigning selectable IOCells (also known as programmable buffers). The IBIS keyword for the DML keyword `ModelSelector` is `[Model Selector]`. Here's an example of `ModelSelector`:

```
(PackagedDevice
  ("p14u1"                                ; ModelName - p14u1
    (ModelSelector
      (prog_polar_out1                    ; <symbolic_IOCell_model_name>
        (CDSDefaultOutput "CMOS output mode")
        (CDSDefaultOpenDrain "NMOS output mode")
        (CDSDefaultOpenSource "PMOS output mode") )
      (prog_tech_out1                     ; <symbolic_IOCell_model_name>
        (CDSDefault_ECL_Output "ECL output mode")
        (CDSDefaultOpenDrain "MOS output mode") ) )
      (ModelSelectorDefault               ; See part 5.(a)
        (prog_polar_out1 "CDSDefaultOutput")
        (prog_tech_out1 "CDSDefaultOpenDrain") )
    (IbisPinMap
      (13
        (signal "RAS0")
```

```
            (signal_model "prog_polar_out1")    ; IOCell model for pin 13.
                                                ; It is a programmable pin. The
                                                ; name is not a real IOCell, but
                                                ; is a symbolic name found under
                                                ; the ModelSelector parameter.
            (ground_bus "gndbus")
            (ground_clamp_bus "gndbus")
            (power_bus "pwrbus")
            (power_clamp_bus "pwrbus")
            (R "200m")
            (L "5n")
            (C "2p")
            (WireNumber "13")
            (BufferDelay                        ; Pin 13 has multiple buffer
              (CDSDefaultOutput                 ; delay sets because it is
                (Rise                           ; programmable. There is one set
                  (minimum "6.91381e-10" )      ; of buffer delays for each
                  (typical "1.37573e-09" )      ; the pin may use.
                  (maximum "1.80825e-09" ) )
                (Fall
                  (minimum "4.57694e-10" )
                  (typical "9.45875e-10" )
                  (maximum "1.23181e-09" ) ) )
              (CDSDefaultOpenDrain
                (Rise
                  (minimum "9.97033e-10" )
                  (typical "2.51026e-09" )
                  (maximum "3.13216e-09" ) )
                (Fall
                  (minimum "3.9196e-10" )
                  (typical "9.79498e-10" )
                  (maximum "1.21978e-09" ) ) )
              (CDSDefaultOpenSource
                (Rise
                  (minimum "5.91843e-10" )
                  (typical "1.48798e-09" )
                  (maximum "1.85488e-09" ) )
                (Fall
                  (minimum "8.40703e-10" )
                  (typical "2.10886e-09" )
                  (maximum "2.63107e-09" ) ) ) ) )
        ...
        ) ) )
```

- `ModelSelectorDefault`

  `ModelSelectorDefault` is a required sub-parameter for `ModelSelector`. It defines the default programming of `ModelSelector` pins. It only contains (`symbolic_IOCell_name default_IOCell_name`) pairs. It is important to have one of these pairs for every `ModelSelector`, and the default IOCell names given must be members of the corresponding `ModelSelector` lists.

- MeasurementLocation

  `MeasurementLocation` is an optional parameter for IbisDevice. It is used to define where the Signal Integrity and Timing measurements are made for buffers. `MeasurementLocation` has optional sub-parameters SiLocation and TimingLocation. Once `MeasurementLocation` is defined for an IbisDevice, the simulation measurements are done at the die or the package pin, based on the SiLocation and TimingLocation values. Here's an example of `MeasurementLocation`:

```
(PackagedDevice
  ("p14u1"                          ; ModelName - p14u1
    (MeasurementLocation
      (SiLocation Die)              ; See SiLocation below
      (TimingLocation Pin) )        ; See TimingLocation below
  ...
) )
```

  - SiLocation

    `SiLocation` is an optional sub-parameter of `MeasurementLocation`. It is used to define where the signal integrity measurements are made for a component. The allowed values for `SiLocation` are `Pin` or `Die` where the default location is at the Pin.

  - TimingLocation

    `TimingLocation` is an optional sub-parameter of `MeasurementLocation`. It is used to define where the timing measurements are made for a component. The allowed values for `TimingLocation` are `Pin` or `Die` where the default location is at the Pin.

    Examples: 1) If a driver and receiver both have `TimingLocation` defined as `Die`, then the delays are measured from the driver Die to the receiver Die, and includes the driver and receiver package delays. 2) If the driver is on a component with `TimingLocation` set to `Pin`, and the receiver is on a component with TimingLocation set to Die, then the measured delay includes the delay through the driver package but not the receiver package.

How to Create an IbisDevice Model

This section describes the different ways a user can create an IbisDevice model, namely, using the model translation utilities such as `ibis2signoise` and `quad2signoise`, using the GUI of Allegro SI (such as using the Signal Model Assignment form, clone selection, and *Model Browser*). and by manually writing DML files. These different methods of creating and IbisDevice are described here in decreasing order of suggested method and common practices of model creation from first to last.

Model Translation Utilities (Ibis2Signoise & Quad2Signoise)

This is the most common method of creating an IbisDevice model. Models that are written in other modeling languages such as IBIS and QUAD can be translated into DML using model translation tools called Ibis2Signoise and Quad2Signoise respectively. Refer to Chapter 8, "Model Translation," to learn more about how this is done. The translators are also available in Model Integrity, which can be run separately from Allegro SI. After a translation, you can use text editors or the GUI to add information — such as buffer delay into the translated DML file if it is not already contained in the original IBIS/QUAD model.

Signal Model Assignment Form

You can easily create an IbisDevice model using the Signal Model Assignment form since the tool can replicate and edit a new device model from a device that has similar properties. The GUI will seed data based on the makeup of the modeled device (number of pins, pinuse, etc.). You can manually select specific buffers or use system defaults as well. Following are the steps to create an IbisDevice model using the Signal Model Assignment form:

1. Open the Signal Model Assignment Form

   From Allegro SI, choose *Analyze – SI/EMI Sim – Model…*

2. Click on a component on the board or select a device from the list in the Signal Model Assignment form.

3. Click on the *Create Model…* button on the Signal Model Assignment form.

   The Create Model Device dialog opens.

4. Make sure that the radio button next to *Create IbisDevice model* is selected and then click *OK*.

   The Create IbisDevice Model dialog appears containing the model name and pin numbers of the original device that you selected.

5. Enter all the values for the IbisDevice that is being modeled as shown in Figure 2-3 on page 48 and click *OK* at the bottom of this window to create the model.

   You can include additional data by using either a text editor or the model editor GUI.

Clone Selection

You can use the `CloneSelection` option to create IbisDevice models for new devices that have similar properties as other devices, and that already have IbisDevice models in the existing DML Library. The following steps need to be taken in order to create an IbisDevice model using this method:

1. Open the Signal Analysis Library Browser dialog box

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library...*

   The Signal Analysis Library Browser dialog box appears.

2. Click the *Browse Models* button located at the top left hand corner of the Signal Analysis Library Browser dialog box.

   The Model Browser window opens.

3. Set the model filters on the Model Browser window to look for specific models to be cloned. You can do this using the pull down menus next to *Show Models From* (*All Libraries*, *Selected Device Library*, etc.) and *Model Type Filter* (*Any*, *IbisDevice*, etc.) located at the top of the Model Browser window.

4. Select the desired model from the Model Browser window after filtering the models and click on the *Add Model* button and followed by selecting *CloneSelection* from the pull-down menu.

   A new menu appears asking for the model name.

5. Enter the new IbisDevice model name and click *OK*.

   The new device model appears in the Model Browser window.

6. Select this new device model and click *Edit* to modify this new model to fit the intended specification of the new device.

Model Browser Form

You can create the IbisDevice model using the IBIS Device Model Editor form as shown in Figure 2-3 on page 48. However, when the IBIS Device Model Editor window is initially invoked, it will be an empty form that the user needs to enter the values for, based on the

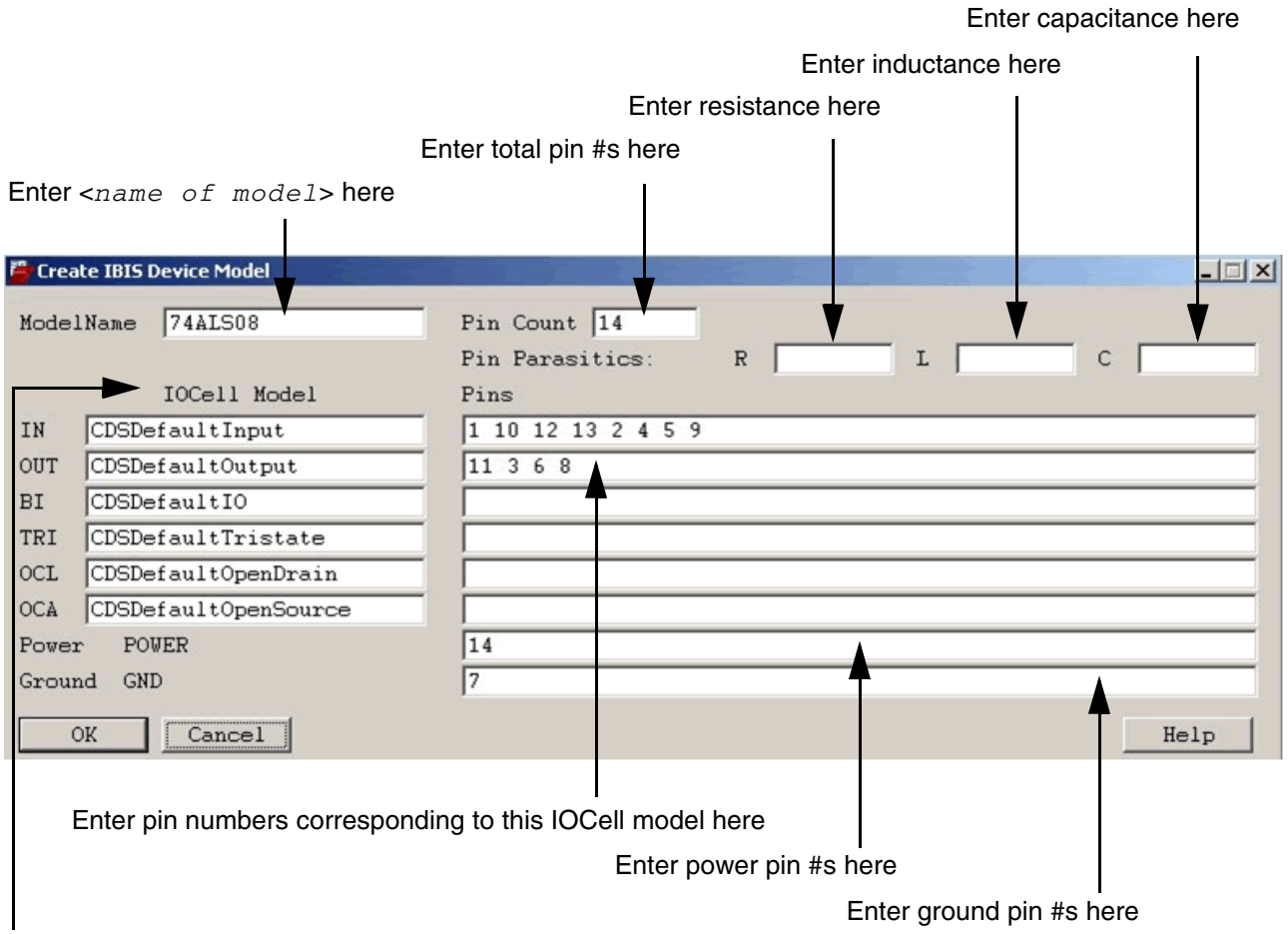
device that is intended to be modeled. The user inputs for this form are *ModelName*, *Pin Count*, *Pin Parasitics*, *IOCell Model*, *Power*, *Ground*, and their respective corresponding pins. See "DML File of GUI Example (Figure 2-3) for IbisDevice" on page 197 for the ASCII version of the model example in Figure 2-3 on page 48. Here's how you can create the IbisDevice model using this method:

1. Open the Create IbisDevice window from the Model Browser, which is accessed from the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library... – Browse Models – Add Model – IbisDevice* from the pull-down menu.

2. Enter all the values for the IbisDevice as shown in Figure 2-3 and click *OK* at the bottom of this window to create the model.

Figure 2-3 IBIS Device Model Editor example for a 74ALS08 device



These IOCell models are initially populated with the system's default models but you can change them when editing the model using this window

Note: The `R`, `L` and `C` values that are entered in the form above will be used as the `minimum`, `typical`, and `maximum` values under the `EstimatedPinParasitics` parameter.

The pin parasitics section of Figure 2-3 is empty because this device has its pin parasitics defined in its PackageModel.

Manual creation of an IbisDevice model

You can create an IbisDevice model by manually writing a DML file using any text editors, although it is rarely done this way. It is important that the file that contains this manually created model is saved with the extension .dml (that is, `<DMLfilename>.dml`). The `<DMLfilename>.dml` has to exactly match the first token of the DML file, which is the name of the DML file as explained in Chapter 1, "Token." Note that the file name is case sensitive. Listed below are the steps to manually create a PackagedDevice model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the template listed in Appendix A, "IbisDevice Template," into this file.
3. Edit the DML file, which has the IbisDevice template as needed to create the new model.
4. Save the file and run `dmlcheck` to check for syntax errors. See Chapter 6, "dmlcheck Utility," for more details.

Note: IbisDevice template:

- This template is a skeleton of the DML file that the user can copy and paste into a text editor and then customize to model a specific device.
- The user might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.
- Refer to the specific parameter and sub-parameter descriptions, covered in "IbisDevice Sub-parameters Descriptions" on page 34, for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`PackagedDevice`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.
- The different ways of defining pin parasitics of a device is described in Pin/Package Parasitics Definition for an IbisDevice.

ESpice

ESpice is Cadence's flavor of Spice and the `tlsim` net listing language. In DML, the ESpice keyword is used to define discrete components such as resistors, capacitors, and inductors. An ESpice model can generally be specified for two different types of devices, namely, 2 pin devices and multi-pin devices, also known as a black box in SigXplorer. The examples that are used in this sub-section are excerpts from a larger DML file called `cds_samples.dml` which is located in the <*`cds_install_dir`*>`/share/pcb/signal/` directory.

ESpice Required Parameter and its Required Sub-parameter

Required Parameter	Sub-parameter	
	Required	**Optional**
`ESpice`	`.subckt`	-

ESpice Optional Parameters and Their Respective Required Sub-parameters

Required Parameter	Sub-parameters	
	Required	**Optional**
`PinConnections`	<*`pin_numbers`*>	-

ESpice Sub-parameters Descriptions

This section contains descriptions and examples of the ESpice sub-parameters.

1. .subckt

   `.subckt` is a required sub-parameter for ESpice. It is used to specify an ESpice sub-circuit definition. The language that is used within the two double quotes after the ESpice parameter is ESpice and not DML. For more information on how to write ESpice models, refer to Appendix J of the " *Allegro PCB SI User Guide*.

   Like Spice, ESpice depends on new-line characters so the formatting of the data is significant and must be retained by software using the data. The device name and the sub-circuit name *must* match. The software assumes that this is true and uses the device name when calling the sub-circuit and simply inserts the sub-circuit into the circuit it is building as given. The node-names in the sub-circuit are used as the pin names in the drawing. Thus, only the pins that are actually connected in the terminal list are needed.

   Following is an example of a 2-pin capacitor that has an ESpice sub-circuit definition.

```
(PackagedDevice
  ("capacitor20pF"
    (PinConnections
      (1 2 )
      (2 1 ) )
    (ESpice
      ".subckt capacitor20pF 1 2
      C1 1 2 20p
      .ends capacitor20pF") ) )
```

2. PinConnections

   `PinConnections` is an optional parameter for an ESpiceDevice. It is used to define pin directionality and pin connectivity between devices. If the `PinConnections` parameter is missing from an ESpice model definition, then all pins are assumed to be not connected. Note that `PinConnections` is *not* a sub-parameter of ESpice but instead a parameter itself. It is usually used with the ESpice parameter for ESpice model definitions under the `PackagedDevice` keyword.

   You can specify `PinConnections` for any PackageModel, but they are primarily for ESpice model definitions under the `PackagedDevice` keyword. For non-ESpice PackageModel devices, the absence of the `PinConnections` parameter does *not* imply any connections. Only the connections specifically called for by other means are used. Any `PinConnections` that is specified is in addition to those implied by the model.

In order to get a bi-directional effect in a 2-pin model (that is, a 2-pin resistor) specification, the `PinConnections` parameter must be used as follows.

```
(PinConnections
  (1 2 )
  (2 1 ) )
```

Here's an example of `PinConnections`:

```
(PackagedDevice
  ("VSource"
    (PinConnections
      ("Rin" "Rout")
      ("Rout" "Rin") )
...
```

Pin Device Model Example

This is an example of a 2-pin ESpice model for a single inductor with value 15 nH.

```
(PackagedDevice
  ("inductor15nH"
    (PinConnections
      (1 2 )
      (2 1 ) )
    (ESpice
      ".subckt inductor15nH 1 2
      R1 1 2 1e-6 L=1.5e-8
      .ends inductor15nH") ) )
```

Multi-Pin Device Model Example

This is an example of a multi-pin ESpice model for a common pin resistor pack with 8 resistors with value of 850 Ohms.

```
(PackagedDevice
  ("resistorPack_850"
    (PinConnections
      (S1 G1)           ; If S1 is in the circuit also include G1. Specify the
      (S2 G1)           ; PinConnections this way so that S1 does not force
      (S3 G1)           ; all the other Sn pins into the circuit.
      (S4 G1)
      (S5 G1)
      (S6 G1)
      (S7 G1) )
    (ESpice
      ".subckt resistorPack_850 G1 S1 S2 S3 S4 S5 S6 S7
      R1 S1 G1 50
      R2 S2 G1 50
      R3 S3 G1 50
      R4 S4 G1 50
```

```
R5 S5 G1 50
R6 S6 G1 50
R7 S7 G1 50
.ends resistorPack_850") ) )
```

HSpice Black Box Model Example

You can use the HSpice language in DML using a black box model. Following is the black box template for a VSource model. HSpice syntax is set by the `LITERAL_HSPICE` keywords. To use this same template in Tlsim, simply remove these two lines. Note that the `.subckt` node names define the names used on the terminals in SigXp.

```
(PackagedDevice
  (VSource
    (PinConnections      ; PinConnections helps the tool comprehend Xnets
                         ; In this example, there's a 50 Ohm resistor
                         ; connected between external nodes Rin and Rout.
                         ; Because of the PinConnections statement, the
                         ; circuits connected to both Rin and Rout will be
                         ; considered one Xnet.
      ("Rin" "Rout" )
      ("Rout" "Rin" ) )
    (ESpice
      ".subckt VSource V1 V1G Rin Rout V3 V3g
      BEGIN_LITERAL_HSPICE
      Vpul V1 V1G PULSE (0 2.5 0n 1n 1n 8n 20n)
      Rterm Rin Rout 50
      etest V3 V3g VOL='v(v1)'
      END_LITERAL_HSPICE
      .ends VSource") ) )
```

How to Create an ESpice Model

This section describes the different ways a user can create an ESpice model, namely, using the model translation utilities such as `ibis2signoise` and `quad2signoise`, using the GUI of Allegro SI (such as using the auto setup function, Signal Model Assignment form, clone selection, and *Model Browser*), and by manually creating the ESpice model. The different methods of creating ESpice models are described here in decreasing order of suggested method and common practices of model creation from first to last.

Auto Setup function

You can use the Auto Setup function to automatically create multiple 2-pin ESpice models simultaneously if the board is setup properly (by using the Database Setup Advisor). This function will attempt to assign and create signal models for all 2-pin discrete components on

a board. Based on the reference designator and `VALUE` property that is set up, it will create and assign a resistor, capacitor, or inductor to each component. After completing the Auto Setup routine, the tool will populate the form with the successful creations and assignments, and creaete a log file. This process will add the `SIGNAL_MODEL` property to each component.

An Allegro user looking to make use of XNets in the Allegro database does not need to be concerned with the actual models that are created. These models are stored in the database and a file on a disk (`devices.dml` or the working Device Library file), and the user can check them at a later time. These new signal models for the discrete devices will allow Allegro to recognize XNets that pass through 2 pin devices.

The Auto Setup function for creating ESpice device models is invoked as follows:

➤ Open the Signal Model Assignment form (from Allegro SI, choose *Analyze – SI/EMI Sim – Model...*) and click on the *Auto Setup* button on this form. The Auto Setup function will run and create ESpice models of all 2-pin discrete devices on a board.

Signal Model Assignment form

You can easily create an ESpice model using the Signal Model Assignment form since the tool can replicate and edit a new device model from a device that has similar properties. The GUI will seed data based on the makeup of the device to be modeled (number of pins, etc.). The user can select specific buffers or use system defaults. Following are the steps to create an ESpiceDevice model using the Signal Model Assignment form:

1. Open the Signal Model Assignment Form

   From Allegro SI, choose *Analyze – SI/EMI Sim –Model...*

2. Click on a component on the board or select a device from the list in the Signal Model Assignment form.

3. Click the *Create Model...* button on the Signal Model Assignment form.

   The Create Model Device window appears. Make sure that the *radio* button next to *Create ESpiceDevice model* is selected.

4. Click *OK*.

   The Create ESpiceDevice Model window appears, containing the model name and pin numbers of the original device that was selected.

5. Enter all the values for the ESpice device as shown in Figure 2-4 on page 56.

6. Click *OK* at the bottom of this window to create the model.

Clone Selection

You can use the *Clone Selection* option to create ESpiceDevice models for new devices that have similar properties as other devices, and that already have IbisDevice models in the existing DML Library. To create an ESpiceDevice model using this method:

1. Open the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library…*

   The Signal Analysis Library Browser window appears.

2. Click the *Browse Models* button located at the top left-hand corner of the Signal Analysis Library Browser window.

   The Model Browser window appears.

3. Set the model filters on the Model Browser window to look for specific models to be cloned. You can do this by using the pull down menus next to *Show Models From* (*All Libraries*, *Selected Device Library*, etc.) and *Model Type Filter* (*Any*, *ESpiceDevice*, etc.) located at the top of the Model Browser window.

4. Select the desired model from the Model Browser window after filtering the models

5. Click the *Add Model* button and select *CloneSelection* from the pull-down menu.

   A new menu appears asking for the model name.

6. Enter the new ESpiceDevice model name and click *OK*.

   The new device model appears in the Model Browser window.

7. Select this new device model and click *Edit* to modify this new model to fit the intended specification of the new device.

Model Translation Utilities (Ibis2Signoise & Quad2Signoise)

You can translate models that are written in other modeling languages (such as IBIS and QUAD) into DML using the model translation tools called Ibis2Signoise and Quad2Signoise respectively. Refer to Chapter 8, “Model Translation,”, to learn more about how this is done.

Model Browser Form

You can also create the ESpice model using the Create ESpice Device Model window as shown in Figure 2-4. However, when the Create ESpice Device Model window is initially invoked, it will be an empty form that the user needs to enter the values for based on the device that is intended to be modeled. See Appendix A, “DML File of GUI Example (Figure 2-

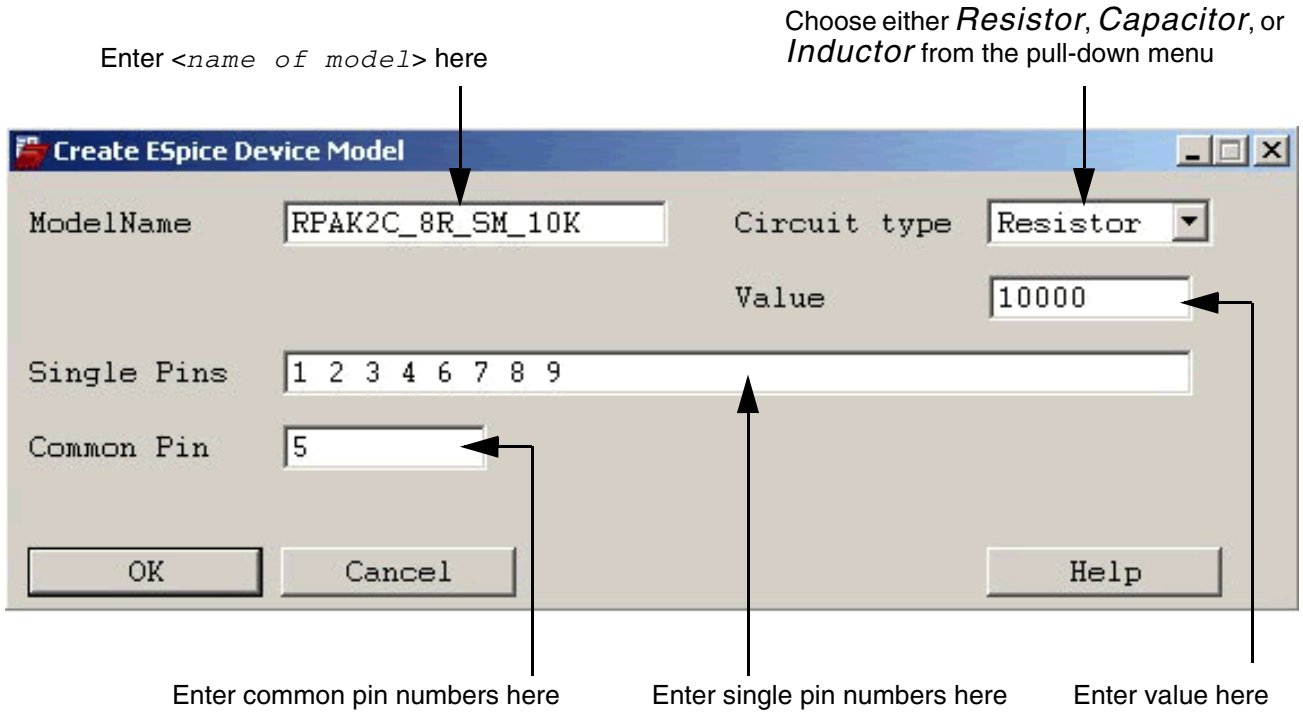
4) for ESpice," for the ASCII version of the model example in Figure 2-4. Here's how the ESpice model can be created using this method:

1. Open the Create ESpice Device window from the Model Browser, which is accessed from the Signal Analysis Library Browser window.

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library... – Browse Models – Add Model – ESpiceDevice*

2. Enter all the values for the ESpice device that is being modeled as shown in Figure 2-4.
3. Click *OK* at the bottom of this window to create the model.

Figure 2-4 Create ESpice Device Model Window Example for a 10k Ohm 8-Pack Resistor



Manual Creation of an ESpice Model

You can create an ESpice model by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` file has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Chapter 1, "Token." Listed below are the steps to create an ESpice model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the template that is listed in Appendix A, "ESpice Template," into this file.
3. Edit the DML file.
4. Save the file and run `dmlcheck` to check for syntax errors. See Chapter 6, "dmlcheck Utility,", for more information.

Note: ESpice model template

- This template is a skeleton of the DML file that the user can copy and paste into a text editor and then customize to model a specific device.
- The user might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.
- Refer to the specific parameter and sub-parameter descriptions described in ESpice Sub-parameters Descriptions for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`PackagedDevice`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

PackageModel

The PackageModel keyword is used to define a model for a component. An IbisDevice model can reference it and use it to model the package parasitics of the entire component package using RLGC matrices or SPICE sub-circuits. There are four main parameters for the PackageModel keyword, namely, `RLGC`, `CircuitModels`, `PinNameToNumber`, and `pin_count`. The following two ables show PackageModel parameters. The first table shows the required parameters for PackageModel and their respective required and optional sub-parameters. The second table shows the optional parameters for PackageModel and their respective required and optional sub-parameters. Some of the examples in this section are excerpts from a larger DML file called `cds_packages.dml` which is located in the `<`*`cds_install_dir`*`>/share/pcb/signal/` directory.

PackageModel Required Parameters and their Respective Required and Optional Sub-parameters

Required Parameters	Sub-parameters	
	Required	**Optional**
CircuitModels	SingleLineCircuits[1]	CoupledLineCircuits
RLGC[2]	Band	
	Dimension	
	Data	
	SubCircuits	

1. SingleLineCircuits and optionally CoupledLineCircuits (one or both) have to be defined under the CircuitModels parameter depending on the specific PackageModel.
2. RLGC data in a PackageModel will be used ONLY IF CircuitModels does not exist.

PackageModel Optional Parameters and their Respective Required and Optional Sub-parameters

Parameter	Sub-parameter	
	Required	**Optional**

Parameter	Sub-parameter
PinNameToNumber	*<pin_name pin_number>* -

CircuitModels

The `CircuitModels` format describes a package model using Spice sub-circuit syntax. The `RLGC` description is completely ignored when the `CircuitModels` keyword is defined. `CircuitModels` is used for large package model descriptions and embeds arbitrary Spice models for packages.

CircuitModels Sub-parameters Descriptions

This section contains examples and descriptions of the sub-parameters of `CircuitModels`.

1. SingleLineCircuits

   `SingleLineCircuits` is a required sub-parameter of `CircuitModels`. It describes a corresponding single line sub-circuit (without coupling to adjacent lines) for each pin in the package. Here's how the general syntax for SingleLineCircuits looks:

```
(SingleLineCircuits
  (<pin_number>
    (SubCircuitName <name_of_subckt>)
    ...
  )
)
```

   Here's an example of usage of SingleLineCircuits:

```
(PackageModel
  (CircuitModels
    (SingleLineCircuits
      ("1"
        (SubCircuitName "simple") )           ; See SubCircuitName below
      ("2-12, 26, 27"
        (SubCircuitName "easy") )
      ("13, 14, 15-25, 28-30"
        (SubCircuitName "cake") ) )
    (SubCircuits                             ; See SubCircuits below.
    ".subckt simple sin sout                 * sin is <input/pad> and
                                             * sout is <output/pin>
    Rpkg sin dud 75m
    Lpkg dud sout 4nH
    Cpkg dud 0   3.0pF
    .ends simple

    .subckt easy  in1 in2 out1 out2          * in1, in2 are <input/pad> and
```

```
                                        * out1 out2 are <output/pin>
...
.ends easy

.subckt cake d1 d2 d3 z1 z2 z3          * d1, d2, d3 are <input/pad>
                                        * z1, z2, z3 are <output/pin>
...
.ends cake")
```

a. SubCircuitName

`SubCircuitName` is a required sub-parameter of `SingleLineCircuits`. It is used to define the name of the sub-circuits of the pins for the `SingleLineCircuits`. The SubCircuitName for the pins MUST match the corresponding `.subckt` name under the `SubCircuits` sub-parameter. In the example above, note that the `SubCircuitNames` `simple`, `easy`, and `cake` match the `.subckt` name `simple`, `easy`, and `cake` respectively.

Large package models can contain `minimum`, `typical`, and `maximum` sub-circuit definitions. This allows for the transparent mapping for fast, typical, and slow simulations. For fast, typical, and slow simulations, if the `ftstype` is not specified, the default mapping will be same as that used for `C_COMP` or `DieCapacitance`. The `DieCapacitance` usually means `minimum` value for fast and `maximum` for slow. The `ftstype`, `RampRates`, usually means `maximum` value for fast and `minimum` for slow. Following is the syntax for `SubCircuitName` to define `minimum`, `typical`, and `maximum` sub-circuit definitions:

```
(SubCircuitName
  (minimum <name1>)
  (typical <name2>)
  (maximum <name3>)
  (ftstype <type>) )
```

Here's an example of usage of `SubCircuitName` using the syntax shown above:

```
(PackageModel
  (CircuitModels
    (SingleLineCircuits
      (1
        (SubCircuitName
          (minimum longsingle_dc_wiremin)
          (typical longsingle_dc_wire)
          (maximum longsingle_dc_wiremax) ) )
      (2
        (SubCircuitName
          (minimum longsingle_dc_wiremin)
          (typical longsingle_dc_wire)
          (maximum longsingle_dc_wiremax) ) )
      (4-14
        (SubCircuitName
          (minimum longsingle_dc_wiremin)
```

```
            (typical longsingle_dc_wire)
            (maximum longsingle_dc_wiremax) ) ) )
        (SubCircuits "
        * See SubCircuits for the SubCircuits definition for
        * this example.
        ...
        ")
      )
    )
```

2. CoupledLineCircuits

   `CoupledLineCircuits` is an optional sub-parameter of `CircuitModels`. It is used only for Crosstalk and Comprehensive simulations. It is used to specify coupling in a PackageModel. There can be multiple sets of coupled line circuits applicable to different speeds of operation.

   When using coupled PackageModels, the specific subcircuit that gets pulled into the simulation circuit from the PackageModel is dependent on the coupling found at the PCB level. The system algorithmically strives to take the largest coupled subcircuit from the PackageModel that contains the coupled pins found in the circuit.

   As an example, three coupled signals—DATA1, DATA2, and DATA3—are brought into a crosstalk circuit. These signals run to pins J1.1, J1.2, and J1.3 respectively on the connector “J1”, which has a PackageModel associated with it. The PackageModel contains multiple subcircuits in the `CoupledLineCircuits` section of the model.

   First, the system searches for a coupled circuit in the PackageModel that contains all of the pins needed from this connector; J1.1-3. If it finds an exact match containing these three pins (and only these pins), it pulls it from the PackageModel and plugs it into the circuit. But if it finds a subcircuit that includes these three pins plus additional pins, it does not use it, and the search for the appropriate subcircuits continues.

   If this exact match is not found, the system next looks for a subcircuit containing any two of the three pins and no others. If it finds a subcircuit containing only J1.1 and J1.2, it takes it, then pulls a SingleLineCircuit (uncoupled) for pin J1.3.

   If no `CoupledLineCircuits` are found that contain two of the three required pins, then SingleLineCircuits are pulled for each of the three pins.

   The organization of information is identical to `SingleLineCircuits` except for the addition of the `Terminals` sub-parameter. Here’s an example of `CoupledLineCircuits`:

```
(PackageModel
  (CoupledLineCircuits
    ("1-5, 8"
      (Terminals "1.in" "1.out" "2.in" "2.out" "3.in" "3.out" "4.in"
                 "4.out" "5.in" "5.out" "8.in" "8.out")
      ; See Terminals for more information about Terminals
```

```
      (SubCircuitName "ideal") ); See SubCircuitName below.
      ("6,7, 9-12"
      (Terminals "6.in" "6.out" "7.in" "7.out" "9.in" "9.out" "10.in"
                 "10.out" "11.in" "11.out" "12.in" "12.out")
      (SubCircuitName "nonideal") ) )
(SubCircuits
".subckt ideal 1i 1o 2i 2o 3i 3o
xpkg 1o 1i 2o 2i 3o 3i vtt vss vtti vssi vccp vssp vccpi pkg_fast
vvcc vccp vssp 5.0
vvtt vtt  0    2.5
vvss vss  0    0.0
Cdud1 vtti  vssi 75pf
Cdud2 vccpi vssi 75pf
.ends ideal

.subckt nonideal d1 d2 d3 d4 z1 z2 z3 z4
...
.ends nonideal") )
```

a. Terminals

`Terminals` is a required sub-parameter for `CoupledLineCircuits`. It is used to identify the terminals of the sub-circuit for `CoupledLineCircuits` only. Each wire has two terminals associated with it. A `SingleLineCircuits` does not need the sub-parameter `Terminals` since there are always only two nodes. The syntax for `Terminals` is as follows:

```
(Terminals "<wire_number>.in" "<wire_number>.out")
```

For IbisDevices,

`.in` = Buffer Side

`.out` = Component side

For Cable models,

`.in` = First Pin in PinMap section

`.out` = Second Pin in PinMap section

b. SubCircuitName

`SubCircuitName` is a required sub-parameter of `CoupledLineCircuits`. It is defined similarly to the SingleLineCircuits sub-parameter of the same name. See SingleLineCircuits on page 59 for more information.

3. `BlockCircuits`

`BlockCircuits` is an optional sub-parameter of `CircuitModels`. It is similar to `CoupledLineCircuits`, but is differentiated from it in the following ways:

- When the circuit builder decides it needs a net that runs through a `BlockCircuit`, the coupled nets are not extracted. (This is exactly opposite the case for a net running through a `CoupledLineCircuit`, where all the coupled wires become part of the analysis and the nets they are connected to are completely extracted at the board level.)
- Coupled, unextracted wires are automatically attached to stub circuits defined by the `UnusedTerminalsTermination` parameter.

These are called block circuits because the terminals of these circuits may include pins, which may not be directly relevant to the physical circuit. (They may also contain large numbers of power and ground pins). `BlockCircuits` are typically very large and usually contain wires which are not in the circuit extractor's requested list. `BlockCircuits` take priority over `CoupledLineCircuits`. The complete search path is:

1. `BlockCircuits` of powerground type
2. `CoupledLineCircuits`
3. `SingleLineCircuits`
4. `BlockCircuits` of signal type

Here is an example of `BlockCircuits`:

```
("example_BlockCircuit_model.dml"
 (PackageModel
  (six_conductor_BC
   (CircuitModels
    (BlockCircuits
     (SignalCircuits
      ("1-6"
       (Terminals "1.in" "1.out" "2.in" "2.out" "3.in" "3.out" "4.in" "4.out" "5.in" "5.out" "6.in"
"6.out" )
       (SubCircuitName six_coupled_traces )
       (UnusedTerminalsTermination
        (SubCircuitName term50 )
        (SubCircuits "
.subckt term50 1
r1 1 0 50
.ends term50
" ) ) ) ) )
    (SubCircuits "
.subckt six_coupled_traces in1 out1 in2 out2 in3 out3 in4 out4 in5 out5 in6 out6
* Length L in meters
* Resistance in ohm/m
* Inductance in henries/m
* Conductance in siemens/m
* Capacitance in farads/m
NTL_1 (in1 in2 in3 in4 in5 in6 0) (out1 out2 out3 out4 out5 out6 0) L=0.0254 rlgc_name=MTL_2S_6R_1
```

```
*
* RLGC definition for MTL_2S_6R_1
DATAPOINTS RLGC MTL_2S_6R_1
FREQUENCY=0
CMATRIX
 1.239500e-010 -2.818500e-011 -1.051000e-012 -5.606700e-014 -3.037200e-015 -1.670600e-016
 -2.818500e-011 1.316800e-010 -2.790600e-011 -1.036200e-012 -5.526400e-014 -3.037200e-015
 -1.051000e-012 -2.790600e-011 1.316900e-010 -2.790500e-011 -1.036200e-012 -5.606700e-014
 -5.606700e-014 -1.036200e-012 -2.790500e-011 1.316900e-010 -2.790600e-011 -1.051000e-012
 -3.037200e-015 -5.526400e-014 -1.036200e-012 -2.790600e-011 1.316800e-010 -2.818500e-011
 -1.670600e-016 -3.037200e-015 -5.606700e-014 -1.051000e-012 -2.818500e-011 1.239500e-010
LMATRIX
 4.261400e-007 9.662400e-008 2.530600e-008 6.682000e-009 1.766700e-009 4.727800e-010
 9.662400e-008 4.213200e-007 9.539600e-008 2.498300e-008 6.602200e-009 1.766700e-009
 2.530600e-008 9.539600e-008 4.210100e-007 9.531900e-008 2.498300e-008 6.682000e-009
 6.682000e-009 2.498300e-008 9.531900e-008 4.210100e-007 9.539600e-008 2.530600e-008
 1.766700e-009 6.602200e-009 2.498300e-008 9.539600e-008 4.213200e-007 9.662400e-008
 4.727800e-010 1.766700e-009 6.682000e-009 2.530600e-008 9.662400e-008 4.261400e-007
GMATRIX
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
RMATRIX
 7.225300e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 7.225300e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 7.225300e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 7.225300e+000 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 7.225300e+000 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 7.225300e+000
END RLGC
.ends six_coupled_traces
" ) ) ) )
 (PackagedDevice
  (six_pin_comp
   (IbisPinMap
    (4
     (signal_model CDSDefaultOutput )
     (ground_bus gndbus6 )
     (power_bus pwrbus1 )
     (WireNumber 4 ) )
    (5
     (signal_model CDSDefaultOutput )
     (ground_bus gndbus6 )
     (power_bus pwrbus1 )
     (WireNumber 5 ) )
    (2
     (signal_model CDSDefaultInput )
     (ground_bus gndbus6 )
```

```
    (power_bus pwrbus1 )
    (WireNumber 2 ) )
   (3
    (signal_model CDSDefaultInput )
    (ground_bus gndbus6 )
    (power_bus pwrbus1 )
    (WireNumber 3 ) )
   (1
    (signal_model POWER )
    (bus pwrbus1 )
    (WireNumber 1 ) )
   (6
    (signal_model GND )
    (bus gndbus6 )
    (WireNumber 6 ) ) )
  (Manufacturer Willis )
  (PackageModel six_conductor_BC ) ) )
(LibraryVersion 136.2 ) )
```

a. `PowerGroundCircuits`

b. `SignalCircuits`

The SubCircuits sub-parameters under `BlockCircuits` indicates how third party models for power and ground networks can be included. The optional `UnusedTerminalsTermination` allows the user to specify termination for the unused pins.

4. SubCircuits

`SubCircuits` is a required sub-parameter of `CircuitModels`. It is used to define a Spice sub-circuit. You have to define the Spice sub-circuit – which is usually written in ESpice or HSpice – inside the two double quotes of the `SubCircuits` sub-parameter. A comment inside the two double quotes after the `SubCircuits` sub-parameter has to be preceded by an asterisk (`*`) and not a semicolon (`;`) as described in Comments. Following is the syntax for the `SubCircuits` sub-parameter:

```
(SubCircuits
".subckt <name_of_subckt> <input/pad> <output/pin>
")

<pin_number> maps the sub-circuit to its corresponding pins.
<name_of_subckt> gives the name of a SPICE sub-circuit.
<input/pad> is to the buffer side (die pad) for a package model.
<output/pin> is to the component pin side.
```

Note: The order of the <*input/pad*> and <*output/pin*> in the `.subckt` definition cannot be altered. See the example in SingleLineCircuits on page 59.

Here's an example of usage of SubCircuits:

```
(PackageModel
```

```
(CircuitModels
  (SingleLineCircuits
    ; See part 1.(a) to see the SingleLineCircuits description for
    ; this example
    ...
  )
  (SubCircuits
  ".subckt longsingle_dc_wiremin 1 2
  r13 1 3 0.002 l=1n
   t32 3 0 2 0 z0=40 td=0.05n
  .ends longsingle_dc_wiremin

  .subckt longsingle_dc_wire 1 2
  r13 1 3 0.002 l=3n
  t32 3 0 2 0 z0=40 td=0.1n
  .ends longsingle_dc_wire

  .subckt longsingle_dc_wiremax 1 2
  r13 1 3 0.002 l=4n
  t32 3 0 2 0 z0=40 td=0.2n
  .ends longsingle_dc_wiremax") ) )
```

5. PinNameToNumber

   `PinNameToNumber` is an optional sub-parameter of `CircuitModels`. It maps pin-names to corresponding wire-numbers, thus associating each pin with an index in the RLGC matrix. `PinNameToNumber` does not have any sub-parameters under it. `PinNameToNumber` is not necessary if the pin-names are numeric (for example, `A1`, `B5`, `OE`). This information essentially duplicates the `WireNumber` (if specified) in PackagedDevice. If both are present in a given circumstance, then the `WireNumber` from PackagedDevice overrides the information here. This allows special cases and pin renaming. Here's an example of usage of `PinNameToNumber`:

```
(PackageModel
  ("CDS_Pkg4DIP"               ; Name of this PackageModel
    (PinNameToNumber           ; PinNameToNumber definition
      ("A1" "1")               ; Pin A1 maps to WireNumber 1
      ("A2" "2")               ; Pin A2 maps to WireNumber 2
      ("A3" "3")               ; Pin A3 maps to WireNumber 3
      ("A4" "4") )             ; Pin A4 maps to WireNumber 4
      ...
    )
)
```

RLGC

`RLGC` is a required parameter for PackageModel. It is used to describe pin parasitics, which represent the parasitics matrices at different frequencies. When a PackageModel is defined, the pin parasitic definition of `CircuitModels` within the PackageModel keyword is taken

with higher precedence than of an RLGC definition. In other words, when both CircuitModels and RLGC are specified, the pin parasitics definition under the RLGC parameter is ignored and pin parasitics under the CircuitModels parameter is used instead.

There are three different formats for RLGC as listed below. The RLGC parameters are defined in different ways depending on which format is used. For more information see RLGC Sub-parameters Descriptions.

BandedSymmetricMatrix	BandedSymmetricMatrix is used to describe the coupling relationship between pins when only the nearest rows of pins are coupled.
SymmetricMatrix	SymmetricMatrix is used to describe a fully coupled model.
SparseSymmetricMatrix	SparseSymmetricMatrix is used when a random assortment of pins are coupled.

RLGC Sub-parameters Descriptions

This section contains descriptions and examples of the sub-parameters of RLGC.

1. Frequency

   The value of *Frequency* is required for an RLGC definition. It allows the user to set the frequency point (in Hertz) at which the RGLC matrix is extracted. *Frequency* is a required sub-parameter of RLGC but it is not a sub-parameter itself, which means that the word *Frequency* does not (and should not) appear in a DML file but instead its value is specified as shown in the example below.

```
(RLGC
  (0                          ; 0 Hz of frequency for RLGC matrix
    ...
```

2. dimension

   dimension is a required sub-parameter of RLGC. It is defined as the number of pins of a package. It is also the matrix dimension of RLGC parasitics. The dimension sub-parameter has the same syntax for all the three formats of RLGC, which are BandedSymmetricMatrix, SymmetricMatrix, and SparseSymmetricMatrix as shown below.

```
(dimension D_NUMBER)
```

   D_NUMBER is the value of the dimension. Here's an example:

```
(SparseSymmetricMatrix
  (dimension "14")              ; This RLGC matrix has dimension = 14
  ...
)
```

3. band

   band is a required sub-parameter for RLGC. The notion of band in DML is different from that in IBIS (bandwidth). The band sub-parameter syntax is listed below under each different format:

   a. BandedSymmetricMatrix

   ```
   (band B_NUMBER)
   ```

   B_NUMBER must always be odd. It ranges from 1 to (2 * D -1), where D is the matrix dimension.

   For a BandedSymmetricMatrix, IBIS defines a keyword *bandwidth*. If the *bandwidth* is denoted as K (0 <= K <= D -1), then the data element (i, j) in a BandedSymmetricMatrix will always be zero as long as $|i - j| > K$. In a Package Model, if the pins are labeled in such a way that the labeling is according to their positions in the matrix, then pin i has at most K non-zero coupled neighbors j, which is equal ti $i + 1 <= D$. This constrains the *bandwidth* to be less than D. For zero *bandwidth*, the matrix is simply diagonal. In DML, the *B_NUMBER* is related to *bandwidth* as follows:

   B_NUMBER = 2 * K + 1 where K = 0, 1, ..., D - 1 and D is the matrix dimension.

   When the *B_NUMBER* is equal to 1 (when K = 0), it means that for row i, all elements (i and j, ! = i) in the matrix are zeros. Only the diagonal (i, i) can be non-zeros. This is the case when there is only self-coupling present in the matrix, which is on the diagonal.

   Note: "<=" refers to less than or equal to and "! =" refers to not equal to in the description above.

   For example:

   ```
   (BandedSymmetricMatrix
     (band 3)
     (dimension 2)
     ...
   ```

   In the example above, the *B_NUMBER* is equal to 3 and therefore, the *bandwidth* has to be 1 since K = (*B_NUMBER* - 1) / 2 = (3-1)/2 = 1. As an affect, the dimension of the matrix has to be at least 2. For the first row of this matrix, there are two elements that could be zero, namely, the self-coupling term (1, 1) and the coupling term (1, 2), which is required for *bandwidth* K = 1. For the two elements in the second row, one is a self-coupled element (2, 2) and the other one is a mutual

coupled element (2, 1), which is equal to the element (1, 2) because of the matrix symmetry and because *B_NUMBER* = 2 * K -1 = 3. These three elements are (1, 1), (1, 2), and (2, 1).

If the dimension in the example above was 3 instead of 2, then the first row, the element (1, 3), is zero since the *bandwidth* is 1. For the second row, the *bandwidth* covers the element (2, 3) while the *B_NUMBER* covers elements (2, 2), (2, 3), and (3, 2).

b. SymmetricMatrix

```
(band B_NUMBER)
```

SymmetricMatrix is used to describe a fully coupled model. Since the SymmetricMatrix may or may not be a BandedSymmetricMatrix, all of the elements in this matrix could be non-zero elements. Therefore, the *bandwidth* has to be the largest possible value, which is D - 1. Hence, the *B_NUMBER* can be calculated as follows:

B_NUMBER = 2 * D – 1 where D is the matrix dimension.

The example for BandedSymmetricMatrix shown above could also be an example for SymmetricMatrix since the *B_NUMBER* equals 3 and dimension equals 2 and *B_NUMBER* = 2 * D - 1 = 2 * 2 - 1 = 3. Following is another example of SymmetricMatrix:

```
(SymmetricMatrix
  (band 5)
  (dimension 3)
  ...
)
```

In the example above, the *B_NUMBER* = 2 * D - 1 = 2 * 3 -1 = 5. In this case, the bandwidth, K = D -1 = 3 -1 = 2

c. SparseSymmetricMatrix

```
(band B_NUMBER)
```

The `band` sub-parameter is usually not defined for the SparseSymmetricMatrix format for RLGC since it has a random assortment of pins that are coupled. An example of its usage is not shown here.

4. data

`data` is a required sub-parameter of `RLGC`. It is used to list the matrix data entries for `RLGC`. You can define it in two different ways depending on which format of RLGC you use. The data definitions are listed below under each different format:

a. BandedSymmetricMatrix

```
(data data_values)
```

data_values includes the matrix entries of RLGC. Each entry is separated with a space. The total number of data entries for a BandedSymmetricMatrix is $N = SUM$ (D - i), where i = 0, 1, 2, ... K, and K = (*B_NUMBER* - 1) / 2. D is the matrix dimension. The *bandwidth*, K, is related to the *B_NUMBER* by K = (*B_NUMBER* -1) / 2. Therefore, N is equal to (K + 2) * [2 * D - (K + 1)] / 2 = (4 * D - *B_NUMBER* -1) * (*B_NUMBER* + 3) / 8.

For example:

```
(BandedSymmetricMatrix
  (band 27)
  (dimension 14)
  (data "2.631284E-09 9.014512E-10 5.038304E-10
         3.486299E-10 2.649051E-10 2.16809E-10
  ...
  ")
)
```

b. SymmetricMatrix

The data parameter for SymmetricMatrix is defined the same way as for BandedSymmetricMatrix with the *B_NUMBER* equal to 2 * D – 1 or bandwidth, K = D -1. The total number of data entries for a SymmetricMatrix is $N = D * (D + 1) / 2$ where D is the matrix dimension.

c. SparseSymmetricMatrix

```
(data "<r_num> <c_num> <value> ...")
<r_num> and <c_num> are the row and column numbers of a non-zero entry.
<value> is the entry's value.
```

The formatting of the data is ignored. The numbers are simply taken in triplets, the first two being the row and column with the third equal to the matrix element. The order of the entries is unimportant, but it is an error to have row > column for any entry. Give only the upper triangle and the diagonal of the matrix. The lower triangle is assumed to be equal to the transpose of the upper triangle. For example:

```
(SparseSymmetricMatrix
  (dimension 14)
    (data "1 1 345.0f
           2 3 -45.7f
           ...
    ")
  ...
)
```

Examples of Connector Models

Connector models are basically models that are used to define connectors. These models are usually defined under the PackageModel keywords. Following are 3 examples of connector models:

Example 1:

```
(PackageModel
  (CircuitModels
    (edspkgmdl2
      (SingleLineCircuits
        ("1-208 "
          (SubCircuitName simple ) ) )
      (SubCircuits "
      .subckt simple 1 2
      td 1 0 2 0 z0=80 td=1.2n
      .ends simple" ) ) )
```

Example 2:

```
(PackageModel
  (CircuitModels
    (cisco2
      (SingleLineCircuits
        ("1-240"
          (SubCircuitName
            (typical longsingle_dc_wire ) ) ) )
      (SubCircuits
        (typical "
        .subckt longsingle_dc_wire 1 2
        R1 1 3 .008
        L1 3 4 1.372NH
        C1 4 0 .378PF
        L2 4 5 1.58NH
        C2 5 0 4.9PF
        L3 5 6 2.4NH
        C3 6 0 .11PF
        R2 6 2 .008
        .ends longsingle_dc_wire
```

Example 3:

```
(PackageModel
  ("dip14plastic_largemtm"
    (PinNameToNumber
      ("A1" 1)
      ("A2" 2)
      ("A3" 3)
```

```
      ("A4" 4)
      ("A5" 5)
      ("A6" 6)
      ("A7" 7)
      ("A8" 8)
      ("A9" 9)
      ("A10" 10)
      ("A11" 11)
      ("A12" 12)
      ("A13" 13)
      ("A14" 14) )
; Single Line circuits are approximated by a "slice" through the
; complete connector model.  This could be arrived at several ways, a
; good way would be to terminate all unused lines in their expected
; impedance and measure the response of a characteristic single
; line...or a cruder method is just use the self values of the full
; model...as was done here.  Note Ctotal_i = Co_i + Cm_ij, j=1-N, and
; i is not equal to j.
    (SingleLineCircuits
      (1
        (SubCircuitName
          (minimum longsingle_dc_wiremin)
          (typical longsingle_dc_wire)
          (maximum longsingle_dc_wiremax) ) )
      (2
        (SubCircuitName
          (minimum longsingle_dc_wiremin)
          (typical longsingle_dc_wire)
          (maximum longsingle_dc_wiremax) ) )

      (4-14
        (SubCircuitName
          (minimum longsinglewiremin)
          (typical longsinglewire)
          (maximum longsinglewiremax) ) ) )

; This is a partial listing of the subcircuits used here:

    (SubCircuits”
    .subckt longsingle_dc_wire 1 2
    r13 1 3 0.002 l=3n
    t32 3 0 2 0 z0=40 td=0.1n
    .ends longsingle_dc_wire

    .subckt longsinglewire 1 2
    r13 1 3 0.002 l=3n
    t32 3 0 2 0 z0=70 td=0.1n
    .ends longsinglewire

    .subckt longsingle_dc_wiremin 1 2
    r13 1 3 0.002 l=1n
```

```
    t32 3 0 2 0 z0=40 td=0.05n
    .ends longsingle_dc_wiremin")
; Coupled Line circuits are made up of a number of sub-groups, to be
; used for different coupling windows.  Note the pin map here and the
; actual pin geometry is important to keep consistent. There can be
; multiple sets of coupled line circuits applicable to different
; speeds of operation. The organization of information is identical to
; SingleLine Circuits except for the addition of Terminals

    (CoupledLineCircuits
      (13
        (Terminals 13.in 13.out 12.in 12.out)
        (SubCircuitName 2line))
      (11-12
        (Terminals x+1.in x+1.out x.in x.out x-1.in x-1.out)
        (SubCircuitName 3line))
      (10
        (Terminals 11.in 11.out 10.in 10.out 5.in 5.out)
        (SubCircuitName 3line))
      (5
        (Terminals 10.in 10.out 5.in 5.out 4.in 4.out)
        (SubCircuitName 3line))
      (3-4
        (Terminals x+1.in x+1.out x.in x.out x-1.in x-1.out)
        (SubCircuitName 3line))
      (2
        (Terminals 3.in 3.out 2.in 2.out 8.in 8.out)
        (SubCircuitName 2line_floating_tline)))

; This is a partial listing of the subcircuits used here:

    (SubCircuits"
    .subckt 2line 1 10 2 20
    ntl 1 2 0 10 20 0 l=0.002
    DATAPOINTS RLGC MTL_2S_2R_2060
    FREQUENCY=0
    CMATRIX
      1.544500e-10 -7.567500e-13
      -7.567500e-13 1.544500e-10
    LMATRIX
      3.746000e-07 1.835400e-09
      1.835400e-09 3.746000e-07
    GMATRIX
      0.000000e+00 -0.000000e+00
      -0.000000e+00 0.000000e+00
    RMATRIX
      9.289700e+00 0.000000e+00
      0.000000e+00 9.289700e+00
    END RLGC
    .ends  2line
```

```
.subckt 2line_floating_tline 1 10 2 20 100 200
r13 1 3 0.001 l=2n
r24 2 4 0.001 l=2n

* dc trace
*rdc 100 200 0.001 l=8n
xdc 100 200 longsinglewire

* 2 line floating coupled transmission line
ntl 3 4 100 10 20 200 l=0.002
DATAPOINTS RLGC MTL_2S_2R_2060
FREQUENCY=0
CMATRIX
  1.544500e-10 -7.567500e-13
  -7.567500e-13 1.544500e-10
LMATRIX
  3.746000e-07 1.835400e-09
  1.835400e-09 3.746000e-07
GMATRIX
  0.000000e+00 -0.000000e+00
  -0.000000e+00 0.000000e+00
RMATRIX
  9.289700e+00 0.000000e+00
  0.000000e+00 9.289700e+00
END RLGC
.ends  2line_floating_tline

.subckt 3line 1 10 2 20 3 30
ntl 1 2 3 0 10 20 30 0 l=0.002
DATAPOINTS RLGC MTL_2S_3R_6212
FREQUENCY=0
CMATRIX
  1.552500e-10 -1.008500e-11 -3.160200e-13
  -1.008500e-11 1.557900e-10 -8.312100e-12
  -3.160200e-13 -8.312100e-12 1.550000e-10
LMATRIX
  3.742600e-07 2.433800e-08 2.068300e-09
  2.433800e-08 3.740300e-07 2.010800e-08
  2.068300e-09 2.010800e-08 3.743700e-07
GMATRIX
  0.000000e+00 -0.000000e+00 -0.000000e+00
  -0.000000e+00 0.000000e+00 -0.000000e+00
  -0.000000e+00 -0.000000e+00 0.000000e+00

RMATRIX
  9.289700e+00 0.000000e+00 0.000000e+00
  0.000000e+00 9.289700e+00 0.000000e+00
  0.000000e+00 0.000000e+00 9.289700e+00
END RLGC
.ends 3line”) ) )
```

Connector Swath Models

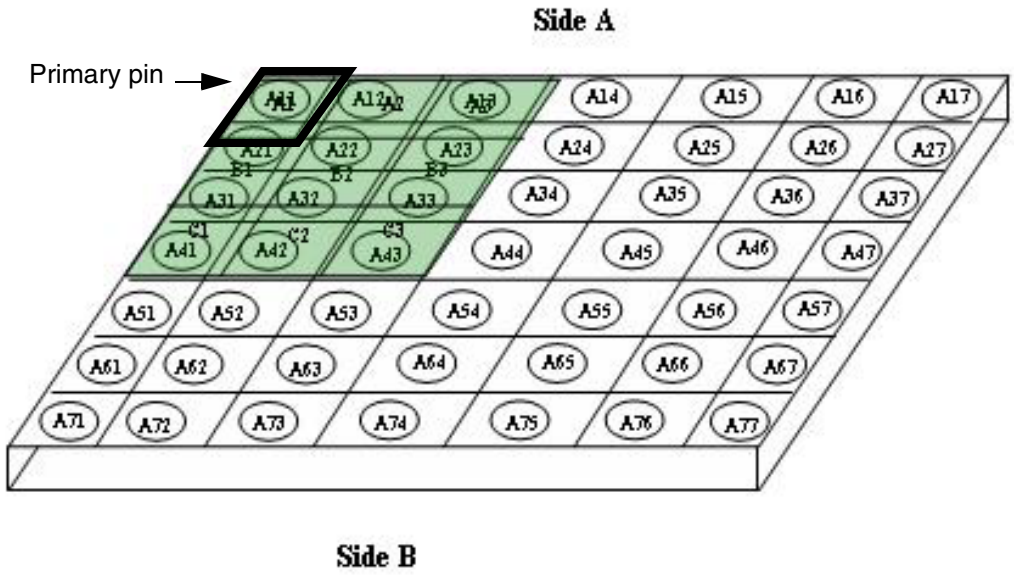
DML syntax for swath models simplifies the ability to model coupled connectors with very high pin counts. The parameters associated with the `Connector` keyword in DML describe a small matrix which is expanded during board-level SI simulations to represent much larger rectangular interconnects. You select swath models that represent a corner, edge, or center pin of the target connector, as described and depicted in the following sections.

Swath Selection Scenarios

Primary Corner Pin

For this case, the primary pin in the swath grid is a corner pin, which determines the model's placement. The swath model parameters must match the target connector's primary pin and describe a total pin number equal to or greater than the number of pins being simulated.The swath model placement is represented by the shadowed area in Figure 2-5.

Figure 2-5 Corner Pin Location

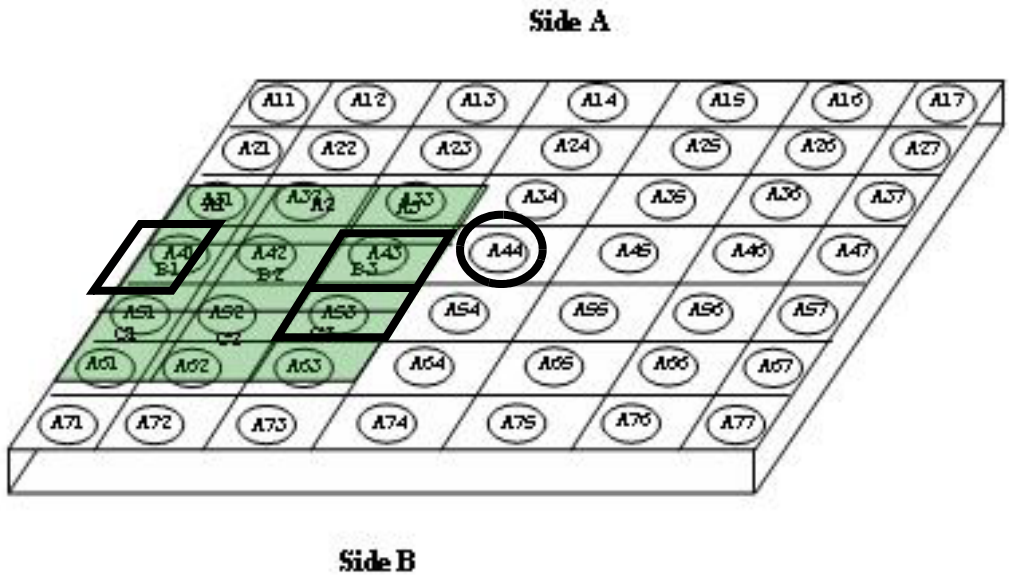


Primary Edge Pin

For this case, the primary pin in the swath grid is an edge pin. The location of the swath model is determined by the primary pin and the two pins closest to the center pin of the interconnect,

as shown in Figure 2-6. The swath model parameters must match the target connector's primary pin and describe a total pin number equal to or greater than the number of pins being simulated.

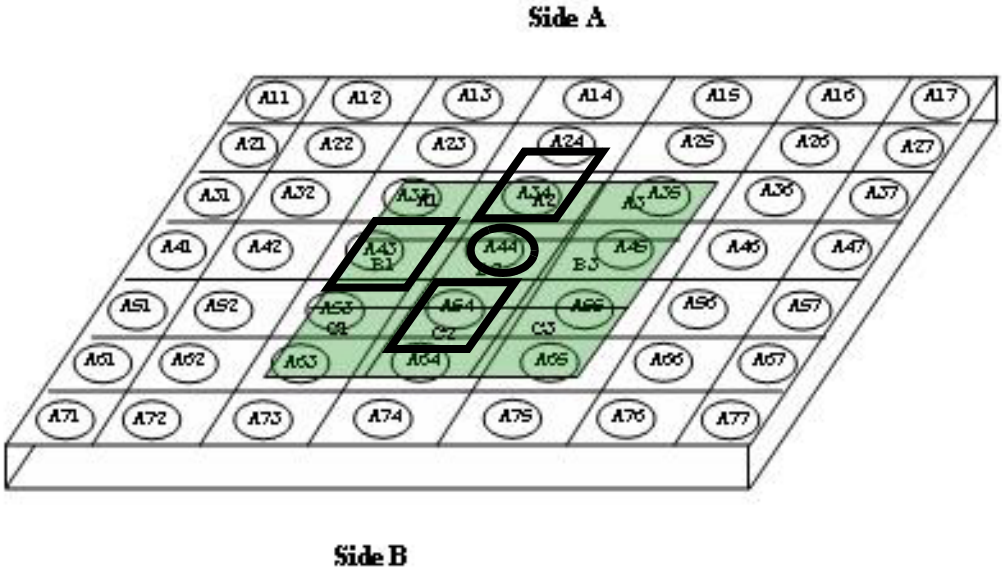
Figure 2-6 Edge Pin Location



Primary Center Pin

For this case, the primary pin in the swath grid is a center pin. Typically, you would use a center pin scenario when a corner or edge pin cannot be located within the defined swath area. In such a case, the location of the swath model is determined by the three pins closest to the center pin of the interconnect, as shown in Figure 2-7.

Figure 2-7 Center Pin Location



In each of the scenarios described above, single line models (SLMs) will be used for floating pins; that is, pins that are not represented in the swath model.

Geometry Window Considerations

Pins and nets in swath model simulations are determined by the geometry window settings in your design database. Nets and pins within the area defined by the geometry window will be simulated according to the parameters defined in the swath model. Pins that are defined in the model but are located outside the perimeter of the geometry window will be terminated according to the model's TerminationType parameter values, OPEN or matched SHORT (connected to a match load/ground).

As examples of this:

- If the geometry window contains only a single pin, the model will select from a minimum two-dimension scenario. If there is no available two-dimensional swath, a single line model is selected.
- If the geometry window contains three pins and the only available swath model defines two rows and two columns (2x2), the fourth pin is terminated.

Sample Swath Connector Model

The following is an annotated example of a swath connector model that describes a 5x5 connector.

```
("Connector_test.dml"
  ; This is a 5X5 connector sample DML file
   (PackagedDevice
      ; New connector model for swath and ICM
      (Connectorsample
       ; The Connector section is compatible with the ICM specification (version 1.1)
       ; The Connector model will contain all sub-parameters as PackageModel.
       ; It will also has some new sub-parameters for ICM (swath model)
          (Connector
          ; The PinNameToNumber maps pin-name to cooresponding wire-numbers.
           ; This is not necessary each if the pin-names are numeric.
             (PinNameToNumber
                 (A1 1 )
                (A2 2 )
                 (A3 3 )
                 (A4 4 )
                 (A5 5 )
                 (B1 6 )
                 (B2 7 )
                 (B3 8 )
                 (B4 9 )
                 (B5 10 )
                 (C1 11 )
                 (C2 12 )
                 (C3 13 )
                 (C4 14 )
                 (C5 15 )
                 (D1 16 )
                 (D2 17 )
                 (D3 18 )
                 (D4 19 )
                 (D5 20 )
                 (E1 21 )
                 (E2 22 )
                 (E3 23 )
                 (E4 24 )
                 (E5 25 )

             )
          ; TreePath/ConnectorPinMap describe the location of  the
           ; pins on the connector model.
          ; These are consistent with ICM.

             (TreePath
```

```
   (Side
      (A
         (PinMap   A_map)
      )
   )
)
(ConnectorPinMap
   (A_map
      (NumOfRows 5)
      (NumOfColumns 5)
      (PinList
         (A1
         )
         (A2
         )
         (A3
         )
         (A4
         )
         (A5
         )
         (B1
         )
         (B2
         )
         (B3
         )
         (B4
         )
         (B5
         )
         (C1
         )
         (C2
         )
         (C3
         )
         (C4
         )
         (C5
         )
         (D1
         )
         (D2
         )
         (D3
         )
         (D4
         )
         (D5
         )
```

```
            (E1
            )
            (E2
            )
            (E3
            )
            (E4
            )
            (E5
            )
         )
      )
   )
    ; SwathModel section includes all singleline/coupledline swath modules
; Each swath module contains size/terminals location/subcircuit/
; termination type/match load value/swath type.
   (SwathModel
      (my_swath_1
         (SingleLineCircuits
            ("1-25"
              (SubCircuitName my_slm_1)
            )
         )
         (SubCircuits "
            .subckt my_slm_1 1 2
            r1 1 3 50m
            t1 3 0 4 0 z0=32 td=100p
            r2 4 2 50m
            .ends my_slm_1"
   )
      )
      (my_swath_2
         (NumOfRows  2)
         (NumOfColumns 2)
 ;Termination type should determine how do terminate extra pins.
;Options include OPEN and SHORT. If it is short, it must have a match
;load resistance value.
         (TerminationType  SHORT)
         (MatchLoad   50)
         ; The swath model type could provide information of Corner/Edge/Center
         (Type
            (Left_edge  0)
            (Right_edge  0)
            (Top_edge  0)
            (Bottom_edge  0)
         )
         (CoupledLineCircuits
            (Terminals A1.in A1.out A2.in A2.out B1.in B1.out B2.in B2.out)
            (SubCircuitName my_mlm_2)
          ; SwathNodeNames includes the physical row and column numbers of
          ; swath element which give the relative physical location of each node.
```

```
        ; This is consistent with ICM
(SwathNodeNames
            (A1.in
               (1 1)
            )
            (A2.in
               (1 2)
            )
            (B1.in
               (2 1)
            )
            (B2.in
               (2 2)
            )
         )
      )
      (SubCircuits "
         .subckt my_mlm_2 1 2 3 4 5 6 7 8
         r1 1 21 2.55e-3
         l1 21 2 1.44e-9
         r2 3 31 2.55e-3
         l2 31 4 1.44e-9
         r3 5 51 2.55e-3
         l3 51 6 1.44e-9
         r4 7 71 2.55e-3
         l4 71 8 1.44e-9
         .ends my_mlm_2 "
      )
   )
   (my_swath_4
      (NumOfRows  2)
      (NumOfColumns 2)
      (TerminationType  SHORT)
      (MatchLoad   75)
      (Type
         (Left_edge  1)
         (Right_edge  0)
         (Top_edge  1)
         (Bottom_edge  0)
      )
      (CoupledLineCircuits
         (Terminals A1.in A1.out A2.in A2.out B1.in B1.out B2.in B2.out)
         (SubCircuitName my_mlm_4)
         (SwathNodeNames
            (A1.in
               (1 1)
            )
            (A2.in
               (1 2)
            )
            (B1.in
```

```
            (2 1)
         )
         (B2.in
            (2 2)
         )
      )
   )
   (SubCircuits "
      .subckt my_mlm_4 1 2 3 4 5 6 7 8
      r1 1 21 2.55e-3
      l1 21 2 1.44e-9
      r2 3 31 2.55e-3
      l2 31 4 1.44e-9
      r3 5 51 2.55e-3
      l3 51 6 1.44e-9
      r4 7 71 2.55e-3
      l4 71 8 1.44e-9
      .ends my_mlm_4 "
   )
)
(my_swath_6
   (NumOfRows  2)
   (NumOfColumns 2)
   (TerminationType  OPEN)
   (Type
      (Left_edge  1)
      (Right_edge  0)
      (Top_edge  0)
      (Bottom_edge  0)
   )
   (CoupledLineCircuits
      (Terminals A1.in A1.out A2.in A2.out B1.in B1.out B2.in B2.out)
      (SubCircuitName my_mlm_6)
      (SwathNodeNames
         (A1.in
            (1 1)
         )
         (A2.in
            (1 2)
         )
         (B1.in
            (2 1)
         )
         (B2.in
            (2 2)
         )
      )
   )
   (SubCircuits "
      .subckt my_mlm_6 1 2 3 4 5 6 7 8
      r1 1 21 2.55e-3
```

```
                     l1 21 2 1.44e-9
                     r2 3 31 2.55e-3
                     l2 31 4 1.44e-9
                     r3 5 51 2.55e-3
                     l3 51 6 1.44e-9
                     r4 7 71 2.55e-3
                     l4 71 8 1.44e-9
                     .ends my_mlm_6 "
                  )
               )
               (my_swath_3
                  (NumOfRows  3)
                  (NumOfColumns 3)
                  (TerminationType  SHORT)
                  (MatchLoad   75)
                  (Type
                     (Left_edge  0)
                     (Right_edge  0)
                     (Top_edge  0)
                     (Bottom_edge  0)
                  )
                  (CoupledLineCircuits
                   (Terminals A1.in A1.out A2.in A2.out A3.in A3.out B1.in B1.out B2.in B2.out B3.in
B3.out C1.in C1.out C2.in C2.out C3.in C3.out)
                     (SubCircuitName my_mlm_3)
                     (SwathNodeNames
                        (A1.in
                           (1 1)
                        )
                        (A2.in
                           (1 2)
                        )
                        (A3.in
                           (1 3)
                        )
                        (B1.in
                           (2 1)
                        )
                        (B2.in
                           (2 2)
                        )
                        (B3.in
                           (2 3)
                        )
                        (C1.in
                           (3 1)
                        )
                        (C2.in
                           (3 2)
                        )
                        (C3.in
```

```
                        (3 3)
                     )
                  )
               )
               (SubCircuits "
                  .subckt my_mlm_3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
                  r1 1 21 2.55e-3
                  l1 21 2 1.44e-9
                  r2 3 31 2.55e-3
                  l2 31 4 1.44e-9
                  r3 5 51 2.55e-3
                  l3 51 6 1.44e-9
                  r4 7 71 2.55e-3
                  l4 71 8 1.44e-9
                  r5 9 91 2.55e-3
                  l5 91 10 1.44e-9
                  r6 11 111 2.55e-3
                  l6 111 12 1.44e-9
                  r7 13 131 2.55e-3
                  l7 131 14 1.44e-9
                  r8 15 151 2.55e-3
                  l8 151 16 1.44e-9
                  r9 17 171 2.55e-3
                  l9 171 18 1.44e-9
                  .ends my_mlm_3 "
               )
            )
         (my_swath_5
               (NumOfRows  3)
               (NumOfColumns 3)
               (TerminationType  SHORT)
               (MatchLoad   75)
               (Type
                  (Left_edge  1)
                  (Right_edge  0)
                  (Top_edge  1)
                  (Bottom_edge  0)
               )
               (CoupledLineCircuits
                (Terminals A1.in A1.out A2.in A2.out A3.in A3.out B1.in B1.out B2.in B2.out B3.in
B3.out C1.in C1.out C2.in C2.out C3.in C3.out)
                  (SubCircuitName my_mlm_5)
                  (SwathNodeNames
                     (A1.in
                        (1 1)
                     )
                     (A2.in
                        (1 2)
                     )
                     (A3.in
                        (1 3)
```

```
            )
            (B1.in
               (2 1)
            )
            (B2.in
               (2 2)
            )
            (B3.in
               (2 3)
            )
            (C1.in
               (3 1)
            )
            (C2.in
               (3 2)
            )
            (C3.in
               (3 3)
            )
         )
      )
      (SubCircuits "
         .subckt my_mlm_5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
         r1 1 21 2.55e-3
         l1 21 2 1.44e-9
         r2 3 31 2.55e-3
         l2 31 4 1.44e-9
         r3 5 51 2.55e-3
         l3 51 6 1.44e-9
         r4 7 71 2.55e-3
         l4 71 8 1.44e-9
         r5 9 91 2.55e-3
         l5 91 10 1.44e-9
         r6 11 111 2.55e-3
         l6 111 12 1.44e-9
         r7 13 131 2.55e-3
         l7 131 14 1.44e-9
         r8 15 151 2.55e-3
         l8 151 16 1.44e-9
         r9 17 171 2.55e-3
         l9 171 18 1.44e-9
         .ends my_mlm_5 "
      )
   )
(my_swath_7
      (NumOfRows  3)
      (NumOfColumns 3)
      (TerminationType  SHORT)
      (MatchLoad   75)
      (Type
         (Left_edge  0)
```

```
                     (Right_edge  0)
                     (Top_edge  1)
                     (Bottom_edge  0)
                  )
                  (CoupledLineCircuits
                   (Terminals A1.in A1.out A2.in A2.out A3.in A3.out B1.in B1.out B2.in B2.out B3.in
B3.out C1.in C1.out C2.in C2.out C3.in C3.out)
                     (SubCircuitName my_mlm_7)
                     (SwathNodeNames
                        (A1.in
                           (1 1)
                        )
                        (A2.in
                           (1 2)
                        )
                        (A3.in
                           (1 3)
                        )
                        (B1.in
                           (2 1)
                        )
                        (B2.in
                           (2 2)
                        )
                        (B3.in
                           (2 3)
                        )
                        (C1.in
                           (3 1)
                        )
                        (C2.in
                           (3 2)
                        )
                        (C3.in
                           (3 3)
                        )
                     )
                  )
                  (SubCircuits "
                     .subckt my_mlm_7 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
                     r1 1 21 2.55e-3
                     l1 21 2 1.44e-9
                     r2 3 31 2.55e-3
                     l2 31 4 1.44e-9
                     r3 5 51 2.55e-3
                     l3 51 6 1.44e-9
                     r4 7 71 2.55e-3
                     l4 71 8 1.44e-9
                     r5 9 91 2.55e-3
                     l5 91 10 1.44e-9
                     r6 11 111 2.55e-3
```

```
                    l6 111 12 1.44e-9
                    r7 13 131 2.55e-3
                    l7 131 14 1.44e-9
                    r8 15 151 2.55e-3
                    l8 151 16 1.44e-9
                    r9 17 171 2.55e-3
                    l9 171 18 1.44e-9
                    .ends my_mlm_7 "
                )
            )
        )
    )
)
```

How to Create a PackageModel

This section describes the different ways a user can create a PackageModel, namely, using the model translation utilities such as `ibis2signoise` and `quad2signoise`, using Allegro Package SI and by manually writing DML files. Note that PackageModel CANNOT be created using the GUI of Allegro SI. The different methods of creating a PackageModel are described here in decreasing order of suggested method and common practices of model creation from first to last.

Allegro Package SI

It is possible to create a PackageModel from an existing APD database using Allegro Package SI and Turbo Package Analyzer (TPA) from Ansoft. There are several options but the basic process involves invoking the TPA field solver from Allegro Package SI. See the Allegro Package SI documentation for more details.

Model Translation Utilities (ibis2signoise & quad2signoise)

You can translate models that are written in other modeling languages, such as Ibis and Quad, into DML using the model translation tools called `ibis2signoise` and `quad2signoise` respectively. Refer to Chapter 8, “Model Translation,” to learn how this can be done.

Manual creation of a PackageModel

You can create a PackageModel by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` file has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token. Listed below are the steps to create a PackageModel:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the PackageModel template from Appendix B, “PackageModel Template,” into this file.
3. Edit the DML file, which has the PackageModel template as needed for the new model.
4. Save the file and run `dmlcheck` to check for syntax errors. See Chapter 6, “dmlcheck Utility,”, for more information.

Note: PackageModel template:

- This template is a skeleton of the DML file, which the user can copy and paste into a text editor and then customize to model a specific device.
- The user might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.
- Refer to the specific parameter and sub-parameter descriptions, which are listed in RLGC Sub-parameters Descriptions, and CircuitModels Sub-parameters Descriptions, for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`PackageModel`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

BoardModel

BoardModel is a DML keyword that is used to model entire boards for situations in which the physical Allegro database is not available. It is usually referenced from a DesignLink and it contains Spice sub-circuits. At the outset, you should notice that BoardModel simulations are available only from Allegro SI and not from SigXplorer. These models are DML representations of allegro databases and have a number of simulation advantages when compared to board files. BoardModel is designed to facilitate the following:

- Allow multiboard simulation without the presence of a real board. In this respect BoardModel has the functionality of IBIS 3.0 Electrical board description (EBD). But this is more powerful than EBD since you can include coupled line circuits in BoardModel. To use the BoardModel this way the DesignLink has to point to the BoardModel rather than the `.brd` file. The program `brd2dml` will automatically create a BoardModel for an existing board. Only this BoardModel needs to be shipped to a downstream user for signal integrity analysis in a multiboard environment.
- You can also use BoardModel in conjunction with an existing board. In this way you can force Allegro SI to use the interconnect models from the BoardModel during signal integrity simulation rather than the ratsnest or actual net.
- Advanced Modeling facility; Third party power/ground plane modeling. The SubCircuits sub-parameter under BlockCircuits indicates how third party models for power and ground networks can be included.

Using BoardModel:

You use BoardModel in the same way an Allegro board is used. A DesignLink is created for each board that needs to be connected. This is accomplished by editing the connector pin mapping between both boards. BoardModel is saved in DML libraries, which means the library must be loaded in order to ensure that the DesignLink can find the board.

By default the software will use a BoardModel over the Allegro database. When a BoardModel is used and called by the software the software creates a temporary Allegro database for the model, and this Allegro database is given the same name as the BoardModel. This means that if a valid version of the Allegro database exists in the run directory, the database will be overwritten by the temporary database. Before running with BoardModel, users should make sure that no conflicting data is present in the directory to prevent data loss.

Advantages of BoardModel:

- BoardModel contains all of the information necessary to run a simulation but does not have to contain redundant information. The information in the board model can be filtered such that only the required nets are in the BoardModel.
- You can create BoardModel with no coupling information (or with coupling information). This reduces the amount of crosstalk information, which is analyzed in order to increase the speed of the simulation. This also ensures that crosstalk problems are due to the "board under test" and not due to an add-in card for which little information may be available.
- You can create a BoardModel very quickly from other board models if an Allegro database is not available. Cutting and pasting from other models in order to obtain the desired model takes little effort. The syntax used is ESpice based.

Disadvantages of BoardModel:

- If an allegro database with the same name as the BoardModel exists in the directory it is over-written.
- You cannot change the xtalk geometry without re-translating or re-editing the model.
- You cannot immediately see the routing of a proprietary board.

An Electrical Board Description is not an IBIS file but is defined in the IBIS specification. A BoardModel corresponds to a "topspec" in a QUAD model. It will map to a `.brd` file, an electrical board description containing ESpiceDevice models for the nets on the board, and a large package model.

There are three main parameters for BoardModel, namely, `PinMap`, `CircuitModels`, and `Comps`. All these parameters are required and therefore there are no optional parameters under BoardModel. The following table shows the required parameters for BoardModel and their respective required and optional sub-parameters.

BoardModel Required Parameters and Their Respective Required and Optional Sub-Parameters

Required Parameters	Sub-parameters	
	Required	**Optional**
PinMap	*<pin interconnects>*	-

BoardModel Required Parameters and Their Respective Required and Optional Sub-Parameters

Required Parameters	Sub-parameters	
CircuitModels	SingleLineCircuits	CoupledLineCircuits
	SubCircuits	
Comps	Pins	dcvolt
	signal_model	

BoardModel Parameters and Sub-parameters Descriptions

This section contains descriptions and examples of BoardModel parameters and sub-parameters. Some of the examples that are used in this section are excerpts from a larger DML file called `cds_samples.dml` which is located in the *<cds_install_dir>*`/share/pcb/signal/` directory.

1. PinMap

   `PinMap` is a required parameter of BoardModel. It defines all the interconnects between connectors and devices for a specific pin. Here's an example of usage of `PinMap`:

```
(BoardModel
  ("brd_ex"
    (PinMap
      ("2" "J1 A4")              ; Shows J1-A4 no connect
      ("9" "J1 A44" "U1 36")     ; Shows J-A44 connected to U1-36
      ...
```

2. CircuitModels

   `CircuitModels` is a required parameter of BoardModel. It describes a board model using ESpice sub-circuit syntax. The `CircuitModels` parameter under BoardModel is defined similarly as it is under PackageModel. Refer to the PackageModel definition for CircuitModels on page 59, to learn more about its usage.

3. Comps

   `Comps` is a required parameter of BoardModel. It is used to define components in a board model. Here's an example of usage of `Comps`:

```
(BoardModel
  ("pci64brd"
    (Comps
      ("U2"                          ; <component_name>
```

```
    (Pins                              ; See Pins below
      (8                               ; <pin_number>
        (pinuse "GROUND" )             ; See pinuse below
        (dcvolt "0") )                 ; See dcvolt below
      (1
        (pinuse "POWER")
        (dcvolt "5") )
      (2
        (pinuse "BI") )
      (9
        (pinuse "OUT") )
      (3
        (pinuse "IN") )
      ...
    )
    (signal_model "p14u1_diffPair") )   ; See signal model below
...
) ) )
```

a. Pins

`Pins` is a required sub-parameter for `Comps`. It is used to define all the pins for a component in a BoardModel. It has two sub-parameters, `pinuse` and `dcvolt`, which are described below.

pinuse

`pinuse` is a required sub-parameter for `Pins`. It is used to define the function of a specific pin such as `GROUND`, `POWER`, `BI`, `IN`, and `OUT`.

dcvolt

dcvolt is an optional sub-parameter for Pins. It is used to define the DC Voltage for a pin (if necessary).

b. signal_model

`signal_model` is a required sub-parameter for `Comps`. It is defined similarly as the `signal_model` sub-parameter under the BufferDelay sub-parameter, which is under the IbisPinMap parameter. Refer to IbisDevice Sub-parameters Descriptions on page 34 for more information.

How to Create a BoardModel

This section describes the different ways a user can create and edit a BoardModel, namely, using model translation utilities such as `ibis2signoise` and `quad2signoise`, and by manually creating DML files. The different methods of creating a BoardModel are described here in decreasing order of suggested method and common practices of model creation from first to last.

Model Translation Utility (`brd2dml` translator)

The translator is run from the command line. Typing in `brd2dml` will display the help file. The translator is capable of translating all of the database information or selected information. You can select nets, components, geometry windows, and frequency dependence of the T-lines. This allows you to translate only the necessary simulation data, which reduces the amount of data and can increase the speed of a simulation. For example, you can reduce the Xeon processor model by selecting only the front side bus signals without any coupling. This reduces the number of signals and creates a board with no crosstalk effects.

Manual creation of a BoardModel

You can create a BoardModel by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token on page 19. Listed below are the steps to create a BoardModel model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the BoardModel template from Appendix C, “BoardModel Template,” into this file.
3. Edit the DML file, which has the BoardModel template as needed for a new model.
4. Save the file and run `dmlcheck` to check for syntax errors. See Chapter 6, “dmlcheck Utility,” for more information.

Note: BoardModel template:

- This template is a skeleton of the DML file, which you can copy and paste into a text editor and then customize to model a specific device.
- You might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.

- Refer to the specific parameter and sub-parameter descriptions, which are listed in BoardModel Parameters and Sub-parameters Descriptions on page 92, for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`BoardModel`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

IbisIOCell

The `IbisIOCell` keyword is used to define IBIS buffer models using IBIS compatible DML keywords. It is usually referenced from an IbisDevice model and is used to model driver and receiver buffers at the pin level. These models contain behavioral information about an I/O buffer, such as clamping diode VI Curve, rising waveform data, VI reference temperature info, and so on.

IbisIOCell Parameters and Sub-parameters Descriptions

This section contains descriptions and examples of IbisIOCell parameters and sub-parameters. Some of the examples that are used in this section are excerpts from a larger DML file called `cds_iocells.dml` which is located in the `<cds_install_dir>/share/pcb/signal/` directory.

1. Technology

   `Technology` is an optional parameter for IbisIOCell EXCEPT for a model that uses ECL technology. `Technology` is used to define the type of technology used for an IbisIOCell. It can have any value, but SigNoise recognizes only `ECL` as significant. The PullUp and PullDown VI curves of ECL IOCells are BOTH *wired* to the PullUpReferenceVoltage. For all other technologies the PullDown curve is wired to it's own PullDownReferenceVoltage supply. There is no IBIS keyword for `Technology`. Here's an example of `Technology`:

   ```
   (IbisIOCell
     ("TechExample"
       (Technology "CMOS")
   ...
   ```

2. Model

   `Model` is a required parameter for `IbisIOCell`. It contains general identifications. The IBIS Keyword for `Model` is `[Model]`. Following is an example of `Model`:

```
(IbisIOCell
  ("ModelIOExample"
    (Model
      (ModelType "IO")                          ; See ModelType below
      (C_comp                                   ; See C_comp below
        (minimum "3p")
        (typical "8p")
        (maximum "10p") )
      (Polarity "Non-Inverting")                ; See Polarity below
      (Enable "Active-High") )                  ; See Enable below
      (ModelSpec                                ; See ModelSpec below
        (VinhP                                  ; See VinhP below
          (typical "2.0") )
        (VinhM                                  ; See VinhM below
          (typical "1.6") )
        (VinlP                                  ; See VinlP below
          (typical "0.4") )
        (VinlM                                  ; See VinlM below
          (typical "-0.3") )
        (S_overshoot_high                       ; See S_overshoot_high below
          (minimum "5.0")
          (typical "5.5")
          (maximum "6.0") )
        (S_overshoot_low                        ; See S_overshoot_low below
          (typical "-0.5") )
        (D_overshoot_high                       ; See D_overshoot_high below
          (minimum "5.5")
          (typical "6.0")
          (maximum "6.5") )
        (D_overshoot_low                        ; See D_overshoot_low below
          (minimum "-1.0")
          (typical "-1.0")
          (maximum "-1.0") )
        (D_overshoot_time                       ; See D_overshoot_time below
          (minimum "20n")
          (typical "20n")
          (maximum "20n") )
        (Pulse_high                             ; See Pulse_high below
          (typical "3v") )
        (Pulse_low                              ; See Pulse_low below
         (typical "0v") )
        (Pulse_time                             ; See Pulse_time below
          (typical "3n") ) )
  ...
```

a. ModelType

`ModelType` is a required sub-parameter for `IbisIOCell`. The IBIS Keyword for `ModelType` is `[Model_type]. ModelType` may be one of the following SigNoise options:

Input (no pull-up, no pull-down, input logic)

Output (pull-up and/or pull-down, no input logic threshold)

Output_OpenPullUp (Output without pull-up or power_resistor)

Output_OpenPullDown (Output without pull-down or ground_resistor)

IO (Output with input logic threshold added)

IO_OpenPullUp (IO without pull-up or power_resistor)

IO_OpenPullDown (IO without pull-down or ground_resistor)

`ModelType` could also be one of the IBIS options. These IBIS options are mapped on input to the SigNoise model types as shown in IBIS-SigNoise Model Type Mapping and Descriptions table on page 99.

b. C_comp

`C_comp` is a required sub-parameter for `Model`. `C_comp` defines the silicon die capacitance. This value should not include the capacitance of the package. `C_comp` is considered as 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 IBIS keyword for `C_comp` is `C_comp`.

c. Polarity

`Polarity` is an optional sub-parameter for `Model`. It is an IBIS keyword that is not used by SigNoise. The value of `Polarity` can be either `Non-Inverting` or `Inverting`.

d. Enable

`Enable` is an optional sub-parameter for `Model`. It is an IBIS keyword that is not used by SigNoise. The value of `Enable` can be either `Active-High` or `Active-Low`.

e. ModelSpec

`ModelSpec` is an optional keyword of IbisIOCell. `ModelSpec` is used to specify static and dynamic overshoot limits, as well as pulse immunity limits. The IBIS keyword for `ModelSpec` is `[Model Spec]`. All sub-parameters of `ModelSpec` as listed below are optional sub-parameters of ModelSpec.

VinhP

`VinhP` is used to define the hysteresis threshold high max Vt+. Only the `typical` value is required for `VinhP`. You can specify the "`minimum`" and `maximum` values, but they are optional.

VinhM

`VinhM` is used to define the hysteresis threshold high min Vt+. Only the `typical` value is required for `VinhM`. You can specify the `minimum` and `maximum` values, but they are optional.

VinlP

`VinlP` is used to define the hysteresis threshold low max Vt-. Only the `typical` value is required for `VinlP`. You can specify the `minimum` and `maximum` values, but they are optional.

VinlM

`VinhP` is used to define the hysteresis threshold low min Vt-. Only the `typical` value is required for `VinlM`. You can specify the `minimum` and `maximum` values, but they are optional.

S_overshoot_high

`S_overshoot_high` is used to define the static overshoot high voltage. Only the `typical` value is required for `S_overshoot_high`. You can specify the `minimum` and `maximum` values, but they are optional.

S_overshoot_low

`S_overshoot_low` is used to define the static overshoot low voltage. Only the `typical` value is required for `S_overshoot_low`. You can specify the `minimum` and `maximum` values, but they are optional.

D_overshoot_high

`D_overshoot_high` is used to define the dynamic overshoot high voltage. Only the `typical` value is required for `D_overshoot_high`. You can specify the `minimum` and `maximum` values, but they are optional.

D_overshoot_low

`D_overshoot_low` is used to define the dynamic overshoot low voltage. Only the `typical` value is required for `D_overshoot_low`. You can specify the `minimum` and `maximum` values, but they are optional.

D_overshoot_time

`D_overshoot_time` is used to define the dynamic overshoot time. Only the `typical` value is required for `D_overshoot_time`. You can specify the `minimum` and `maximum` values, but they are optional.

Pulse_high

`Pulse_high` is used to define the pulse immunity high voltage. Only the `typical` value is required for `Pulse_high`. You can specify the `minimum` and `maximum` values, but they are optional.

Pulse_low

`Pulse_low` is used to define the pulse immunity low voltage. Only the `typical` value is required for `Pulse_low`. You can specify the `minimum` and `maximum` values, but they are optional.

Pulse_time

`Pulse_time` is used to define the pulse immunity time. Only the `typical` value is required for `Pulse_time`. You can specify the `minimum` and `maximum` values can, but they are optional.

IBIS-SigNoise Model Type Mapping and Descriptions

IBIS Model_Type	SigNoise ModelType	Description
Input	IbisInput	These models must have `Vinl` and `Vinh` defined. Otherwise, the parser issues a warning and the default values of `Vinl = 0.8V` and `Vinh = 2.0V` are assumed.
I/O	IbisIO	
IO	IbisIO	
IO_open_drain	IbisIO_OpenPullUp	
IO_open_sink	IbisIO_OpenPullUp	
IO_open_source	IbisIO_OpenPullDown	
IO_OpenPullDown	IbisIO_OpenPullDown	
IO_OpenPullUp	IbisIO_OpenPullUp	

IBIS-SigNoise Model Type Mapping and Descriptions

IBIS Model_Type	SigNoise ModelType	Description
Input_ECL IO_ECL Output_ECL	IbisInput IbisIO IbisOutput	These model types specify that the model represents an ECL type logic that follows different conventions for the PullDown keyword. These model types (except Output_ECL) must have `Vinl` and `Vinh` defined. Otherwise, the parser issues a warning and the default values of `Vinl= -1.475V` and `Vinh = -1.165V` are assumed.
Terminator	IbisTerminator	This is an input-only model that can have analog loading effects on the circuit being simulated but has no digital logic thresholds. Examples of terminators are capacitors, termination diodes, and pullup resistors.
Output	IbisOutput	This model type indicates that an output always *sources* or *sinks* current and cannot be disabled.
3-state Tristate	IbisOutput IbisOutput	This model type indicates that an output be disabled (that is put into a high impedance state).
OpenDrain Open_drain Open_sink Output_OpenPullDown Output_Open PullUp	IbisOutput_OpenPullUp IbisOutput_OpenPullUp IbisOutput_OpenPullUp IbisOutput_OpenPullDown IbisOutput_OpenPullUp	These model types indicate that the output has an *open* side (do not use PullUp parameter or if it is used, `set I = 0mA` for all voltages specified) and the output *sinks* current. The IBIS Open_drain model type is retained for backward compatibility.

IBIS-SigNoise Model Type Mapping and Descriptions

IBIS Model_Type	SigNoise ModelType	Description
Open_source	IbisOutput_OpenPullDown	This model type indicates that the output has *open* side (do not use PullDown parameter or if it is used, `set I = 0mA` for all voltages specified) and the output *sources* current.

3. Ramp

   `Ramp` is a required parameter for `IbisIOCell` except for inputs and terminators model types. It 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` sub-parameter, which is described above. `Ramp` allows specification of how long it takes to change the gate voltages for the totem pole transistors to turn one *on* and turn the other one *off* during a transition. This information is only used when no TVCurve data is present. Ramp values are categorized by slew rate, so `minimum` is the slowest slew rate and so on. The IBIS Keyword for `Ramp` is `[Ramp]`. Here's an example of `Ramp`:

```
(IbisIOCell
  ("RampIOExample"
    (Ramp
      (Rise                                ; See Rise below
        (minimum                           ; Minimum slew rate
          (dV "3")                         ; See dV below
          (dt "0.9n") )                    ; See dt below
        (typical                           ; Typical slew rate
          (dV "3")
          (dt "0.6n") )
        (maximum                           ; Maximum slew rate
          (dV "3")
          (dt "0.48n") ) )
      (Fall                                ; See Fall below
        (minimum
          (dV "3")
          (dt "0.9n") )
        (typical
          (dV "3")
          (dt "0.6n") )
        (maximum
          (dV "3")
          (dt "0.48n") ) ) )
    ...
```

   a. Rise

`Rise` is a required sub-parameter for `Ramp`. The `Rise` time is defined as the time it takes the output to go from 20% to 80% of its final value. You can specify The `minimum`, `typical`, and `maximum` values for `Rise` and you can also specify the `dV` and `dt` values for each of them in order to calculate the slew rate. The `minimum` value should have the slowest slew rate or, in other words, have the lowest value for `dV` and `dt` (`Ramp` rate). The opposite is true for `maximum`, and `typical` has the intermediate value of the slew rate. The IBIS keyword for `Rise` is `dV/dt_r`.

dV

`dV` is a required sub-parameter for `Rise` and `Fall`. The value of `dV` must represent the difference between 20% and 80% of actual voltage swing. Note that you must specify `dV` before `dt`.

dt

`dt` is a required sub-parameter for `Rise` and `Fall`. The `dt` values must correspond to the actual time to cover the 20% to 80% voltage range of `dV`. Note that you must specify `dt` after `dV`.

b. Fall

`Fall` is a required sub-parameter for `Ramp`. The `Fall` time is defined as the time it takes the output to go from 20% to 80% of its final value. You can specify the `minimum`, `typical`, and `maximum` values for `Rise` and you can also specify the `dV` and `dt` values for each of them in order to calculate the slew rate. The `minimum` value should have the slowest slew rate or, in other words, have the lowest value for `dV` and `dt`. The opposite is true for `maximum`, and `typical` has the intermediate value of the slew rate. The IBIS keyword for `Fall` is `dV/dt_f`. The `dV` and `dt` sub-parameters of `Fall` are defined in the same way as they are for `Rise`. Refer to the definitons above for more on `dV` and `dt`.

4. LogicThresholds

`LogicThresholds` is a required parameter for `IbisIOCell`. It is used for noise margin and delay calculations. Here's an example of `LogicThresholds`:

```
(IbisIOCell
  ("LogicTExample"
    (LogicThresholds
      (Input                        ; See Input below
        (High                       ; See High below
          (minimum 3.0)             ; Minimum Input High value
          (typical 3.0)             ; Typical Input High value
          (maximum 3.0) )           ; Maximum Input High value
        (Low                        ; See Low below
          (minimum 2.0)
          (typical 2.0)
          (maximum 2.0) ) )
```

```
    (Output                        ; See Output below
      (High
        (typical 4.5) )
      (Low
        (typical 0.1) ) )
    (ThermalShift                  ; See ThermalShift below
      (High
        (typical 0.0015) )
      (Low
        (typical 0.0006) ) ) )
...
```

a. Input

`Input` is a required sub-parameter for `LogicThresholds`. It is used to define the high and low voltage values for the input of a buffer.

High

`High` is a required sub-parameter for `Input`. It specifies the `minimum`, `typical`, and `maximum` input high voltage values of a buffer. Only the `typical` value is required for `High`. You can specify the `minimum` and `maximum` values, but they are optional. The IBIS keyword for `High` is `vinh`.

Low

`Low` is a required sub-parameter for `Input`. It specifies the `minimum`, `typical`, and `maximum` input low voltage values of a buffer. As with `High`, only the `typical` value is required for `Low`. You can specify the `minimum` and `maximum` values, but they are optional. The IBIS keyword for `Low` is `vinl`.

b. Output

`Output` is a required sub-parameter for `LogicThresholds`. It is used to define the high and low voltage values for the output of a buffer. The `High` and `Low` values for `Output` are similarly defined as in the `Input` keyword. The only difference is that there are no corresponding IBIS keywords for `High` and `Low` under the Output keyword. In other words, the IBIS keywords, `vinh` and `vinl` only correspond to `High` and `Low`, respectively, under the keyword `Input`.

c. ThermalShift

`ThermalShift` is an optional sub-parameter for `LogicThresholds`. It does not have a corresponding IBIS keyword. It is used to adjust the `LogicThresholds Input` values based on actual die temperature.

```
Vhi-adj = Vhi + (Tempco * (comp-junc-temp - VIReferenceTemperature) )
```

The units of this formula are `mV/degC`.

5. DelayMeasurementFixture

   `DelayMeasurementFixture` is a required parameter for `IbisIOCell`. It similar to the `DelayMeasurementFixture` sub-parameter that is defined in IbisIOCell Parameters and Sub-parameters Descriptions on page 95. Refer to that section to learn more about this parameter and its usage.

6. VIReferenceTemperature

   `VIReferenceTemperature` is an optional parameter for `IbisIOCell`. It defines the temperature range over which the model is to operate. When the VI-curves are used to represent temperature variations, the nominal temperature for each curve may be set using `VIReferenceTemperature`. In the absence of this parameter, default values are assumed. In the example below, the `minimum` temperature is 10 degrees Centigrade. The `minimum` VI curve is assumed to hold for this temperature. Linear interpolations are used for temperatures between 10 and 50 degrees Centigrade. Likewise for the `maximum` temperature. The default values are 0, 50, and 100. The IBIS keyword for `VIReferenceTemperature` is `[Temperature Range]`.

```
(IbisIOCell
  (“TempExample”
    (VIReferenceTemperature
      (minimum “10”)
      (typical “50”)
      (maximum “100”) )
...
```

7. PullUp

   `PullUp` is an optional parameter for `IbisIOCell`. `PullUp` defines the I-V tables of the pullup structures of an output buffer. Here’s an example of `PullUp`:

```
(IbisIOCell
  (“PUExample”
    (PullUp
      (ReferenceVoltage                  ; See ReferenceVoltage below
        (minimum “5”)
        (typical “5”)
        (maximum “5”) )
      (VICurve “                         ; See VICurve below
        10   -142ma   -137ma   -147ma
        5   -137ma   -132ma   -142ma
        ...
        -4    126ma    121ma    131ma
        -5    137ma    132ma    142ma”)
      (ImpedanceClass                    ; See ImpedanceClass below
        (High minimum)
        (Typical typical)
        (Low maximum) ) )
      ...
```

a. ReferenceVoltage

`ReferenceVoltage` is a required sub-parameter for `PullUp`. It defines a voltage rail as the reference voltage for the PullUp I-V data. The IBIS keyword for `ReferenceVoltage` is `[Pullup Reference]`.

b. VICurve

`VICurve` is a required sub-parameter for `PullUp`. `VICurve` is the table of data giving V/I curves for voltage, typical current, minimum current, and maximum current (correct order is necessary) in sets of four. The data must be ordered from low to high voltage. The suffixes are standard Spice suffixes. Currents are positive into the device. Voltage in this table is `ReferenceVoltage - Actual`. The IBIS keyword of `VICurve` is `[Pullup]`.

c. ImpedanceClass

`ImpedanceClass` is an optional sub-parameter for `PullUp`. `ImpedanceClass` is automatically calculated for every V-I curve if it is not explicitly provided. You can override the automatic calculation by changing the values stored in the library. The Fast/Typical/Slow feature uses `ImpedanceClass` to choose the High, Typical, and Low impedance curves. In the example above, the High impedance curve is the IBIS-minimum curve (the second current column), and the Low impedance curve is the IBIS-maximum (last column). The automatic calculation is done simply by adding the absolute values of the currents in each column. The column with the lowest total current is assigned to the High `ImpedanceClass`, and so on. The `ImpedanceClass` sub-parameter can also be defined under the `PullDown`, `GroundClamp`, and `PowerClamp` parameters.

8. PullDown

`PullDown` is an optional parameter for `IbisIOCell`. `PullDown` defines the I-V tables of the pulldown structures of an output buffer. Here's an example of `PullDown`:

```
(IbisIOCell
  ("PDExample"
    (PullDown
      (ReferenceVoltage                    ; See ReferenceVoltage below
        (minimum "0")
        (typical "0")
        (maximum "0"))
      (VICurve "                           ; See VICurve below
        -5.0 -215ma   -210ma   -225ma
        -4.0 -212.0ma -207.0ma -217.0ma
        ...
        5.0  215ma    210ma    220ma
        10.0  220ma    215ma    225ma ") )
      (ImpedanceClass                      ; See ImpedanceClass below
        (High minimum)
```

```
    (Typical typical)
    (Low maximum) ) )
  ...
```

a. ReferenceVoltage

`ReferenceVoltage` is a required sub-parameter for `PullDown`. It defines the power supply rail other than `0V` as the reference voltage for the PullDown I-V data. The IBIS keyword for `ReferenceVoltage` is `[Pulldown Reference]`.

b. VICurve

`VICurve` is a required sub-parameter for `PullDown`. `VICurve` is the table of data giving V/I curves for voltage, typical current, minimum current, and maximum current (correct ordering is required) in sets of four. The data must be ordered from low to high voltage. The suffixes are standard Spice suffixes. Currents are positive into the device. Voltage in this table is `ReferenceVoltage – Actual`. The IBIS keyword of `VICurve` is `[PullDown]`.

c. ImpedanceClass

`ImpedanceClass` is an optional sub-parameter for `PullDown`. It is defined the same way as it is in the `PullUp` section above.

9. PowerClamp

`PowerClamp` is an optional parameter for `IbisIOCell`. `PowerClamp` defines the I-V tables of the clamping diodes connected to the power pins. Here's an example of `PowerClamp`:

```
(IbisIOCell
  ("PWRCExample"
    (PowerClamp
      (ReferenceVoltage            ; See ReferenceVoltage below
        (minimum "5")
        (typical "5")
        (maximum "5"))
      (VICurve                     ; See VICurve below
        "0.0 0 0 0
        -0.1 0 0 0
        ...
        -1.0 25ma 25ma 25ma
        -5.0 293ma 293ma 293ma")
      (ImpedanceClass              ; See ImpedanceClass below
        (High minimum)
        (Typical typical)
        (Low maximum) )
      (TTpower                     ; See TTpower below
        (minimum "9n")
        (typical "10n")
        (maximum "12n") ) )
```

...

a. ReferenceVoltage

`ReferenceVoltage` is a required sub-parameter for `PowerClamp`. It defines a voltage rail as the reference voltage for the `PowerClamp` I-V data. The IBIS keyword for `ReferenceVoltage` is `[POWER Clamp Reference]`.

b. VICurve

`VICurve` is a required sub-parameter for `PowerClamp`. `VICurve` is the table of data giving V/I curves for voltage, typical current, minimum current, and maximum current (correct order is necessary) in sets of four. The data MUST be ordered from low to high voltage. The suffixes are standard Spice suffixes. Currents are positive into the device. Voltage in this table is `ReferenceVoltage - Actual` The IBIS keyword of `VICurve` is `[POWER Clamp]`.

c. ImpedanceClass

`ImpedanceClass` is an optional sub-parameter for `PowerClamp`. It is defined the same way as it is in the `PullUp` section above.

d. TTpower

`TTpower` is an optional sub-parameter for `PowerClamp`. `TTpower` is used to specify the transit time parameters to estimate the transit time capacitances for `PowerClamp`. Only the `typical` value is required for `TTpower`. You can specify the `minimum` and `maximum` values but they are optional. The IBIS keyword for `TTpower` is `[TTpower]`.

10. GroundClamp

`GroundClamp` is an optional parameter for `IbisIOCell`. `GroundClamp` defines the I-V tables of the clamping diodes connected to the ground pins. Here's an example of `GroundClamp`:

```
(IbisIOCell
  ("GNDCExample"
    (GroundClamp
      (ReferenceVoltage              ; See ReferenceVoltage below
        (minimum "0")
        (typical "0")
        (maximum "0"))
      (VICurve                       ; See VICurve below
        "0 0 0 0
        -0.1 0 0 0
        ...
        -1.0 -25ma -25ma -25ma
        -5.0 -293ma -293ma -293ma")
      (ImpedanceClass                ; See ImpedanceClass below
```

```
    (High minimum)
    (Typical typical)
    (Low maximum) )
  (TTgnd                          ; See TTgnd below
    (minimum 9n)
    (typical 10n)
    (maximum 12n) ) )
  ...
```

a. ReferenceVoltage

`ReferenceVoltage` is a required sub-parameter for `GroundClamp`. It defines the power supply rail other than `0V` as the reference voltage for the GroundClamp I-V data. The IBIS keyword for `ReferenceVoltage` is `[GND Clamp Reference]`.

b. VICurve

`VICurve` is a required sub-parameter for `GroundClamp`. `VICurve` is the table of data giving V/I curves for voltage, typical current, minimum current, and maximum current (correct order is necessary) in sets of four. The data must be ordered from low to high voltage. The suffixes are standard Spice suffixes. Currents are positive into the device. Voltage in this table is `ReferenceVoltage - Actual`. The IBIS keyword of `VICurve` is `[GND Clamp]`.

c. ImpedanceClass

`ImpedanceClass` is an optional sub-parameter for `GroundClamp`. It is defined the same way as it is in the `PullUp` section above.

d. TTgnd

`TTgnd` is an optional sub-parameter for `GroundClamp`. `TTgnd` is used to specify the transit time parameters that are used to estimate the transit time capacitances for `GroundClamp`. Only the `typical` value is required for `TTgnd`. You can specify the `minimum` and `maximum` values, but they are optional. The IBIS keyword of `TTgnd` is `[TTgnd]`.

11. Terminators

`Terminators` is an optional parameter for `IbisIOCell`. It is used to define shunt terminators in DML. SigNoise supports it, although Allegro SI provides a better way of specifying terminators. Following are two examples of `Terminators`:

Example 1:

```
(IbisIOCell
  ("CDSDefaultTerminator"
    (Model
      (ModelType "Terminator") )
  (Terminators
```

```
    (power_resistor                  ; See power resistor below
      (typical "330") )
    (ground_resistor                 ; See ground resistor below
      (typical "220") ) ) )
    ...
```

Example 2:

```
(IbisIOCell
  ("CDSDefaultACTerminator"
    (Model
      (ModelType "Terminator") )
  (Terminators
    (ac_resistor                     ; See ac_resistor below
      (typical "50") )
    (ac_capacitor                    ; See ac_capacitor below
      (typical "50p") ) )
    ...
```

Note: For Example 2, the `power_resistor` and `ground_resistor` keywords should be absent since it is using an AC Terminator instead of the standard Terminator model.

a. power_resistor

`power_resistor` is a required sub-parameter for `Terminators`. The `power_resistor` sub-parameter defines the resistance value that is connected to the power pins. The `typical` values are required but the `minimum` and `maximum` values are optional, and cannot be supplied by an IBIS file. The IBIS keyword for `power_resistor` is `Rpower`.

b. ground_resistor

`ground_resistor` is a required sub-parameter for `Terminators`. The `ground_resistor` sub-parameter defines the resistance value that is connected to the ground pins. The `typical` values are required but the `minimum` and `maximum` values are optional, and cannot be supplied by an IBIS file. The IBIS keyword for `ground_resistor` is `Rground`.

c. ac_resistor

`ac_resistor` is a required sub-parameter for `Terminators` for an AC Terminator model. The `ac_resistor` sub-parameter is used to define the resistance value for an AC Terminator. The `typical` values are required but the `minimum` and `maximum` values are optional, and cannot be supplied by an IBIS file. The IBIS keyword for `ac_resistor` is `Rac`.

d. ac_capacitor

`ac_capacitor` is a required sub-parameter for `Terminators` for an AC Terminator model. The `ac_capacitor` sub-parameter is used to define the

capacitance value for an AC Terminator. The `typical` values are required but the `minimum` and `maximum` values are optional, and cannot be supplied by an IBIS file. The IBIS keyword for `ac_capacitor` is `Cac`.

12. RisingWaveform

    `RisingWaveform` is an optional parameter for `IbisIOCell`. It describes the shape of the rising edge waveform of a driver. It has three sub-parameters as described below. Here's an example of `RisingWaveform`:

```
(IbisIOCell
  (RisingWaveform
    ("risew1"                          ; waveform model name
      (TestFixture                     ; See TestFixture below
        (R 25)                         ; See R below
        (V                             ; See V below
          (minimum 0)
          (typical 0)
          (maximum 0) ) )
        (SlewRateClass                 ; See SlewRateClass below
          (Low minimum)
          (Typical typical)
          (High maximum) )
                                       ; See TVCurve below for more info
          (TVCurve "
          0           0          0          0
          1.32e-10    0.06964    0.06964    0.06964
          1.7116e-09  0.207593   0.207593   0.207593
          1.9558e-09  0.301442   0.301442   0.301442
          2.2e-09     0.331619   0.331619   0.331619
          2.3e-09     0.331619   0.331619   0.331619") ) )
          ...
)
```

    a. TestFixture

    `TestFixture` is a required sub-parameter for `RisingWaveform`. It is used to describe the loading conditions under which the waveform is taken. Its sub-parameters are described next.

    R

    `R` is a required sub-parameter for `TestFixture`. It describes the load resistance under which the waveform is taken.

    V

    `V` is a required sub-parameter for `TestFixture`. It describes the load voltage under which the waveform is taken.

    b. SlewRateClass

`SlewRateClass` is a required sub-parameter for `RisingWaveform`. It is used to define the `low`, `typical`, and `high` slew rates for `RisingWaveform`.

c. TVCurve

`TVCurve` is a required sub-parameter for `RisingWaveform`. It is the table of data giving T/V curves for temperature, typical voltage, minimum voltage, and maximum voltage (correct order is necessary) in sets of four. The data must be ordered from low to high temperature. The suffixes are standard Spice suffixes. Voltages are positive into the device. The temperature in this table is `ReferenceTemperature - Actual`.

13. FallingWaveform

`FallingWaveform` is an optional parameter for `IbisIOCell`. It describes the shape of the falling edge waveform of a driver. It has three sub-parameters as described below. Here's an example of `FallingWaveform`:

```
(IbisIOCell
  (FallingWaveform
    ("fallw1"                          ; waveform model name
      (TestFixture                     ; See TestFixture below
        (R 25)                         ; See R below
        (V                             ; See V below
          (minimum 1.5)
          (typical 1.5)
          (maximum 1.5) )
        (SlewRateClass                 ; See SlewRateClass below
          (Low minimum)
          (Typical typical)
          (High maximum) )
                                       ; See TVCurve below for more info
        (TVCurve "
          0           1.5        1.5        1.5
          2.139e-10   1.49684    1.49684    1.49684
          2.325e-10   1.49053    1.49053    1.49053
          5.456e-10   1.10559    1.10559    1.10559
          7.285e-10   0.958337   0.958337   0.958337
          9.114e-10   0.795313   0.795313   0.795313
          1.24e-09    0.605994   0.605994   0.605994
          1.643e-09   0.542888   0.542888   0.542888
          1.8755e-09  0.511334   0.511334   0.511334
          2.3715e-09  0.478729   0.478729   0.478729
          2.9171e-09  0.458746   0.458746   0.458746
          3.1e-09     0.448228   0.448228   0.448228
          3.2e-09     0.448228   0.448228   0.448228") ) )
...
```

a. TestFixture

`TestFixture` is a required sub-parameter for `FallingWaveform`. It is used to describe the loading conditions under which the waveform is taken. Its sub-parameters are described next.

R

`R` is a required sub-parameter of `TestFixture`. It describes the load resistance under which the waveform is taken.

V

`V` is a required sub-parameter of `TestFixture`. It describes the load voltage under which the waveform is taken.

b. SlewRateClass

`SlewRateClass` is a required sub-parameter for `FallingWaveform`. It is used to define the `low`, `typical`, and `high` slew rates for `FallingWaveform`.

c. TVCurve

`TVCurve` is a required sub-parameter for `FallingWaveform`. It is the table of data giving T/V curves for temperature, typical voltage, minimum voltage, and maximum voltage (correct order is necessary) in sets of four. The data must be ordered from low to high temperature. The suffixes are standard Spice suffixes. Voltages are positive into the device. The temperature in this table is `ReferenceTemperature - Actual`.

14. MacroModel

Refer to Chapter 3, “DML Macromodeling,” for more information on how to use the MacroModel parameter.

How to Create an IbisIOCell

This section describes the different ways you can create an IbisIOCell model, namely, using the model translation utilities such as `ibis2signoise` and `quad2signoise`, using the GUI of Allegro SI (such as using clone selection and Model Browser), and by manually writing DML files. The different methods of creating an IbisIOCell model are described here in decreasing order of suggested method, and common practices of model creation, from first to last.

Model Translation Utilities (Ibis2Signoise & Quad2Signoise)

You can translate models that are written in other modeling languages, such as Ibis and Quad, into DML using the model translation tools called Ibis2Signoise and Quad2Signoise respectively. Refer to Chapter 8, "Model Translation," to learn how to do this. You can also convert a model that is in Spice format into the Ibis format first before using Ibis2Signoise to convert it into DML format.

Clone Selection

You can use the CloneSelection option to create IbisIOCell models for new devices, which have similar properties as other devices that already have IbisIOCell models in the existing DML Library. Follow the steps below to create an IbisIOCell model using this method:

1. Open the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library…*

   The Signal Analysis Library Browser window appears.

2. Click the *Browse Models* button at the top left hand corner of the Signal Analysis Library Browser window.

   The Model Browser window appears.

3. Set the model filters on the Model Browser window to look for specific models to clone. You can do this by using the pull down menus next to *Show Models From* (*All Libraries*, *Selected Device Library*, etc.) and *Model Type Filter* (*Any*, *IbisIOCell*, etc.) located at the top of the Model Browser window.

4. Select the desired model from the Model Browser window after filtering the models and click on the *Add Model* button and then select *CloneSelection* from the pull-down menu.

   A menu appears asking for the new IbisIOCell model name.

5. Enter the new IbisIOCell model name and click *OK*.

   The new device model appears in the Model Browser window.

6. Select this new device model and click *Edit* to modify this new model to fit the intended specifications of the new model.

Model Browser Form

You can also create an IbisIOCell model by using the Create IbisIOCell Model form. When the Create IbisIOCell Model window is initially invoked, it will be an empty form, which you need to enter the values for based on the model that needs to be created. The same form is used for the different IbisIOCell models and, depending on which values are entered in the form, for the different sections (General, Input, Output and DelayMeasurement). Note that not all the sections need to be completed in order to create an IbisIOCell model. For example, an Ibis Input type model only needs the Input tab to be completed in order to create the model. Here's how the IbisIOCell model can be created using the Model Browser form:

1. Open the Create IbisIOCell Window from the Model Browser, which is accessed from the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library… – Browse Models – Add Model – IbisDevice*

2. Enter all the required and optional values for the IbisIOCell depending on the type of IbisIOCell (See IBIS-SigNoise Model Type Mapping and Descriptions table on page 99) that needs to be created.

3. Click *OK* at the bottom of this window to create the model.

Figure 2-8 General Section Tab of the IOCell Editor Window From Allegro SI

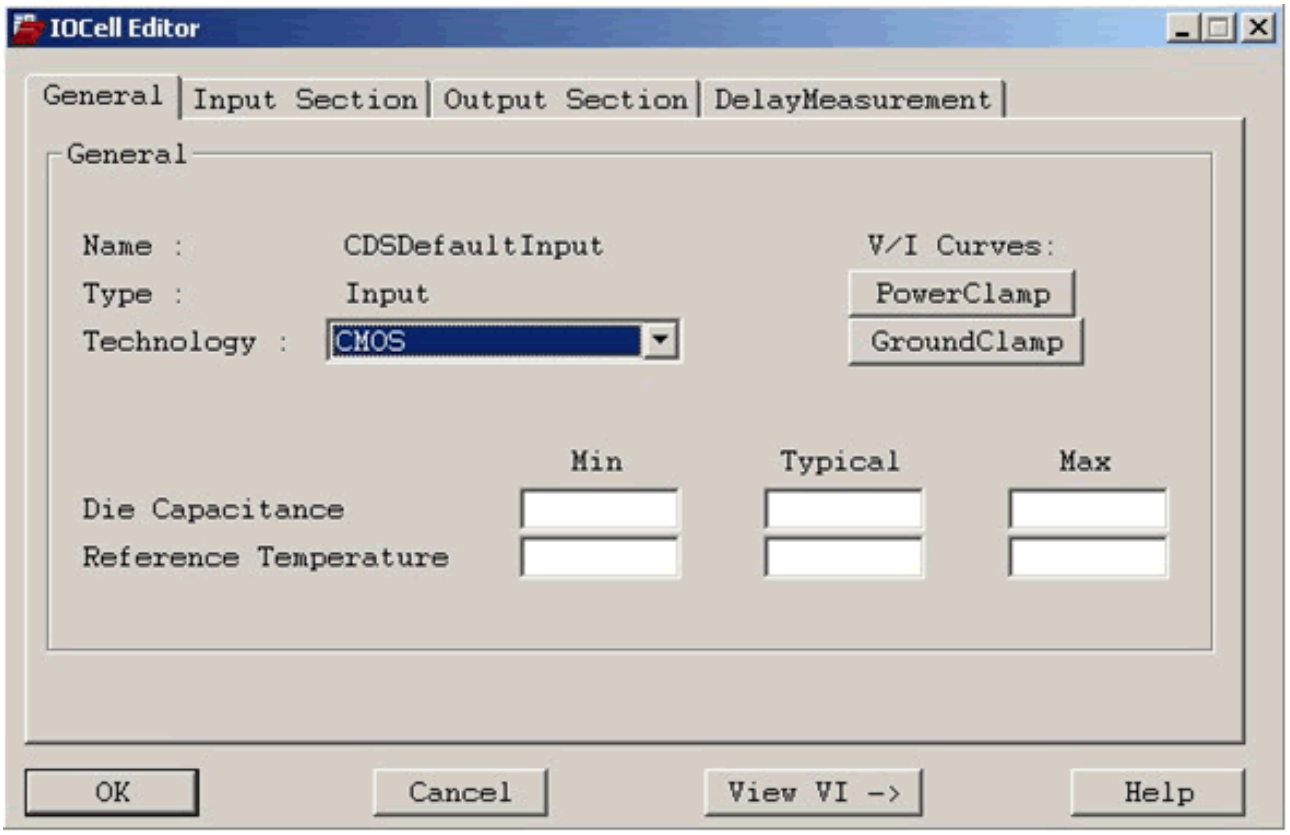


Figure 2-9 Input Section Tab of the IOCell Editor Window from Allegro SI

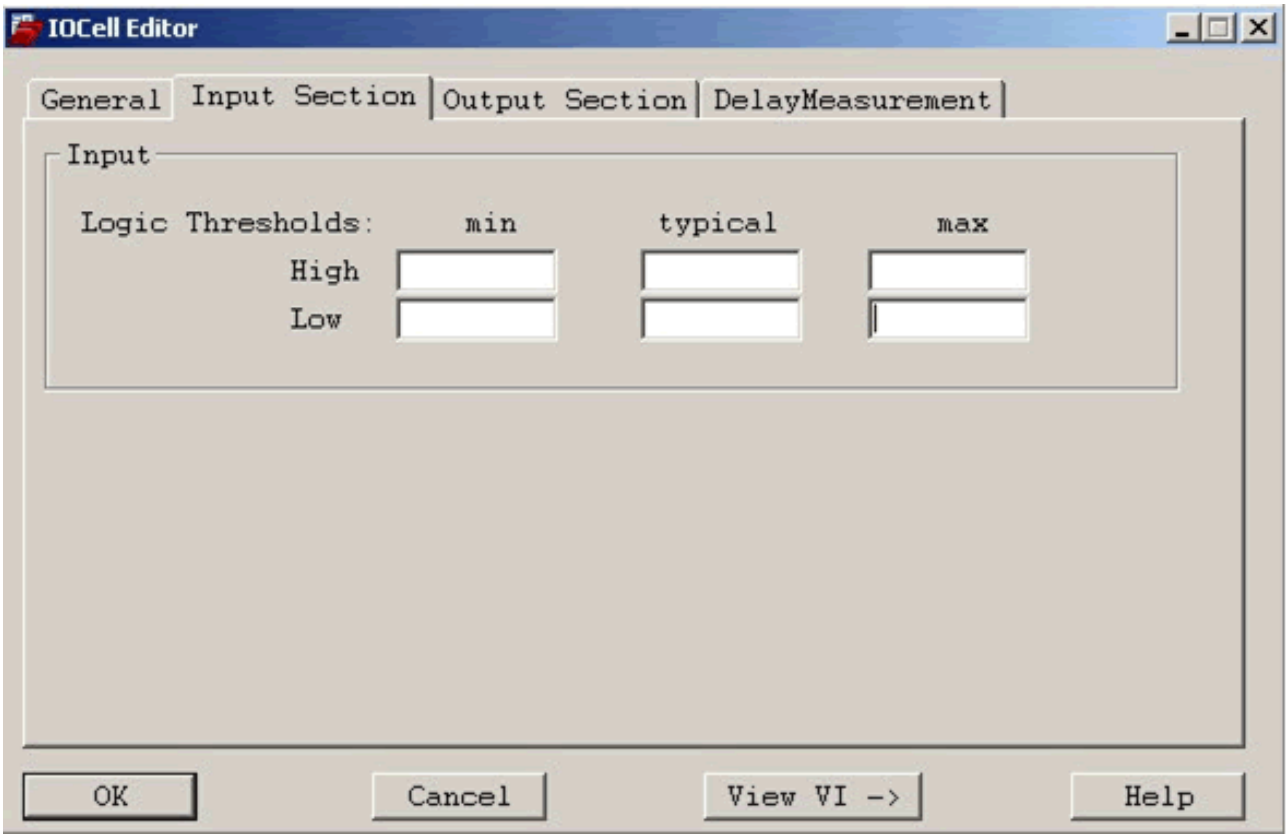


Figure 2-10 Output Section Tab of the IOCell Editor Window from Allegro SI

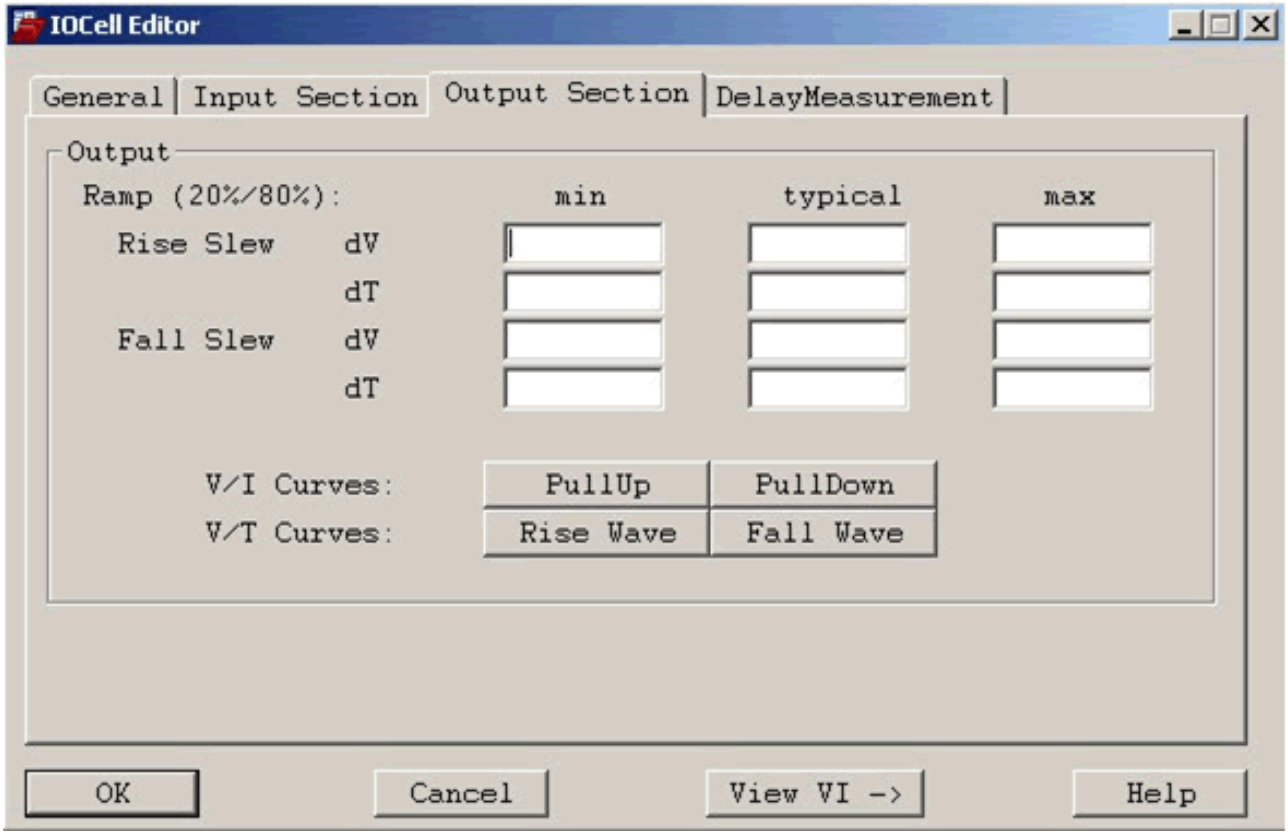
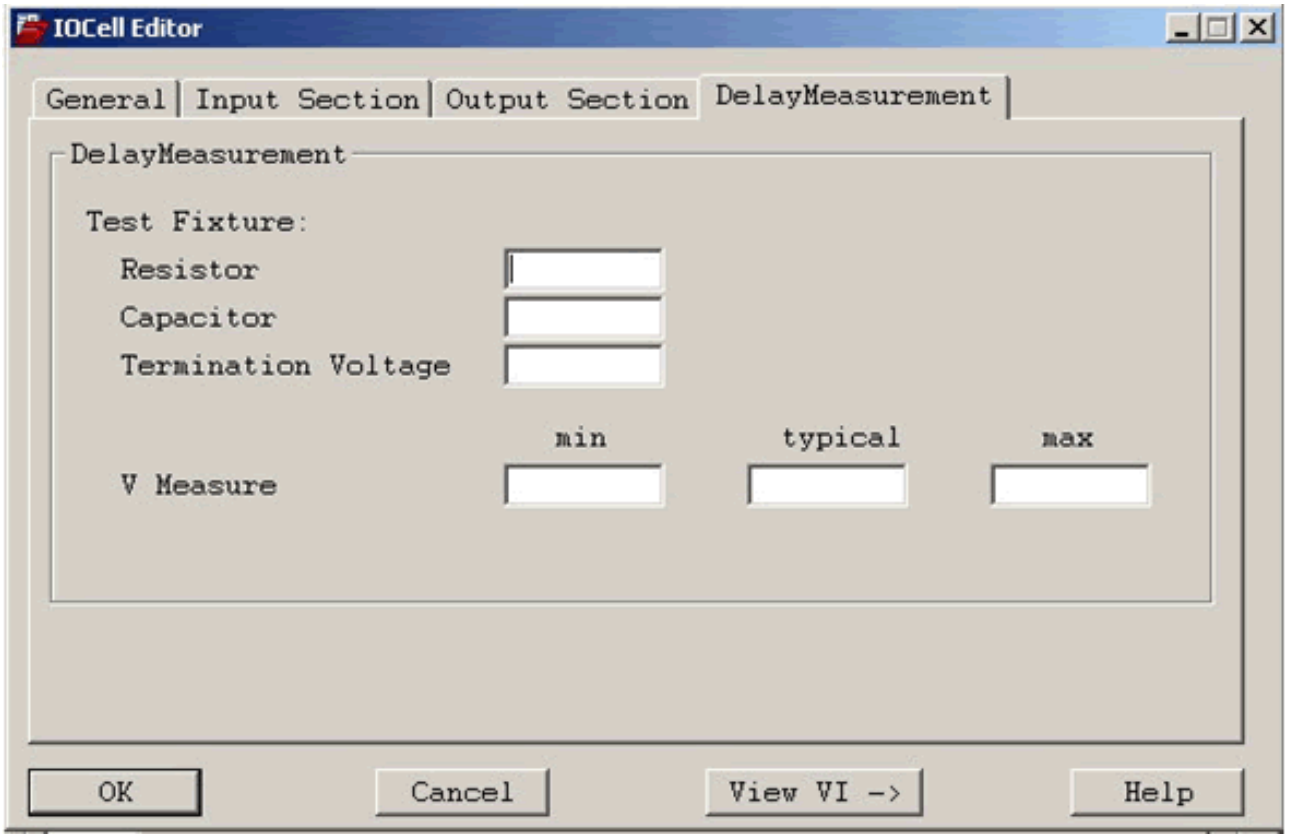


Figure 2-11 DelayMeasurement Section Tab of the IOCell Editor Window from Allegro SI



Manual creation of an IbisIOCell model

You can create an IbisIOCell model by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token on page 19. Listed below are the steps to manually create an IbisIOCell model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Type in the necessary keywords, parameters, and sub-parameters as needed for the new model. Refer to the specific parameters and sub-parameters descriptions, which are listed in IbisIOCell Parameters and Sub-parameters Descriptions on page 95, for more details on their usage.
3. Save the file and run `dmlcheck` to check for syntax errors.

Note: It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`PackagedDevice`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

AnalogOutput

`AnalogOutput` allows the user to store analog waveforms and manipulate them to represent a driver in ways similar to the way an IbisIOCell is used. You may specify a rising, falling, pulse, or inverted-pulse simulation. The corresponding waveform for the selected AnalogOutput model is used. There are seven main parameters under `AnalogOutput`, namely, `TheveninCircuit`, `Rise`, `Fall`, `Pulse`, `InvertedPulse`, `GroundVoltage`, and `PowerVoltage`.

Analog Output Required Parameters and Their Respective Required and Optional Sub_Parameters

Required Parameters	Sub-parameters	
	Required	**Optional**
TheveninCircuit	.subckt	-
Rise	IDL	WaveFile
Fall	IDL	WaveFile
Pulse	IDL	WaveFile
InvertedPulse	IDL	WaveFile

AnalogOutput Optional Parameters and Their Respective Required and Optional Sub-parameters

Optional Parameters	Sub-parameters	
GroundVoltage	*<voltage_value>*	-

AnalogOutput Optional Parameters and Their Respective Required and Optional Sub-parameters

Optional Parameters	Sub-parameters	
	Required	**Optional**
PowerVoltage	*<voltage_value>*	-

AnalogOutput Parameter and Sub-parameter Descriptions

This section contains descriptions and examples of AnalogOutput parameters and sub-parameters. Some of the examples that are used in this sub-section are excerpts from a larger DML file called `cds_samples.dml` which is located in the *<cds_install_dir>*`/share/pcb/signal/` directory.

1. ThéveninCircuit

   `TheveninCircuit` is a required parameter for `AnalogOutput`. It is a four-terminal model similar to the five-terminal IOCell models. Only the enable pin is missing. The terminals are in the same order in both cases.

   The conventional name for the Thevenin circuit is the name for the AnalogOutput with `_Thevenin` appended. The sub-circuit name must be unique within each simulation.

   The `TheveninCircuit` assigned to each `AnalogOutput` gives the user the flexibility to put realistic impedances on the analog source, and thus much more accurately model the phenomena of interest. If a `TheveninCircuit` is not specified, it defaults to a single resistor of `1 mOhm` connecting the input and output pins.

   Here's an example of `TheveninCircuit`:

```
(AnalogOutput
  ("ThevCExample"
   (TheveninCircuit
      ".subckt CDS_AO_1_Thevenin power output ground input
      R1 output input 1e-3
      .ends CDS_AO_1_Thevenin ")
      ...
```

2. Rise, Fall, Pulse, InvertedPulse

   `Rise`, `Fall`, `Pulse`, and `InvertedPulse` are required parameters for `AnalogOutput`. They are the types of stimulus that you can define for an AnalogOutput model. However, only one of these keywords are required depending on the type of stimulus that is desired. For example, you can specify `Rise` by itself or with `Pulse` (or

Fall or InvertedPulse) in an AnalogOutput mode. The following example shows how you can define the Rise parameter . Note that this definition is similar for Fall, Pulse, and InvertedPulse with the exception of the parameter Rise being replaced by the appropriate parameter and therefore examples of these other parameters are not shown here.

```
(AnalogOutput
  (“RiseExample”
    (Rise
      (IDL
        “.subckt CDS_AO_1_rise vout vret
        V1 vout vret PWL(
        + 0 0
        + 2e-9 4.5
        + 3e-9 5.0)
        .ends CDS_AO_1_rise ”)
      (WaveFile “/home/auser/wave/osc/trace1”) )
      ...
```

a. IDL

IDL is a required sub-parameter for Rise, Fall, Pulse, and InvertedPulse. It is used to define a sub-circuit (PWL for example) for the specific stimulus.

b. WaveFile

WaveFile is an optional sub-parameter for Rise, Fall, Pulse, and InvertedPulse. It is used to define the corresponding wave file (including its path) for the sub-circuit defined under the IDL sub-parameter.

3. GroundVoltage

GroundVoltage is an optional parameter of AnalogOutput. It used to describe the ground level voltage for an AnalogOutput model. If GroundVoltage is not specified, the ground voltage is defaulted to 0 Volts. Here’s an example of GroundVoltage:

```
(AnalogOutput
  (“GVExample”
    (GroundVoltage 1.2)
  ...
```

4. PowerVoltage

PowerVoltage is an optional parameter of AnalogOutput. It used to describe the High voltage for an AnalogOutput model. If PowerVoltage is not specified, the high voltage is defaulted to 5 Volts. Here’s an example of PowerVoltage:

```
(AnalogOutput
  (“PVExample”
    (PowerVoltage 4.3)
  ...
```

How to Create an AnalogOutput model

This section describes the different ways you can create and edit an AnalogOutput model, namely, using the GUI of Allegro SI (such as using the AnalogOutput Model Editor Form and Clone Selection) and by manually writing DML files. The different methods of creating an AnalogOutput model are described here in decreasing order of suggested method, and common practices of model creation, from first to last.

AnalogOutput Model Editor Form of Allegro SI

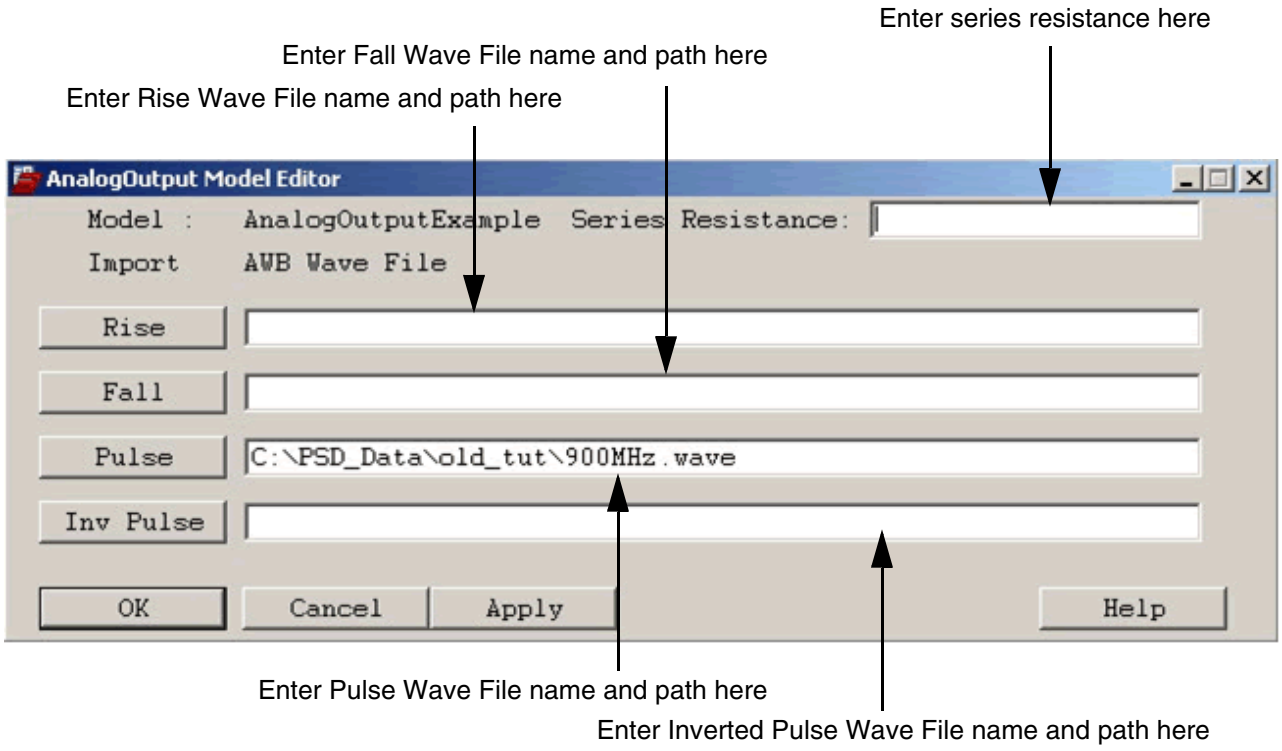
Figure 2-12 shows the AnalogOutput Model Editor window from Allegro SI. See Appendix D, "DML File of GUI Example (Figure 2-9)," for the ASCII version of this model example. Referring to Figure 2-12 as a template, an AnalogOutput model can be created as follows:

1. Open the AnalogOutput Model Editor window from the Model Browser, which is accessed from the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library… – Browse Models – Add Model – AnalogOutput*

2. Enter all the values for the AnalogOutput that is being modeled as shown in Figure 2-12
3. Click *OK* at the bottom of this window to create the model.

Figure 2-12 AnalogOutput Model Editor Window of Allegro SI



Clone Selection

You can use the *CloneSelection* option to create AnalogOutput models for new devices, which have similar properties as other devices that already have AnalogOutput models in the existing DML Library. The following steps need to be taken in order to create an AnalogOutput model using this method:

1. Open the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library…*

   The Signal Analysis Library Browser window appears.

2. Click the *Browse Models* button, which is located at the top left hand corner of the Signal Analysis Library Browser window.

   The Model Browser window appears.

3. Set the model filters on the Model Browser window to look for specific models to be cloned. To do this, you can use the pull down menus next to *Show Models From* (*All*

Libraries, *Selected Device Library*, etc.) and *Model Type Filter* (*Any*, *AnalogOutput*, etc.) located at the top of the Model Browser window.

4. Select the desired model from the Model Browser window after filtering the models and click the *Add Model* button and then select *CloneSelection* from the pull-down menu.

   A pop-up menu appears asking for the new AnalogOutput model name.

5. Enter the new AnalogOutput model name

6. Click *OK*.

   The new device model appears in the Model Browser window.

7. Select this new device model and click *Edit* to modify this new model to fit the intended specifications of the new device.

Manual creation of an AnalogOutput model

You can create an AnalogOutput model by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` file has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token on page 19. Listed below are the steps to create an AnalogOutput model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the AnalogOutput template from Appendix D, “AnalogOutput Template,” into this file.
3. Edit the DML file.
4. Save the file and run `dmlcheck` to check for syntax errors.

Note: AnalogOutput template:

- This template is a skeleton of the DML file, which you can copy and paste into a text editor, and then customize to model a specific device.
- You might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.
- Refer to the specific parameter and sub-parameter descriptions, which are listed in AnalogOutput Parameter and Sub-parameter Descriptions on page 120, for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was

originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`AnalogOutput`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

DesignLink

The `DesignLink` keyword is used to specify system-level connectivity such as multi-board or package-on-board scenarios. It also does not have an IBIS counter-part. Each `DesignLink` connects any number of boards through cable containing RLGC matrices to other boards. There are two main parameters under `DesignLink`, namely, `Connections` and `Drawings`. The following table shows the required parameters for `DesignLink` and their respective required and optional sub-parameters. `DesignLink` does not have any specific optional parameters other than the ones specified in Optional Keywords on page 17, such as `Manufacturer`.

Required Parameters for DesignLink and Their Respective Required and Optional Sub-Parameters

Required Parameters	Sub-parameters	
	Required	**Optional**
Connections	Length	RLGC
	PinMap	CircuitModels
	Cable	
Drawings	*<design_name>*	-

DesignLink Parameters and Sub-parameters Descriptions

This section contains descriptions and examples of DesignLink parameters and sub-parameters. Some of the examples that are used in this section are excerpts from a larger DML file called `cds_samples.dml` which is located in the *<cds_install_dir>*`/share/pcb/signal/` directory.

1. Connections

Connections is a required parameter for DesignLink. It introduces the section listing of each cable connection of a DesignLink model. There can be multiple numbers of cables that can be defined in a DesignLink model. These connections need not be actual cables, but conceptually it is convenient to think of them that way. Here's an example of Connections:

```
(DesignLink
  (BRD1_U1_to_BRD2_U2             ; symbolic name for this DesignLink model
                                  ; example. It is used in the GUI to
                                  ; activate the DesignLink.
    (Connections
      (cable_1                            ; Name of connection
        (Length 0.05)                     ; See Length below
        (PinMap                           ; See Pin Map below
          (1 "BRD1 U1 4" "BRD2 U2 4" )    ; Wire 1 connects BRD1_U1.4
          (2 "BRD1 U1 5" "BRD2 U2 5" )    ; to BRD2_U2.4 BRD1 and BRD1
          (3 "BRD1 U1 6" "BRD2 U2 6" )    ; and BRD 2  are user defined
          (4 "BRD1 U1 7" "BRD2 U2 7" ) )  ; symbolic names for the
                                          ; drawings (.brd or .mcm
                                          ; files) that make up the
                                          ; system.
      (Cable FourWireCable )              ; See Cable below
      ...
```

a. Length

Length is a required sub-parameter of Connections. It is used to define the length of a specific cable connection in unit Meters. If the length is zero, then the RLGC matrix should be interpreted as the lumped parasitics for some connecting device, instead of parasitics per unit length.

b. Pin Map

PinMap is a required sub-parameter of Connections. PinMap relates the wire number in Cable model, which in turn gives an index into the RLGC matrix to the pins of the connectors it joins.

c. Cable

Cable is a required sub-parameter of Connections. The value of the Cable sub-parameter points to a library entry in the Cable model type category of the library section. In other words, the value of Cable has to be the name of the intended cable model in the DML library. For example, FourWireCable.

d. RLGC

RLGC is an optional sub-parameter of Connections. It is defined similarly as the RLGC parameter that is described under the PackageModel keyword, except that this sub-parameter is placed under the Connections parameter under

`DesignLink`, which makes it a sub-parameter (as opposed to parameter) in this case. Refer to RLGC Sub-parameters Descriptions on page 67, to see examples of this sub-parameter as well as to learn more about its usage.

e. CircuitModels

`CircuitModels` is an optional sub-parameter of `Connections`. It is defined similarly as the `CircuitModels` parameter that is described under the `PackageModel` keyword except that this sub-parameter is placed under the `Connections` parameter under `DesignLink`, which makes it a sub-parameter (as opposed to parameter) in this case. Refer to CircuitModels Sub-parameters Descriptions on page 59, to see examples of this sub-parameter as well as to learn more about its usage.

2. Drawings

`Drawings` is a required parameter of `DesignLink`. It is used to define the names of the board files that are used for a specific DesignLink model. The drawings for each design name used in `DesignLink` must be included so that SigNoise can start the required software for the proper databases.

```
(DesignLink
  (BRD1_U1_to_BRD2_U2
    (Drawings
      (BRD1 "blm2_pos.brd")           ; Symbolic name and actual
      (BRD2 "blm2_pos.brd") ) )       ; board filename mapping
      ...
```

How to Create a DesignLink model

This sub-section describes the different ways you can create and edit a DesignLink model, namely, using the GUI of Allegro SI (such as the System Configuration Editor Form, and Clone Selection) and by manually writing DML files. The different methods of creating a DesignLink model are described here in decreasing order of suggested method, and common practices of model creation, from first to last.

System Configuration Editor form of Allegro SI

Figures 2-13 and 2-14 show the Signal Analysis Initialization and System Configuration Editor forms of Allegro SI, respectively. These are the two main forms that are used to create DesignLink models using Allegro SI. See Appendix E, "DML File of GUI Example (Figure 2-11) for DesignLink," for an ASCII version of this model example. Referring to Figures 2-13 and 2-14 as templates, you can create a DesignLink model as follows:

1. Open the Signal Analysis Initialization window

From Allegro SI, choose *Analyze – SI/EMI Sim – Initialize*...

2. Click the *New DesignLink...* button.

   A pop-up menu appears asking for the name of the new DesignLink.

3. Enter the desired DesignLink name and click *OK*.

   The System Configuration Editor window appears.

4. Enter all the values for the DesignLink that is being modeled as shown in Figure 2-14.

5. Click *OK* at the bottom of this window to create the model.

Figure 2-13 Signal Analysis Initialization Window of Allegro SI

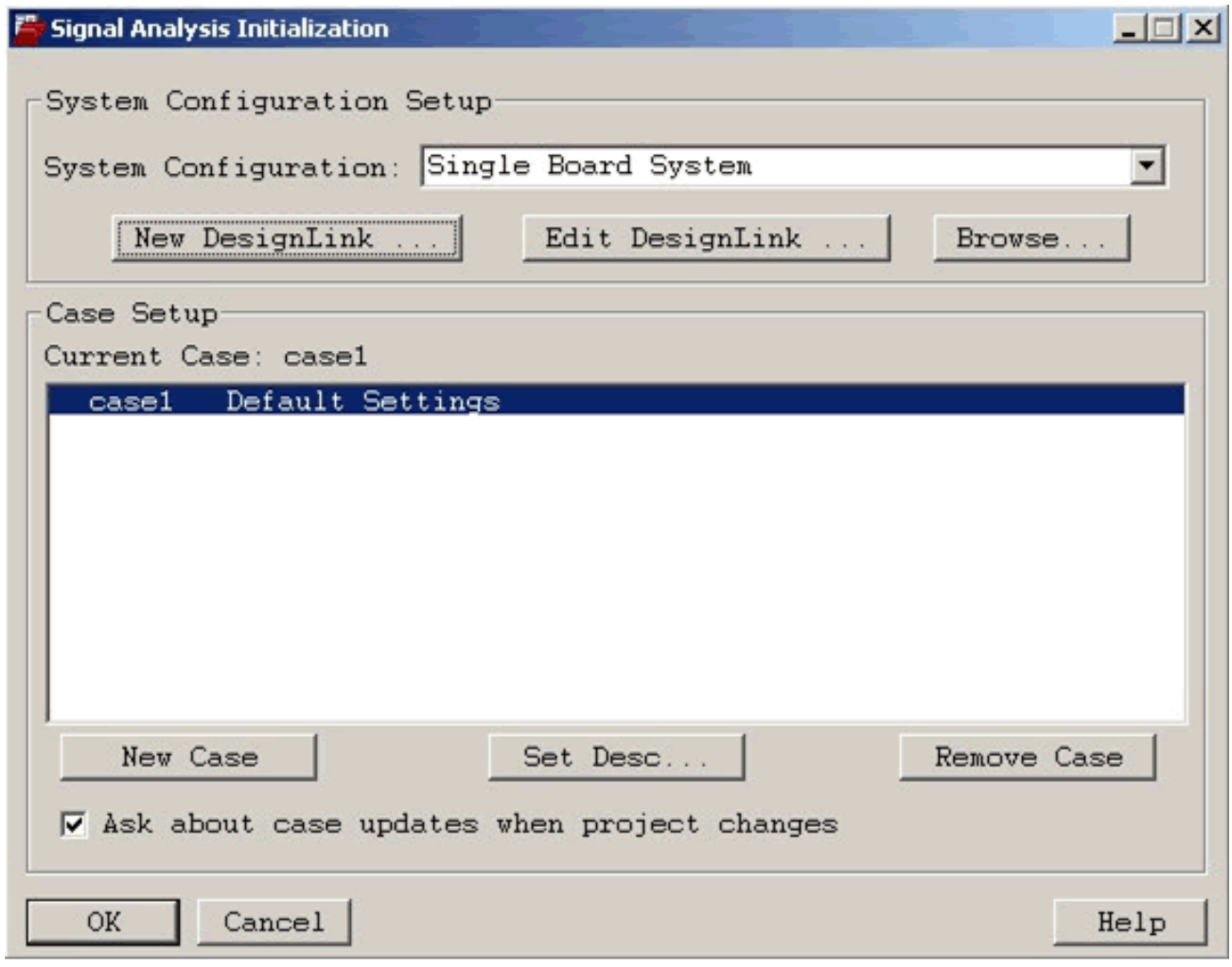
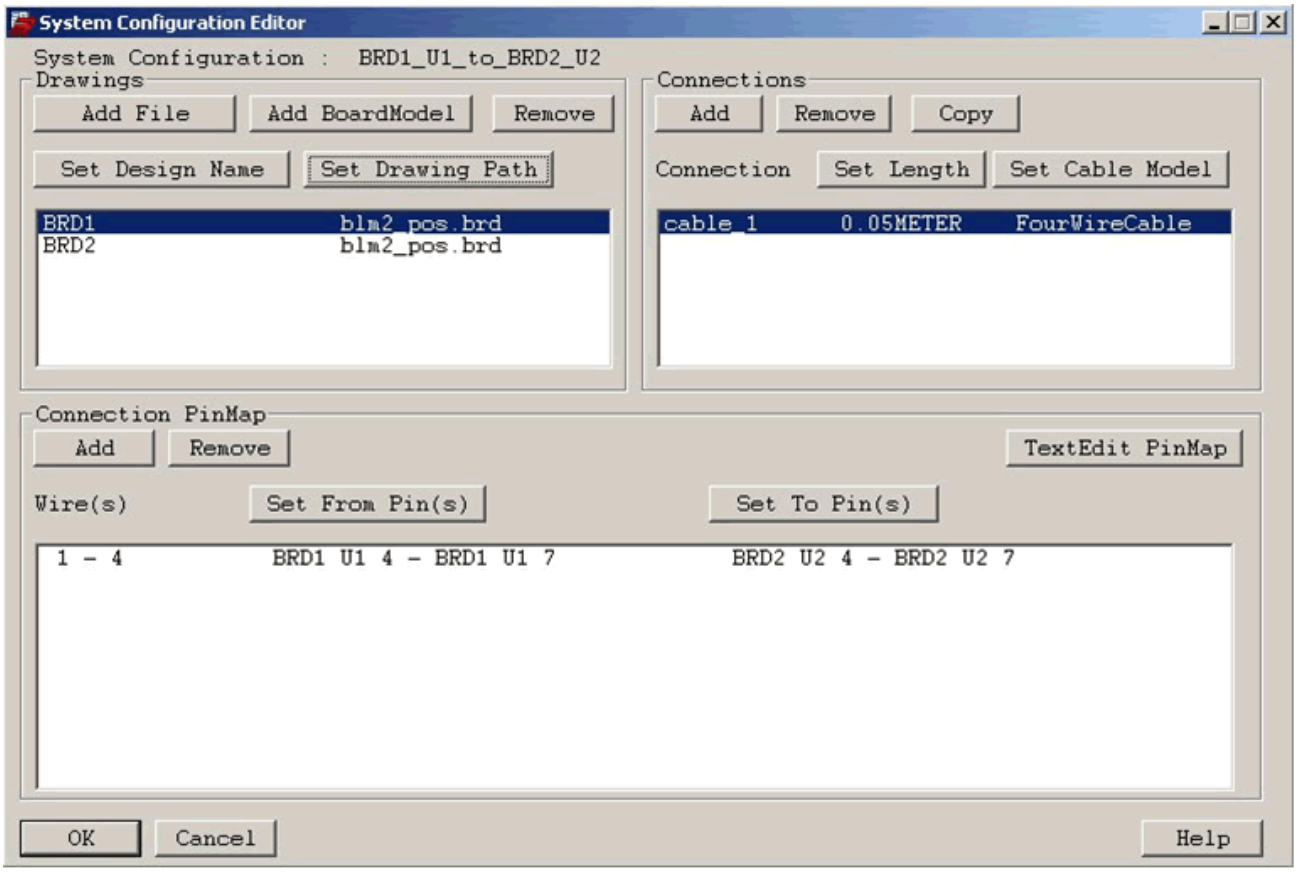


Figure 2-14 System Configuration Editor Window Example for the Design BRD1_U1_to_BRD_U2



Clone Selection

You can use the *CloneSelection* option to create DesignLink models for new devices that have similar properties as other devices and that already have DesignLink models in the existing DML Library.

Follow these steps to create a DesignLink model using this method:

1. Open the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library*…

   The Signal Analysis Library Browser window will now appear.

2. Click on the *Browse Models* button located at the top left hand corner of the Signal Analysis Library Browser window.

The Model Browser window appears.

3. Set the model filters on the Model Browser window to look for specific models to clone. You do this by using the pull down menus next to *Show Models From* (*All Libraries*, *Selected Device Library*, etc.) and *Model Type Filter* (*Any*, *DesignLink*, etc.) located at the top of the Model Browser window.
4. Select the desired model from the Model Browser window after filtering the models and click the *Add Model* button and then select *CloneSelection* from the pull-down menu.

   A menu appears asking for the new IbisIOCell model name.
5. Enter the new DesignLink model name and click *OK*.

   The new device model appears in the Model Browser window.
6. Select this new device model and click *Edit* to modify this new model to fit the intended specifications of the new device.

Manual creation of a DesignLink model

You can create a DesignLink by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token on page 19. Listed below are the steps to create a DesignLink model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the DesignLink template from Appendix E, “DesignLink Template,” into this file.
3. Edit the DML file.
4. Save the file and run `dmlcheck` to check for syntax errors. See Chapter 6, “dmlcheck Utility,” for more information.

Note: DesignLink template:

- This template is a skeleton of the DML file, which you can copy and paste into a text editor and then customize to model a specific device.
- You might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.
- Refer to the specific parameter and sub-parameter descriptions, which are listed in DesignLink Parameters and Sub-parameters Descriptions on page 125, for more details on their usage.

- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`DesignLink`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

Cable

The `Cable` keyword is used to model cables interconnecting multiple boards. It is referenced from a `DesignLink` to specify connections. A Cable model can use RLGC matrices to represent self and mutual impedances among wires. Alternately, a Cable model can also use CircuitModels to describe single line wire models and coupling among them in the form of Spice sub-circuits. Parameters of Cable include `RLGC`, `Wires`, and `CircuitModels`.

Required Parameters for Cable and Their Respective Required Sub-parameters

Required Parameters	Sub-parameters	
	Required	**Optional**
Wires	<wire_number>	-
RLGC	band	-
	dimension	
	data	

Optional Parameters for Cable and Their Respective Required and Optional Sub-parameters

Required Parameters	Sub-parameters	
	Required	**Optional**

Required Parameters	Sub-parameters	
CircuitModels	SingleLineCircuits[1]	CoupledLineCircuits
	SubCircuits	

1. SingleLineCircuits and optionally CoupledLineCircuits (one or both) have to be defined in a CircuitModels depending on the specific PackageModel.

Cable Parameters and Sub-parameters Descriptions

This section contains descriptions and examples of `Cable` parameters and sub-parameters. Some of the examples that are used in this section are excerpts from a larger DML file called `cds_samples.dml` which is located in the `<cds_install_dir>/share/pcb/signal/` directory.

- Wires

  `Wires` is a required parameter for `Cable`. It is used to state the number of wires in a Cable model. The number of wires must match the RLGC matrix dimension. Here's an example of `Wires`:

```
(Cable
  ("FourWireCable"
    (Wires 4)
...
```

- RLGC

  `RLGC` is a required parameter for `Cable`. In a Cable model, it is used to describe wire parasitics, which represent the parasitics matrices at different frequencies. The RLGC parameter is defined similarly as the `RLGC` parameter that is described under the `PackageModel` keyword, except that it is placed under the `Cable` keyword. Refer to RLGC Sub-parameters Descriptions on page 67, to see examples of this sub-parameter as well as to learn more about its usage.

- CircuitModels

  `CircuitModels` is an optional parameter for `Cable`. The `CircuitModels` format describes a cable model using SPICE sub-circuit syntax. The `RLGC` section is completely ignored when the `CircuitModels` keyword is defined. The `CircuitModels` parameter is defined similarly as the `CircuitModels` parameter that is described under the `PackageModel` keyword, except that it is placed under the `Cable` keyword. Refer to CircuitModels Sub-parameters Descriptions on page 59, to see examples of this sub-parameter as well as to learn more about its usage.

How to Create a Cable model

This section describes how a user can create and edit a Cable model, namely, using the GUI of Allegro SI (Clone Selection) and by manually creating the Cable model. The different methods of creating a Cable model are described here in decreasing order of suggested method, and common practices of model creation, from first to last.

Clone Selection

The *CloneSelection* option can be used to create Cable models for new devices that have similar properties as other devices and that already have Cable models in the existing DML Library. The following steps need to be taken in order to create a Cable model using this method:

1. Open the Signal Analysis Library Browser window

   From Allegro SI, choose *Analyze – SI/EMI Sim – Library*…

   The Signal Analysis Library Browser window appears.

2. Click the *Browse Models* button located at the top left hand corner of the Signal Analysis Library Browser window.

   The Model Browser window appears.

3. Set the model filters on the Model Browser window to look for specific models to clone. You can do this using the pull down menus next to *Show Models From* (*All Libraries*, *Selected Device Library*, etc.) and *Model Type Filter* (*Any*, *Cable*, etc.) located at the top of the Model Browser window.

4. Select the desired model from the Model Browser window after filtering the models and click on the *Add Model* button and then select *CloneSelection* from the pull-down menu

   A menu appears asking for the new Cable model name.

5. Enter the new Cable model name and click *OK*.

   The new model appears in the Model Browser window.

6. Select this new model and click *Edit* to modify this new model to fit the intended specifications of the new device.

Manual creation of a Cable model

You can create a Cable model by manually writing a DML file using any text editors, although it is very rarely done this way. It is important that the file that contains this manually created

model is saved with the extension `.dml`. The `<DMLfilename>.dml` has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token on page 19. Listed below are the steps to create a Cable model:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the Cable templatefrom Appendix F, “Cable Template,” into this file.
3. Edit the DML file.
4. Save the file and run `dmlcheck` to check for syntax errors.

Note: Cable template

- This template is a skeleton of the DML file, which you can copy and paste into a text editor and then customize to model a specific device.
- You might need to add or delete some parameters and sub-parameters in this template based on the specific device that is being modeled.
- Refer to the specific parameter and sub-parameter descriptions, which are listed in Cable Parameters and Sub-parameters Descriptions on page 132, for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you create manually, even though the DML file will not contain the top-level DML keyword that was originally specified in the DML file when viewed using a text editor invoked from Allegro SI. This is because Allegro SI uses the top-level DML keyword from the DML file to map the DML models to its corresponding `ModelTypeCategory` keywords (`Cable`, etc.) for the ease of model filtering using the Model Browser form and then removes the keyword when it displays the DML file.

3

DML Macromodeling

Macromodel Overview

MacroModel is an adaptable form of an IOCell model as opposed to the simpler hard-coded `bdrvr` model primitive that implements a simple IBIS 2.1 style model. A MacroModel is implemented by using an optional parameter under the `IbisIOCell` keyword. Macro models contain sub-circuits and therefore are extremely flexible and can be as complex as the user wants it to be. These sub-circuits contain active elements like `bdrvr`, and combine to make a model that can be used as any other IOCell in Allegro SI. This allows you to use all the normal stimulus and measurement capabilities with your MacroModel and assign it to a component just as you would any other IOCell. Otherwise, they are analogous to large package and ESpice models. Macromodels give the user the ability to model buffers that have more complex behaviors than a `bdrvr`. A MacroModel provides DML with flexibility above and beyond IBIS. It also gives the ability to quickly adapt to new IBIS constructs.

The MacroModel sub-circuits are described by a Spice type nodal language. The major circuit primitives employed are the `bdrvr` IBIS black box that are assembled together through various types of controlled current and voltage sources. In effect these sources implement a versatile array of voltage or time controlled switches and latches.

MacroModels are used for multistage buffers, early clamp (kicker) buffers, buffers with feedback, integrated macrocells (containing buffers + other digital/analog circuitry like flip flops etc.), and can be programmed by the user to implement a variety of other functions and behaviors.

The Spice sub-circuits are brought into the DML library system using the parameter MacroModel under the `IbisIOCell` keyword. The header of a MacroModel looks just like a regular IbisIOCell. The rise and fall times, and logic thresholds are derived from this header info. However, MacroModels contain a list of all the IbisIOCells that are in it (it is required to have at least one). For predictable and proper operation it is IMPORTANT that these IbisIOCells are IBIS2.1 or later with valid Voltage-Time (VT) curves.

The first table shows the required parameters of MacroModel and their respective required and optional sub-parameters. The second table shows the optional sub-parameters of MacroModel and their respective required and optional sub-parameters.

Required Sub-Parameters of MacroModel and Their Respective Required and Optional Sub-Parameters

Required Sub-parameters	Sub-parameters	
	Required	**Optional**

Required Sub-parameters	Sub-parameters	
Parameters	Buffers	MinTypMaxParams
NumberOfTerminals[1]	*<number_of_terminals>*	-
SubCircuits	.subckt	-

1. NumberOfTerminals is only required when defining a macro model for components that have other than 7 terminals.

Optional Parameters of MacroModel and Their Respective Required and Optional Sub-Parameters

Optional Sub-parameters	Sub-parameters	
	Required	**Optional**
MacroType	*<macromodel type>*	-
Language	hspice	-
PinTerminalsMap	*<pin_name pin_number>*	-
MeasurementNodes	*<name_of_node>*	-
MeasurementLocation	-	SiLocation TimingLocation

MacroModel Sub-Parameter Descriptions

This section contains descriptions and examples of MacroModel sub-parameters. Some of the examples that are used in this section are excerpts from a larger DML file called `cds_macromodels.dml` which is located in the `<cds_install_dir>/share/pcb/signal/` directory.

1. Parameters

   `Parameters` is a required sub-parameter for MacroModel. It is used to define `buffers` and other `minimum`, `typical`, and `maximum` values. It has two sub-parameters, namely `Buffers` and `MinTypMaxParams` as described below. Here's an example of `Parameters`:

```
(MacroModel
  (Parameters
    (Buffers                        ; See Buffers below
      ("BUFF1" "main_buff")         ; <buffer name> <buffer model name>
      ("BUFF2" "extraclampbuff") )
    (MinTypMaxParams                ; See MinTypMaxParams below
      ("TIMER_RISE"
        (typical "1n")
        (minimum "0.75n")
        (maximum "3n")
        (ftstype DieCapacitance) )
      ("TIMER_FALL"
        (typical "2n")
        (minimum "1n")
        (maximum "5n") )
      ("TIMER_WIDTH"
        (typical "2.5n")
        (minimum "2n")
        (maximum "3n") ) ) )
    (NumberOfTerminals "7" )
    (SubCircuits

* Modes kicker diodes using VOLTAGE Controlled latch
* power = 1
* output = 2
* ground = 3
* input = 4
* enable = 5
* power-clamp_reference = 6
* ground clamp reference = 7
* Model can be used as a driver if the enable is on and
* there is a valid input. If the enable is off , the device
* will be a receiver . The input terminal will have no
* effect.
```

```
".subckt Generic_kicker 1 2 3 4 5 6 7 ibis_file=kicker_models.inc BUFF1=main_buff
BUFF2=kicker_clamp TIMER_RISE=3n TIMER_FALL=5n
+ TIMER_WIDTH=3n
bdrvr_always_on 1 2 3 4 5 6 7 Model=BUFF1 file=ibis_file
bdrvr_ground_clamp 1 2 7 100 101 6 7 Model=BUFF2 file=ibis_file
v_kicker_input 100 7 0
e_enable 101 7 pwl 2 7
datapoints coeff_on_latch TransitionIndependentLatch=1
+ switchontime=TIMER_RISE switchofftime=TIMER_FALL
+ duration=TIMER_WIDTH
falling
0.7 0
end coeff
.ends Generic_kicker")
```

a. Buffers

`Buffers` is a required sub-parameter of `Parameters`. It is used to specify the names of the buffers, which are used inside a MacroModel. The buffer name and model name must match the sub-circuit buffer name and model name for a specific buffer. Referring to the example above, note that the buffer name, `BUFF1` and model name, `main_buff` match the sub-circuit definition, `BUFF1=main_buff` declared under the `SubCircuits .subckt` line sub-parameter.

b. MinTypMaxParams

`MinTypMaxParams` is an optional sub-parameter of `MacroModel`. It can contain minimum, typical, and maximum parameters used in a MacroModel sub-circuit. For fast, typical, nd slow simulations, if the `ftstype` is not specified, the default mapping will be same as that used for `C_COMP` or `DieCapacitance`. The `DieCapacitance` mapping sets the minimum value for fast and maximum for slow. The `ftstype RampRates` mapping sets the maximum value for fast and minimum for slow.

2. SubCircuits

`SubCircuits` is a required sub-parameter for `MacroModel`. It is used to define parameterized Spice sub-circuits. `SubCircuits` should be followed with a double-quote that begins the spice language description of the MacroModel's associated sub-circuit. The `SubCircuits` sub-parameter under `MacroModel` is similar to the `SubCircuits` sub-parameter under `CircuitModels`. Refer to CircuitModels Sub-parameters Descriptions on page 57, to learn more about the usage of this sub-parameter.

3. MacroType

`MacroType` is an optional parameter for `MacroModel`. It is used to define the type of macromodel. Here's an example of `MacroType`:

```
(MacroModel
  (Parameters
```

```
    (Buffers
    ("BUFF" "somebuffer") ) )
  (NumberOfTerminals "4")
  (MacroType "DiffIO")
  (SubCircuits
   ".subckt anotherexample 1 2 3 4 ibis_file=ibis_models.inc
   + BUFF=somebuffer
   ...
   .end anotherexample") )
...
```

4. Language

   `Language` is an optional parameter for `MacroModel`. It is used to define MacroModel sub-circuits that are written in a language other than ESpice. When this parameter is used, the tool assumes the sub-circuit is written according to the syntax rules of the simulator specified. Here's an example of `Language`:

```
(MacroModel
  (Parameters
    (Buffers
    ("BUFF" "Output") ) )
  (NumberOfTerminals "7")
  (Language "hspice")
  (SubCircuits

* This section contains the parameterized spice sub-circuits
* Following are the pin numbers and their corresponding default functions:
* power = 1
* output = 2
* ground = 3
* input = 4
* enable = 5
* power-clamp_reference = 6
* ground clamp reference = 7

          ".subckt hs_out 1 2 3 4 5 6 7
* HSpice output belem
          bdrvr 1 3 2 4 6 7
+ file = 'ibis3_2.ibs'
+ model = 'Output'
+ power = off
          .ends hs_out" ) ) )
```

5. PinTerminalsMap

   `PinTerminalsMap` is an optional sub-parameter for MacroModel. It causes the terminals listed to appear in the list of custom measurements nodes, and also makes the terminals to be editable in the Stimulus Editor. Here's an example of `PinTerminalsMap`:

```
(MacroModel
  (PinTerminalsMap
```

```
    (4 "Data")
    (5 "Enable") )
    ...
```

6. MeasurementNodes

   `MeasurementNodes` is an optional sub-parameter for `MacroModel`. It is used to specify internal nodes where various measurements can be made for a component when waveform data is captured at that node (by the use of an ESpice `.node_param` statement). The `MeasurementNodes` sub-parameter is similarly defined under the `PackagedDevice` keyword. Refer to IbisDevice Sub-parameters Descriptions on page 32, to learn more about this sub-parameter's usage. Here's an example of `MeasurementNodes`:

```
(MacroModel
  (Parameters
    (Buffers
      ("BUFFER" "mnode") ) )
  (MeasurementNodes
    ("vdd_die")
    ("gnd_die") )
  (NumberOfTerminals "6")
  (MacroType "IO")
  (SubCircuits "
  .subckt Example_IO 1 2 3 4 5 6 ibis_file=ibis_models.inc
  BUFFER=mnode
  .node_param 1 NAME=(name(2).vdd_die) cycles=(all) PRINT
...
") )
```

7. MeasurementLocation

   `MeasurementLocation` is an optional sub-parameter for `MacroModel`. It is similarly described under the `PackagedDevice` keyword. Refer to IbisDevice Sub-parameters Descriptions on page 32, to learn more about its usage.

8. SubCircuitLibrary

   `SubCircuitLibrary` is an optional sub-parameter for `MacroModel`. It is useful for model developers who want to encrypt subcircuits and have an exposed wrapper that can be used to pass control parameters to the encrypted subcircuit. The value of SubCircuitLibrary is a DML model name. The DML files containing the SubCircuitLibrary model and the MacroModel which refers to it must be located in the same directory, as shown below:

```
("scl_example_top_level.dml"
 (IbisIOCell
  (eq_amp_rcvr_shell
   (MacroModel
    (MacroType TDiffIO )
    (SubCircuitLibrary passive_eq )
```

```
    (NumberOfTerminals 8 )
    (Parameters
     (MinTypMaxParams
      (rt 50 )
      (pas_on 1 ) ) )
    (SubCircuits "
*  The comment character inside the SubCircuits double quotes is a *.
*
*  The items listed in the Parameters section above are passed in and override
*  items in the various subcircuits below.
*  The Parameters for this particular Macromodel are as follows:
*
*  Parameter rt is termination resistance
*  Parameter pas_on is used to turn the passive equalization on or off.
*  A value of 1 turns it on, and a 0 turns it off.
*
* ========================================================================
*
*  This MacroModel is of MacroType TDiffIO. It is a differential bdrvr. This
*  differs from a single-ended MacroModel, which uses a 7-terminal subcircuit.
*  A differential MacroModel such as this uses an 8-terminal subcircuit, the extra
*  terminal being the N-side output.
*
*  The terminals in an 8-terminal differential MacroModel are as follows:
*      power = 1
*      outp = 2
*      ground = 3
*      input = 4
*      enable = 5
*      power_clamp_reference = 6
*      ground_clamp_reference = 7
*      outn = 8
*
* ========================================================================
*
*  This is the top-level subcircuit for MacroModel eq_amp_rcvr_shell.
*  It MUST have the same name as the IOCell, or it will not work.
*
*  In this case, the nodes used are power, ground, outp, and outn.
*  The in, enable, and clamp nodes are not used in this example.
.subckt eq_amp_rcvr_shell 1 2 3 4 5 6 7 8
+ padcap=1p
+ rt=50
+ pas_on=1
+ rpas=130
+ cpas=0.5p
+ rt2=1000
+ gain=3
rp 2 1 rt
rn 8 1 rt
cp 2 3 'padcap'
```

```
cn 8 3 'padcap'


*  This receiver includes a passive EQ and also an amplifier. The parameters
*  declared in the subcircuit above with the pas suffix are used in the EQ
*  circuit. For example, to modify the EQ, you could increase the cpas
*  parameter from 0.5pF to 1.5pF. The amplification is controlled with the
*  gain parameter above. To turn off amplification, change the gain from 3
*  to 1.

*  This is the subcircuit call for the passive EQ. The actual subcircuit is
*  stored in a separate DML file as a SubCircuitLibrary model, called passive_eq.
*
xpassive 2 8 outp outn 1 passive_eq

*  This voltage source derives the differential output of the passive EQ,
*  and then amplifies it by the gain parameter.
erx rx 0 v='v(outp, outn) * ((rpas + rt2) * gain/rt2)'

*  This statement prints out the final differential voltage, after the amplification.
.node_param rx name=(rx_ name(2) _ name(8) _diff) print

.ends eq_amp_rcvr_shell
" ) )
   (Model
    (ModelType Input ) )
   (LogicThresholds
    (Input
     (High
      (maximum 0.5 )
      (minimum 0.5 )
      (typical 0.5 ) )
     (Low
      (maximum 0.5 )
      (minimum 0.5 )
      (typical 0.5 ) ) ) )
   (Technology CMOS )
   (VIReferenceTemperature
    (maximum 100 )
    (minimum 0 )
    (typical 25 ) )
   (Notes "This model is provided as a template by Cadence Design Systems,
and is intended to be used as a starting point from which users can derive
their own specific behavioral models." ) ) )
 (LibraryVersion 136.2 ) )
```

Here is an example of a SubCircuitLibrary DML model:

```
("scl_example.dml"
 (SubCircuitLibrary
  (passive_eq "
*  This is the subcircuit definition for the passive EQ.
```

```
.subckt passive_eq inp inn outp outn nvdd
xp inp outp nvdd eq_ckt
xn inn outn nvdd eq_ckt

*  This is the actual EQ subcircuit
.subckt eq_ckt in out nvdd
rc in out 'rpas'
c in out 'cpas * pas_on'
rt2 out nvdd rt2
.ends eq_ckt

.ends passive_eq
" ) )
 (LibraryVersion 136.2 ) )
```

How to Create a MacroModel

This section describes the different ways a user can create a MacroModel, namely, using a model translation utility such as ibis2signoise and by manually writing the DML files. Note that MacroModels *cannot* be created using the GUI of Allegro SI. The different methods of creating a MacroModel are described here in decreasing order of suggested method, and common practices of model creation, from first to last.

Model Translation Utility (Ibis2Signoise)

You can translate models that are written in IBIS into DML using the `ibis2signoise` utility. Refer to Chapter 8, "Model Translation," to learn how this can be done. Note that ibis2signoise will automatically create a number of different MacroModels for various types of IBIS syntax.

Manual creation of a MacroModel

You can create a MacroModel by manually writing a DML file using any text editors. It is important that the file that contains this manually created model is saved with the extension `.dml`. The `<DMLfilename>.dml` has to exactly match the first token of the DML file, which is the name of the DML file, as explained in Token on page 19. Note that the file name is case sensitive. Listed below are the steps to create a MacroModel:

1. Open and save a new file as `<DMLfilename>.dml` using a text editor.
2. Copy and paste the MacroModel starter file from Appendix G, "MacroModel Starter File," into this file.
3. Edit the DML file.

4. Save the file and run `dmlcheck` to check for syntax errors. See Chapter 6, "dmlcheck Utility," for more information)

Note: MacroModel starter file:

- This starter file is a simple example of an output resistor that is embedded in IbisIOCell, which you can copy and paste into a text editor and then customize to achieve the desired behaviors of the new model.
- You might need to add or delete some parameters and sub-parameters in this starter file based on the specific device that is being modeled. The values of the parameters and sub-parameters might also need to be changed depending on the new model.
- Refer to the specific parameter and sub-parameter descriptions, covered in MacroModel Sub-Parameter Descriptions on page 138, for more details on their usage.
- It is very important to include the top-level DML keyword in the DML file you manually create, even though when the DML file is viewed using a text editor (which is invoked from Allegro SI), it will not contain the top-level DML keyword that was originally specified in the DML file. This is because Allegro SI uses the top-level DML keyword from the DML file you created to map the DML models to its corresponding ModelTypeCategory keywords (`MacroModel`, and so on) for the ease of model filtering using the Model Browser form, and then removes the keyword when it displays the DML file.

MacroModel Examples

This section shows some of the common uses of MacroModels through examples.

Example 1

IBIS keyword `[Driver Schedule]` is handled as an enhancement to DML using MacroModel

The IBIS driver schedule keyword would be automatically translated by Allegro SI to a Spice sub-circuit inside MacroModel. Here are two examples of the IBIS `[Driver Schedule]` keyword usage in IBIS and its corresponding DML translation:

Example 1.(a)

In this example, the input to ModelA is delayed by the Rise_on_dly of 0.13n in the first e source. The second e source controls the enable signal to ModelA using a two-dimensional structure. Firstly, the enable is only active when the device is driving (voltage between node 5 and 3 is 1V). Then the enable follows the *rising* time-voltage points that enable the model by 0.13nS and disables it by 0.47nS for the Rise_off_dly.

For a `[Driver Schedule]` keyword of

```
[Driver Schedule]
| Model_name   Rise_on_dly   Rise_off_dly   Fall_on_dly   Fall_off_dly
   MODELA         0.13ns        0.47ns           NA            NA
```

The translation to SPICE sub-circuit inside a MacroModel would be

```
.subckt macro 1 2 3 4 5 6 7 ..... THRSH=1e-4
   bdrvr 1 2 3 input enable 6 7 Model=ModelA file=ibis_file C_compx=0
   e input 3 pwl 4 3 delay=0.13n
   e enable 3 pwl 5 3 4 3
   datapoints vv
   0 0
   1 1
   end vv
   datapoints timecoeff vhilo=THRSH vlohi='1-THRSH'
   rising
   0 0
   0.129n 0
   0.13n 1
   0.4699n 1
   0.47n 0
   falling
   0 0
   end
```

Example 1.(b)

This example is similar in implementation to Example 1.(a), except that now a `Fall_on_dly` is used as well.

For a `[Driver Schedule]` keyword of:

```
[Driver Schedule]
| Model_name   Rise_on_dly   Rise_off_dly   Fall_on_dly   Fall_off_dly
    MODELA        0.13ns          NA           0.47ns          NA
```

The translation to SPICE sub-circuit inside a MacroModel would be

```
.subckt macro 1 2 3 4 5 6 7 ..... THRSH=1e-4
 + bdrvr 1 2 3 input enable 6 7 Model=ModelA file=ibis_file
 + C_compx=0
 vdummy  dummy 3 1
 e input 3 pwl dummy 3 4 3
 datapoints vv
 0 0
 1 1
 end vv
 datapoints timecoeff vhilo=THRSH vlohi='1-THRSH'
 rising
 0.13n 0
* start rising now
 0.14n 1
 falling
 0.47n  1
* start the fall
0.48n 0
end timecoeff
e enable 3 pwl 5 3 4 3
datapoints vv
0 0
1 1
end vv
datapoints timecoeff vhilo=THRSH vlohi='1-THRSH'
rising
0 0
0.129n 0
0.13n 1
falling
0 0
0.469n 0
0.47n 1
end
```

Example 2:

A model that scales the V/I curves based on power droop and ground bounce.

Allegro SI does feed non-ideal power to the power terminals of a device. That non-ideal power will be reflected on the output node of an IBIS driver through either the pullup or pulldown characteristics. However, a second-order effect of non-ideal power is that it changes the gate voltage on the output transistors that actually modifies the strength of the VI curves in real-time. Allegro SI can also handle this effect can by wrapping your IBIS driver as an 11-terminal MacroModel as shown here:

```
.subckt scaled_driver 1 2 3 4 5 6 7 ibis_file=ibis_models.inc
+ BUFF=buffer1
bdrvr 1 2 3 4 5 6 7 8 9 10 11 model=BUFF file=ibis_file

* do not scale power and ground clamps

v 10 0 1
v 11 0 1

* scale 1.5V pullup curve by as much as 1.3 for +300mV
* and 0.78 for -300mV

eup 8 0 pwl 1 0
datapoints vi
1.8 1.3
1.7 1.2
1.6 1.1
1.5 1.0
1.4 0.921
1.3 0.85
1.2 0.78
end vi

* scale pulldown VI curve for ground bounce,
* positive bounce weakens, negative bounce
* multiplies the curve as shown since the gate
* voltage is greater

edn 9 0 pwl 3 0
datapoints vi
0.3 0.89
0.2 0.92
0.1 0.96
0 1
-0.1 1.05
-0.2 1.09
-0.3 1.13
end vi
```

```
.node_param 3 name=(name(2).die_gnd) print
.node_param 9 name=(name(2).gnd_scaler) print

.ends scaled_driver
```

Example 3

A programmable buffer implemented with bit strapping options.

IBIS often uses the `[Model Selector]` keyword to handle various programmable options by pointing to unique `[Model]`s, each with their own VI and VT data. However, many programmable buffers simply offer stronger/weaker VI options or faster/slower VT “slew” options. In this example, the MacroModel establishes different impedance and slew “bits” that adapt the model’s VI and VT data (respectively) through the use of if-then-else and multipliers in the .param statements. It is a complex implementation, but is worth studying to better understand the power available in the MacroModel structure and the use of parameters.

```
(MacroModel
  (Parameters
    (Buffers
      (“BUFF” “Base_io”) )
    (MinTypMaxParams
      (“IMP_0” “0”)
      (“IMP_1” “1”)
      (“IMP_2” “1”)
      (“IMP_3” “0”)
      (“SLEW_0” “0”)
      (“SLEW_1” “0”)
      (“SLEW_2” “0”)
      (“SLEW_3” “0”) ) )
    (NumberOfTerminals “7”)
    (SubCircuits
      “.subckt prog_buffer 1 2 3 4 5 6 7 ibis_file=ibis_models.inc
      + BUFF=Base_io IMP_0=0 IMP_1=1 IMP_2=1 IMP_3=0 SLEW_0=0 SLEW_1=0
      + SLEW_2=0 SLEW_3=0
    .param impedance='IMP_3*8 + IMP_2*4 + IMP_1*2 + IMP_0*1'
    .param slew='SLEW_3*8 + SLEW_2*4 + SLEW_1*2 + SLEW_0*1'

.param driver 'slew*16+impedance'
.param tol='1e-6'

* the following tests the value of the parameter driver and then
* either decreases the strength of the pulldown by multiplying it * by 0.83, or increases its strength
by multiplying by 1.16, or
* leaves it as it is by multiplying by 1.0

.param vipdf='if(abs(driver-5) < tol || abs(driver-21) < tol ||  abs(driver-37) < tol) (.83)
elseif(abs(driver-7) < tol || abs(driver-23) < tol ||  abs(driver-39) < tol) (1.16) else (1.0)'
```

```
.param vipuf='if(abs(driver-5) < tol || abs(driver-21) < tol ||  abs(driver-37) < tol) (.83)
elseif(abs(driver-7) < tol || abs(driver-23) < tol ||  abs(driver-39) < tol) (1.16) else (1.0)'


.param vtr1f='if(abs(driver-5) < tol) (0.84)  elseif(abs(driver-7) < tol) (1.15) elseif(abs(driver-21)
< tol) (0.73) elseif(abs(driver-22) < tol) (0.83) elseif(abs(driver-23) < tol) (0.92) elseif(abs(driver-
37) < tol) (0.6) elseif(abs(driver-38) < tol) (0.7) elseif(abs(driver-39) < tol) (0.8) else (1.0)'


.param vtr2f='if(abs(driver-5) < tol) (0.84)  elseif(abs(driver-7) < tol) (1.15) elseif(abs(driver-21)
< tol) (0.73) elseif(abs(driver-22) < tol) (0.83) elseif(abs(driver-23) < tol) (0.92) elseif(abs(driver-
37) < tol) (0.6) elseif(abs(driver-38) < tol) (0.7) elseif(abs(driver-39) < tol) (0.8) else (1.0)'


.param vtf1f='if(abs(driver-5) < tol) (0.84)  elseif(abs(driver-7) < tol) (1.15) elseif(abs(driver-21)
< tol) (0.73) elseif(abs(driver-22) < tol) (0.83) elseif(abs(driver-23) < tol) (0.92) elseif(abs(driver-
37) < tol) (0.63) elseif(abs(driver-38) < tol) (0.7) elseif(abs(driver-39) < tol) (0.8) else (1.0)'


.param vtf2f='if(abs(driver-5) < tol) (0.84)  elseif(abs(driver-7) < tol) (1.15) elseif(abs(driver-21)
< tol) (0.73) elseif(abs(driver-22) < tol) (0.83) elseif(abs(driver-23) < tol) (0.92) elseif(abs(driver-
37) < tol) (0.63) elseif(abs(driver-38) < tol) (0.7) elseif(abs(driver-39) < tol) (0.8) else (1.0)'


bdrvr 1 2 3 4 5 6 7 VIScale_pulldown='vipdf' VIScale_pullup='vipuf'
+ TVScale_rise_1='vtr1f' TVScale_fall_1='vtf1f' TVScale_rise_2='vtr2f'
+ TVScale_fall_2='vtf2f' Model=BUFF File=ibis_file


.ends prog_buffer") )
```

Example 4

Behavioral input model

This MacroModel allows you to model an IOCell that has a complete receiver model. The model plots the waveforms at the output to the receiver, as seen from inside the integrated circuit. The receiver output models both the switching characteristic given a reference voltage (Vref) as well as the edge-dependent propagation time across the receiver itself.

```
.subckt behavioral_input 1 2 3 4 5 6 7 ibis_file=ibis_models.inc BUFF=CDSDefaultOutput


bdrvr 1 2 3 4 5 6 7 model=BUFF file=ibis_file


* isolate inner sections of receiver model from external pins


eina in 0 v='v(2,3)*(1-v(5,3))'


* this disables the receiver when the driver is tri-state
epwr power 0 v='v(1,3)'
Vvref vref 0 1.0


* hard-code vref here
xdiffamp power in vref 0 out bhvr_in


* receiver output capacitance
Cout out 0 20ff


* makes stimulus go 0 to Vcc
```

```
escstim stim 0 v='v(4)*v(1,3)'

* makes enable go too
escenab enab 0 v='v(5)*v(1,3)'

* prints/names waveforms for display with unique names
.node_param stim name=(name(2).drvr_in) print
.node_param out name=(name(2).rcvrout) print

**************************************************
*         BEHAVIORAL INPUT SUBCIRCUIT              *
**************************************************

.subckt bhvr_in   1      2    vref    3     out
*                power input  vref  ground output

* this is the low-pass filter capacitance value
.param c_filt='1.5p'

* subckt for amplifier - assuming vref = voltage of node ref
* first generate vcritcial.  This is the Actual receiver DC switching * point

* This is the DC transfer curves as a function of Vref
ecrit crit 3 pwl lpf2 3
*    outputs     inputs
* pwl=type of source

datapoints vv
* vref  critical switch voltage  (columns)
0.5 0.7
1.0 1.0
1.5 1.3
end vv

* Low Pass Input Stage
* The capacitance is used to
* compensate for delay characteristics
* with respect to Vref.  The front-end
* LPF filters out spikes...at Vref=1.0
* the spike width rejected is 100 ps.

Cfilter lpf 3 'c_filt'
Rin 2 lpf 100
Rout lpf 3 1e6

* i = Vr/R, R=100.  Scale by 1e10, R*Scale = 1e12

* THE NEXT LINE SOLVES DV/DT OF THE INPUT SLOPE!!!!
* dV/dt = V/R*C
ederiv deriv 3 v='v(2,lpf)/(100*1e10*'c_filt')'
```

```
Rref vref lpf2 100
Cref lpf2 3 1.7p
Rout lpf2 3 1e6

eout 6 3 pwl lpf crit
* slews the transfer curve to 80mV/2.0V

datapoints vv
5.0 0
0.040 0
-0.040 2.0
-5.0 2.0
end vv

* This delay line adds delay based on the rise/fall edge rate in the
* transition region (+/- 150 mV).  This region is of arbitrary size
* and may need to be changed for future refinement.

* ---------------------------Extra Delay Line ----------------------

x1 deriv lpf crit 6 8 lpf2 3 delay_adder

Cdamp 8 3 1pf
*---------------------------------------------------------------------

* Now add the regular intrinsic delay (the delay from a 100 ps edge)
* Get this from testing circuit with the fastest edge speed you
* expect (100pS in this case).
* This is a special e source, a threshold controlled voltage source
* with a time delay in the left column.

* Note that the output voltage(s) of the receiver is set here.

e out 3 pwl 8 3

datapoints timecoeff vlohi=0.8 vhilo=0.8
rising
0 0
.020n 0
.090n 2.0
falling
0 2.0
.110n 2.0
.190n 0
end timecoeff

*---------------------------Delay Adder Subcircuit Def ---------------

.subckt delay_adder deriv1a filtered_in crit delayin out vref 3

.param cderiv 1.0
```

```
* If dV/dt < 10.0V/ns and > 0 (rising edge) add up to 40-60 ps of
* delay, if
* dV/dt < 0, then add up to 50-60 ps of delay, otherwise add 1ps (~
* 0).

* table is now directly in terms of dly
* Check for transition region.  If not in the transition region
* do no delay adjustments.

* Hack for numerical integration oscillations

ederiv1 deriv1 0 v='(v(deriv1a)+prev(v(deriv1a)))/2'

* the extra delay is a function of 100mV either side of vref, it may
* cost you up to 60pS if you are slow

edlylvla dlylvl 0 v='if((abs(v(filtered_in,crit)) < 0.1)) (v(deriv1)) else (3)'

edly dly 0 pwl dlylvl 0

datapoints vv
-3.0 0.001n
-2.0 0.001n
-1.0 0.005n
-.2  0.005n
-.1  0.020n
-.04 0.050n
0.0 0.060n
.04 0.040n
.1   0.015n
.2   0.005n
1.0  0.005n
2.0  0.001n
3.0  0.001n
end vv

x delayin 31 dly 0 vardlysec sec=5
x 31 32 dly 0 vardlysec sec=5
x 32 33 dly 0 vardlysec sec=5
x 33 34 dly 0 vardlysec sec=5
x 34 out dly 0 vardlysec sec=5

* Resistor added to eliminate reflections

Rout out 0 50
.ends delay_adder

.subckt vardlysec delayin out dly ref imp=50 sec=5

ecap cap 0 v='v(dly)/(imp * sec)'
```

```
eind ind 0 v='imp * imp * v(cap)'

vind delayin in1 0

* no charge conservation
*            V=   L    *  di / dt
eind in1 out v='v(ind) * ddt(i(vind))'
*            I    C    *  dv / dt
gcap out ref i='v(cap) * ddt(v(out,ref))'

.ends vardlysec

.ends bhvr_in

.ends behavioral_input
```

4

DML Connector Models

Connector Models Overview

Connector models are basically models of connectors on a board. They are usually defined under the PackageModel keyword in DML. Refer to Examples of Connector Models on page 69, to see some examples of it.

5

DML HSpice Models

HSpice Models Overview

HSpice models can be defined in DML under the `MacroModel` and `ESpice` parameters as described in MacroModel Sub-Parameter Descriptions on page 136.

6

dmlcheck Utility

dmlcheck Overview

`dmlcheck` is a program used to check SigNoise models, which are in DML format. This program is mainly a syntax checker. The program allows many questionable but syntactically legal constructs, some of which cause convergence problems in `tlsim`, the field solver in SigNoise. A robust model checker is needed. Users need high confidence that any model that passes through `dmlcheck` without warnings or errors will not cause problems during `tlsim`, and will not contain data, which is obviously erroneous.

In general, each data in the model should be sanity checked. A value, which is physically impossible, such as a negative capacitance, would be treated as an error. A value that is out of reasonable bounds would be treated as a warning. Such warnings will help users to find incorrectly entered units or misplaced decimal points. Where minimum, typical, and maximum values are entered for a parameter, it will be an error to enter these numbers out of order, except for a VICurve, a TVCurve, or a VIReferenceTemperature. The reasonable bounds for a parameter may depend on other parameters in the model. For a given type of model, some parameters are always required, while other parameters are always optional. A parameter may be required or optional, depending on which other parameters are present or absent. Some checks depend on the device technology, with ECL having special rules. Clearly it is important to have sanity checks on the magnitudes of numbers in the DML file. For example, a reference to a model from a major IC vendor with a dt number of 3.2e-10ns, when they meant to have 0.32ns. Also, it is appropriate to check for consistency between values. For example, large pin parasitics are inconsistent with small dt numbers. The goal is to find major discrepancies, such as the wrong units having been used.

Checking Device Model and Library Syntax

Use the `dmlcheck` utility to check the syntax of one or more library files or models. There are actually several ways to invoke `dmlcheck`. The IBIS model editors have *DML Check* buttons, which enable you to check the syntax of new models as they are created. You can check a group of libraries (`.dml` files) using a command line entry. Also, in some cases, the `dmlcheck` utility is invoked automatically, such as after assigning IOCell models from the Database Setup Advisor. Ibis2signoise can call `dmlcheck` automatically as well when it is run. Still another way to invoke `dmlcheck` is to use one of the following *Library Audit* options.

To check device model and library syntax from Allegro SI:

1. Select one of the options shown in the following figure from the main menu of either the Allegro SI or PCB Editor, APD.

   A file browser appears.

2. Select or enter the name of the library (`.dml`) or library list (`.lst`) file to be checked.

   Upon completion of the library syntax check, the `dmlcheck` message log window appears with a description of any warnings or errors found with a library or model format. A sample log is shown in the following figure. In cases where `dmlcheck` has modified a curve to fix some problem, a confirmer pop-up is displayed asking to overwrite the original library with the output from dmlcheck.

To check library syntax from the UNIX command line:

At the operating system prompt, enter the `dmlcheck` command with the following arguments:

```
dmlcheck [options] <library_filename>...
```

dmlcheck Command Arguments

Argument	Function
-o <*extension*>	Appends the specified extension to the output file created by dmlcheck
<*library_filename*>	One or more device model libraries to be checked
-bufferdelay	Calculate BufferDelay for each IbisDevice pin (requires `-o` <*extension*>)
-curvedir dir	Create a directory of waveform files. A waveform file for each VICurve and TVCurve is placed in this directory. Use sigwave to view the waveforms. The waveforms show original typ/min/max curves as well as the fixed curves.
-version	Print the dmlcheck program version and exit.

Note: The `dmlcheck` utility checks the syntax of each file in turn. It prints errors and warning messages as necessary to standard text output and also reports when a file checks out okay.

Command Line Examples

- This example checks all library files in the current directory.

  ```
  dmlcheck *.dml
  ```

- The following example checks all library files in the current directory, and writes converted data into files with the `.new` extension.

  ```
  dmlcheck -o new *.dml
  ```

- The following example simulates to measure BufferDelay, which is stored in the `.new` output file.

  ```
  dmlcheck -bufferdelay -o new *.dml
  ```

- The following examples creates a `./curves` directory into which is placed a `.sim` waveform file for each V/I and V/T curve.

  ```
  dmlcheck -curvedir curves *.dml
  ```

Each waveform file contains the following curves:

  - The minimum, typical, and maximum curves in the input file.
  - The same three curves after dmlcheck fixing.

The files are named with the IOCell name and curve name, separated by an underscore. For example, CDSDefaultIO_Pullup.sim. Use SigWave to view the curves.

- This example checks all library files in the current directory, and writes converted data into files with the extension `.new` and also writes curve waveforms in the `./curves` directory after creating the directory.

  ```
  dmlcheck -o new -curvedir curves *.dml
  ```

7

Cadence Default Model Library

Library Overview

The Cadence sample device model library is:

`/<install_dir>/share/pcb/signal/cds_iocells.ndx`

Look at this file for additional details of the structure of a Device Model Library file, as well as for examples of the different types of device models. You can use the Model Browser to list the models in the library, and to edit, add, and delete models.

The models listed in the Cadence Default Library are split into three categories:

- DIG_LIB Library Models

  Standard digital logic families containing 217 unique parts in a format directly compatible with SigNoise with their corresponding IBIS files. These models have been made from spice files from **Signetics** (except for one device).

- DEFAULT_LIB Library Models

  Default set of models having IOCELLS for GTL, PCI and ASIC types. These IO cells can be used to select the right buffer for a given application.

- PACKAGE Library Models

  Default set of models having IOCELLS for GTL, PCI and ASIC types. These IO cells can be used to select the right buffer for a given application.

For a complete list of models, refer to the following file in your Cadence installation directory.

`share/pcb/signal/cds_partlib.ndx`

DIG_LIB Library Models

The digital device model library supports models for parts from four type of digital logic families. A part's digital logic family refers to the different processes and implementations used in the manufacture of the parts used in integrated circuits. For example, you can use bipolar transistor-transistor-logic, CMOS, bipolar emitter coupled logic, or a combination of several technologies.

Each technology has different input and output parameters. Table shows the digital logic families and number of parts under each technology that are included in the library.

Digital Logic Families Technology

Technology Family	Description	Number of Parts
FTTL	Fast Transistor-Transistor Logic for speed critical circuits	106
ABT	Advanced BICMOS Technology for bus interfaces with high drive, lower power consumption and fast propagation	36
ALVC	Advanced Low Voltage CMOD for low power consumption	10
ALS	Advanced Low Power Schottky for low power consumption in non-speed critical circuits	65

DEFAULT_LIB Library Models

The next table shows the default model families and number of parts under each technology that are included in the library.

Default Model Families

Technology Family	Description	Number of IOCells
GTL	GTL IO cell derived from Texas gtl spice file	01
PCI	PCI Compatible IO cells for both 3v and 5v derived from Intel pci spice file	04
ASIC	ASIC IO cells for different current capability both for 3 and 5v; with and without slew made from suitable approximations for the ASIC CMOS technology	22

Package Library Models

The default package models include:

- 20dip
- 14soic
- 16soic
- 20soic
- 16ssop
- 20ssop
- 28plcc
- 44plcc

Library Location and Structure

The Cadence Default Model Library is located in your Cadence installation directory. The default path is: `/cds/share/pcb/signal/SignalPartLib`. The directory contains two sub-directories corresponding to the three model categories.

- DEFAULT_LIB

  Contains SigNoise `.dml` files and their corresponding `.ibs` files. The corresponding `assumption.txt` file lists the assumptions, approximations, and validation test_set_up used.

- DIG_LIB

  Contains subdirectories corresponding to the digital logic family. Each digital logic family subdirectory contains files for Device models and IOCell models (`.dml` files). There is one device model in each file. The corresponding IBIS files(`.ibs` files) also exist. There is one `DIGlib_assump.txt` file giving the test setups used to validate each family.

- PKG_LIB

Note: In the signal directory, the simulator uses the device model index file `cds_partlib.ndx` to quickly load groups of models.

8

Model Translation

Overview

The SigNoise simulator supplies utilities for the purpose of translating IBIS and Quad model formats into the Device Model Library (DML) file format.

There are two methods that you can use to translate models:

- Model translation through the user interface
- Model translation at the command line

Translation

To translate an IBIS or Quad model file to DML format through the user interface:

1. Open the Signal Analysis Library Browser *(Analyze – SI/EMI – Library*) from the toolbar of either Allegro SI or PCB Editor, APD.
2. Click the *Translate* button.

   A menu appears, listing the translator formats. The formats include:

   - IBIS — Runs the `ibis2signoise` translation utility. Use `ibis2signoise` if you have an industry standard `.ibs` file.
   - QUAD — Runs the `quad2signoise` translation utility. Use `quad2signoise` if you have a `.mod` file (and any required `.tlb` file) for use with a ViewLogic TLC or XTK based simulator.
3. Click the translator you want to use.

   A file browser appears. For IBIS models, files of type `.ibs` are listed. For Quad models, files of type `.mod` are listed.
4. Use the browser to locate the file you want to translate, then click *OK*.

   The new `.dml` file is created and added to the directory. Resulting messages or warnings display in a text window.

To translate an IBIS model file to DML format using the command line:

➤ Enter the ibis2signoise command in the following syntax:

```
ibis2signoise [-options] in=infile [out=outfile]
```

`where infile` is the name of the file to translate and `outfile` is the device model library output file to create from the input data. The default is to create `infile.dml` when `outfile` is not specified.

Available options for this command are listed in the following table.

Ibis2signoise Command Options

Option	Function
-u	Creates unique model names. This is the default.
-nu	Leaves model names unchanged.
-ebdcomp	For EBD (Electronic Board Description) files, creates an IbisDevice with a PackageModel instead of a BoardModel.
-serswcomp	Creates an ESpiceDevice component for each Series_switch model, in addition to the IbisDevice that contains one or more Series_switch models.
-i	Does not pass the input file through ibischk4. The default is to pass the input file through ibischk4 before translation.
-d	Does not pass the output file through dmlcheck. The default is to pass the output file through dmlcheck after translation and save the results.
-bufferdelay	Calculates buffer delay for each IbisDevice pin.
-curvedir dir	Creates a directory in which a SigWave-viewable waveform file is created for each VI curve and VT curve. Waveforms display original typical/minimum/maximum and fixed curves. The waveforms are created only if the output of ibis2signoise passes dmlcheck.
-em	Forces a language setting inthe MacroModel section of the file. You can set a tag for the following simulators: ■ Tlsim ■ HSpice ■ Spectre The option immediately overwrites all other external model language files in the IBIS model. In cases where an IBIS model contains multiple external models, the `-em` designation will overwrite all of them.
-version	Displays the version of the software and exits.
-icm	Indicates that the input file is formatted as an InterConnect Model
-icmpkg	Indicates that the input is an ICM file and should output a package model.

Ibis2signoise Command Options

Option	Function
-icmcon	Indicates that the input file isan ICM file and should output a connector model.

The ibis2signoise utility reads the input file. If the ibischk4 program is in your path and you do not specify the -i option, ibis2signoise runs ibischk4 to verify that the input file is a valid IBIS model. The ibis2signoise utility reports any errors detected by ibischk4 and continues. The ibischk4 program is included in the standard cds_install directory.

The ibis2signoise utility then creates the output DML file, and passes this file to the dmlcheck utility unless you specify the -d option. If no warnings or errors are found, then the output file is replaced with the cleaned up file produced by dmlcheck.

If errors are found during any of these three phases, the output file is renamed with the additional .txt extension. In this case the output file should not be used, but it can be examined to diagnose problems.

To translate a Quad model file to DML format using the command line:

➤ Enter the quad2signoise command in the following syntax:

```
quad2signoise [-options] in=infile [out=outfile]
```

where infile is the name of the file to translate and outfile is the device model library output file to create from the input data. The default is to create infile.dml when outfile is not speciifed.

Available options for this command are listed in the following table.

.

Quad2signoise Command Options

Option	Function
-curvedir *dir*	Creates a directory in which a SigWave waveform file is created for each VI curve and VT curve.
-d	Does not pass the output file through *dmlcheck*. The default is to pass the output file through *dmlcheck* after translation and save the results.
-version	Displays the version of the software and exits.

If there is a .tlb file associated with your .mod file, make sure that there is an include statement for the .tbl file in the .mod file. For example, part.mod must have an include part.tlb statement at the beginning.

The `quad2signoise` utility then creates the output DML file, and passes this file to the `dmlcheck` utility unless you specify the `-d` option. If no warnings or errors are found, then the output file is replaced with the cleaned up file produced by dmlcheck.

If errors are found during either of these two phases, the output file is renamed with the additional `.txt` extension. It should not be used in this case, but it can be examined to diagnose problems.

IBIS to DML

Translating EBD files

By default, when `ibis2signoise` translates an IBIS Electronic Board Description (EBD) file, it produces a BoardModel for each EBD file encountered. You can use the resulting BoardModel in Allegro SI as one of the designs in a system configuration.

During translation `ibis2signoise` loads and translates all `.ibs` files listed as required in the `[Reference Designator Map]` section of the EBD file. The BoardModel, as well as all IbisDevice and IbisIOCell models from all input files, are translated into the single output `.dml` file.

When you include the `-ebdcomp` option during translation, `ibis2signoise` models the EBD file as a component by producing an IbisDevice with an associated PackageModel rather than a BoardModel.

- The IbisDevice models the single component internal to the EBD that has driver pins. If the EBD includes either multiple components with drivers or no components with drivers, `ibis2signoise` will not translate the EBD with the `-ebdcomp` option.
- The PackageModel includes SingleLineCircuits. It is created and assigned to the IbisDevice to model the internal interconnect within the EBD.

You can use the IbisDevice as a pluggable component in both Allegro SI and SigXplorer. In this case a system configuration is not required.

Translating Series_switch Models

By default, when `ibis2signoise` translates an IBIS file containing Series_switch models, it produces an IbisDevice component with an associated PackageModel. You can use the resulting IbisDevice as a pluggable component in Allegro SI. Allegro SI recognizes the series connection and accordingly simulates the connected nets as an xnet. However, Series_switch pins added into SigXplorer will ignore the series connection.

To study the operation of Series_switch models in SigXplorer, run `ibis2signoise` with the `-serswcomp` option. A two-pin ESpiceDevice component model is created for each Series_switch model found in the IBIS file. Each model contains a SPICE `subckt` for one switch. You can add the resulting Series_switch model into SigXplorer to study the behavior of the switch, but the model is probably not useful as a model to be assigned to a design component.

The Series_switch SPICE sub-circuits reside in the PackageModel associated with IbisDevice components, and in the ESpiceDevice component, if the `-serswcomp` option is used.

The switch is initially in the ON state. You can modify the `subckt` to simulate the OFF state behavior, or to simulate dynamic ON and OFF switching by text editing the PackageModel. The following code example taken from a Series_switch IBIS model shows the voltage source that let's you turn the switch on and off:

```
...
.subckt subcktName_SERIES 1 2
R1 1 2 1e+06
* Voltage source for controlling Series_switch elements
* Set sersw_gate to 0V for Off mode or 5 for On mode
* Or, attach sersw_gate to a pin for external control
Vsersw_gate sersw_gate 0 5
* [On]
* [Series MOSFET] Vds = 1.000000
* This current source is controlled by Vgs and Vds
GDS_ON0 1 2 PWL 1 2 sersw_gate 2
DATAPOINTS VV
-3.000000 0
-2.000000 0
-1.000000 -1
0 0
1.000000 1
2.000000 0
3.000000 0
END VV
DATAPOINTS VI
0 0
0.1 -0.00061171
0.15 -0.000602017
0.2 -0.000592324
0.25 -0.000594392
```

It is possible, for example, to have the switch turn on or off every 100ns:

```
Vsersw_gate sersw_gate 0 PULSE(0 5 0 5n 5n 45n 100n)
```

If you run the simulation with a fixed duration of 500ns, you will see the effects of switch-on and switch-off on the incoming pulse data.

Multilingual External Model Support

The `ibis2signoise` translation utility supports the [External Model] keyword initiated in IBIS 4.1. The Language parameter in [External Model] is translated differently according to the model type being translated:

- Spice to DML MacroModel

  Behavioral Spice models default to Tlsim (see Example). This will be the case whenever a Language tag is not specified. In situations that you think warrant other simulator languages (HSpice or Spectre), enter the `-em` option in `ibis2signoise`.

  Example 8-1 Spice to DML MacroModel

  ```
  [External Model]
  Language SPICE
  ```
  DML: (Language Tlsim) or No Language tag needed for Tlsim (default)

- Verilog-AMS to DML MacroModel

  The Language tag in the DML IOCell MacroModel translates to Spectre in Verilog-AMS models.

  Example 8-2 Verilog-AMS to DML MacroModel

  ```
  [External Model]
  Language Verilog-AMS
  ```
  DML: (Language Spectre)

- True Differential Models in Spice and Verilog-AMS to DML MacroModel

  The Language tag in the DML 8-terminal IOCell MacroModel translates to the Tlsim default in true differential models. To specify a different Language selection, enter the `-em` option in `ibis2signoise`.

- Non-standard to DML MacroModel

  The Language tag in the DML IOCell MacroModel translates to (that is, matches) either HSpice, Spectre, or Tlsim in non-standard language models. These situations apply except in instances where you enter the `-em` option in `ibis2signoise`.

The `ibis2signoise` translation utility supports only the languages described above (VHDL-AMS is not supported). Attempts to translate non-supported languages generate an error message and terminate the translation process.

Viewing VI and VT Curve Waveforms

Use the `-curvedir` option to have ibis2signoise create waveforms for each VI curve and VT curve found in the translated IBIS file. The waveforms are produced by `dmlcheck`, so they will not be produced if you use the `-d` option with `ibis2signoise`. If the curve directory does not exist, it is created.

One waveform file is created for each VI curve and VT curve in the translated IBIS file. Waveform filenames have the form `IOCellName_CurveName.sim`.

Each waveform file contains 6 waveforms — the original minimum, typical, and maximum curves and the repaired minimum, typical, and maximum curves. Use SigWave to view the waveform files.

In cases where `ibis2signoise` has modified a curve to fix some problem, the change should be apparent when you view the curve waveforms. The fixed waveforms are identical to the original waveforms for unmodified curves.

IBIS to DML Translation Rules

- The `ibis2signoise` utility translates IOCell and device data from the following formats to DML:
  - 1.1
  - 2.1
  - 3.0
  - 3.1
  - 3.2
- Translation is case independent. The `ibis2signoise` utility preserves upper and lowercase for names and strings. You do not have to check the case of keywords, numeric suffixes, and other syntax elements because they do not require upper or lowercase for translation.
- Keywords that consist of two words (such as *Voltage Range*) must have a single space character between the words. The `ibis2signoise` translator does not recognize two-word keywords if the words are separated by anything other than a single space.

- You can include scaling suffixes for numerical quantities. The valid scaling suffixes and the values they specify are shown in the next table.

Ibis2signoise Scaling Suffixes

Suffix	Scaling Value
F	1.00E-15
P	1.00E-12
N	1.00E-09
U	1.00E-06
M	1.00E-03
K	1.00E+03
X	1.00E+06
MEG	1.00E+06
G	1.00E+09
T	1.00E+12

The following table shows suffixes that are valid but have no effect either in translation or simulation.

Other Valid Ibis2signoise Scaling Suffixes

A	S
AMP	V
DEG	OHM
H	

- The suffix *F* is a special case. When preceded by another scaling prefix, such as *pF* for pico Farads, the translator sees the *F* as a unit of measurement and not a scaling suffix. When not preceded by another suffix, such as *FA* for femto Amp, the translator sees the *F* as a scaling suffix.

  All other suffixes are invalid and result in a syntax error.

Warning and Error Messages from ibis2signoise

The `ibis2signoise` translation utility displays warning and error messages that are generated during the translation. These messages are displayed in:

- an `outfile.temp` file if the utility was invoked from the Library Browser dialog box.

-or-

- the window, which contains the command line from where you invoked the utility.

The significance of the line numbers appearing in the messages depends on which phase of the translation they appear in. Line numbers in the `ibischk4` phase and in the IBIS-to-DML conversion phase refer to lines in the input IBIS file. Line numbers in the `dmlcheck` phase refer to lines in the output DML file. If `dmlcheck` finds no errors, the input DML file is replaced with a new output DML file that may have lines inserted and deleted relative to the original input file.

To investigate warning messages by line number, you might need to run `ibis2signoise` with the `-d` option to obtain a DML file containing the line numbers to which `dmlcheck` is referring. For details on warning and error messages, their probable cause and suggested solution, see "Warning Messages from ibis2signoise" on page 176 and "Error Messages from ibis2signoise" on page 180.

Warning Messages from ibis2signoise

```
WARNING @line line_number – BOTH Vinl and Vinh must be specified, or neither
```

Probable Cause: The IBIS file omits a Vinl or Vinh statement or there is a syntax error in one of these statements.

Suggested Solution: Look for the Vinl and Vinh statements beneath the line specified in the warning message.

```
WARNING@ line line_number – Duplicate section_name section
```

Probable Cause: In the IBIS file an IOCell model contains more than one of the specified sections. Possible duplicated sections include all the V/I curves, ramps, and voltage ranges.

Suggested Solution: Look for a duplicate of the specified section above the specified line number in the IBIS file and rename or remove one of these sections.

```
WARNING@ line line_number – Duplicate component name: component_name
```

Probable Cause: A component name was used previously in the IBIS file.

Suggested Solution: Look in the IBIS file at the specified line number. The component name at that line number was used previously in the file. Determine which of the two

components you want translated to an DML device model and delete or rename the other one.

`WARNING @line line_number – Duplicate pin name`

Probable Cause: A pin section contains entries for two pins with the same name.

Suggested Solution: At the specified line number, in a pin section, find the pin name and then look above this line number for a pin with the same name. Rename or delete one of these two pins.

`WARNING @line line_number – Duplicate model name`

Probable Cause: The IBIS file contains models for two IOCell models with the same name.

Suggested Solution: The specified line contains a model name for an I/O cell. Look above this line for an IOCell model with the same name. Rename or delete one of these models.

`WARNING @line line_number – Duplicate model parameter parameter_name`

Probable Cause: A model in the IBIS file contains more than one entry of a model parameter. Model parameters include *Model_type*, *Polarity*, *Enable*, *C_comp*, *Vinl*, and *Vinh*.

Suggested Solution: The specified line contains a model parameter for a model. Look above this line for another entry of this model parameter and delete one of the entries of this parameter

`WARNING @line line_number – Duplicate [Model Spec] sub-parameter <PARAM>`

Probable Cause: One of the following parameters has appeared twice within one `[Model Spec]` section: *Vinh*, *Vinl*, *Vinh+*, *Vinl+*, *Vinh-*, *Vinl-*, *S_overshoot_high*, *S_overshoot_low*, *D_overshoot_high*, *D_overshoot_low*, *D_overshoot_time*, *Pulse_high*, *Pulse_low*, *Pulse_time*, *Vmeas*.

Suggested Solution: The specified line contains a sub-parameter for a model. Look above this line for another entry of this sub-parameter and delete one of the entries of this sub-parameter.

`WARNING @line line_number – Lexical error:`

Probable Cause: The IBIS file contains a syntax error at the specified line. Possible lexical errors include misspelled keywords.

Suggested Solution: Edit the specified line in the IBIS file.

`WARNING @line line_number – Model selector entry <MODEL> repeated`

Probable Cause: A model in the IBIS file contains a duplicated model selector name.

Suggested Solution: The specified line contains a duplicated model selector name. Look above this line for another model selector name and delete one of them.

`WARNING @line line_number – PackageModel <MODEL> removed from component <MODEL>`

Probable Cause: One of the internal components of an EBD has a PackageModel assigned to it, and the `-ebdcomp` option has been given. Since this option produces an IbisDevice component for the EBD module as a whole, only the top level IbisDevice may have a PackageModel. Internal components are regarded as "bare die" in this case.

Suggested Solution: Verify that the referenced PackageModel is not required for EBD files translated to IbisDevice with the associated PackageModel.

`WARNING @line line_number – Repeated C_pkg`

Probable Cause: A model in the IBIS file contains two C_pkg specifications.

Suggested Solution: The specified line contains a C_pkg specification. Look above this line for another C_pkg specification and delete one of them

`WARNING @line line_number – Repeated L_pkg`

Probable Cause: A model in the IBIS file contains two L_pkg specifications.

Suggested Solution: The specified line contains an L_pkg specification. Look above this line for another L_pkg specification and delete one of them.

`WARNING @line line_number – Repeated R_pkg`

Probable Cause: A model in the IBIS file contains two R_pkg specifications.

Suggested Solution: The specified line contains an R_pkg specification. Look above this line for another R_pkg specification and delete one of them.

`WARNING @line line_number – Repeated ramp keyword: keyword`

Probable Causes:

- If the specified ramp keyword is Fall, the Ramp section at the specified line number contains more than one dV/dt_f specification.
- If the specified ramp keyword is Rise, the Ramp section at the specified line number contains more than one dV/dt_r specification.

Suggested Solution: The specified line contains a dV/dt_r or dV/dt_f specification. Look above this line for another dV/dt_r or dV/dt_f specification and then delete one of them.

`WARNING @line line_number – Too many dV/dt pairs (should be only 3)`

Probable Cause: A dV/dt_f or dV/dt_r specification contains more than three pairs. A dV/dt_f or dV/dt_r specification should contain three pairs of values. Each pair specifies the change of voltage over the change in time. The three pairs specify minimum, typical, and maximum ramp values.

Suggested Solution: Edit the specified line number so that the dV/dt_f or dV/dt_r specification contains only three pairs of values.

`WARNING @line line_number – Unrecognized Enable type character_string. Defaulting to Active-High.`

Probable Cause: An IOCell model has an invalid enable type. IOCell models can have only two types of enables: Active-High and Active-Low. The `ibis2signoise` translator substitutes Active-High for the invalid enable type.

Suggested Solution: If you do not want SigNoise to use the Active-High enable type, edit the specified line so that it contains the Active-Low enable type.

```
WARNING @ line line_number – Unrecognized model type: character_string, Defaulting to Load
```

Probable Cause: A model type in Model_type specification is invalid. The valid model types are Output, I/O, 3-state, and Load. The `ibis2signoise` translator substitutes Load for the invalid model type.

Suggested Solution: If you do not want SigNoise to use the Load model type, edit the specified line so that it contains another valid model type.

```
WARNING @line line_number – Unrecognized Polarity character_string. Defaulting to Non-Inverting.
```

Probable Cause: An IOCell model has an invalid polarity specification. IOCell models can have only two types of polarities: Inverting and Non-Inverting. The ibis2signoise translator substitutes Non-Inverting for the invalid polarity specification.

Suggested Solution: If you do not want SigNoise to use the Non-Inverting polarity, edit the specified line so that it contains the Inverting polarity specification.

```
WARNING @line line_number – Pentium_processor(735|90,815|100) will be written as
Pentium_processor_73590_815_100
```

Probable Cause: The `ibis2signoise` translator replaces all illegal model name characters with underscores in the final SigNoise model output.

```
WARNING @line line_number – [C Series] will be used in both [On] and [Off] states
WARNING @line line_number – [L Series] will be used in both [On] and [Off] states
WARNING @line line_number – [Lc Series] will be used in both [On] and [Off] states
WARNING @line line_number – [R Series] will be used in both [On] and [Off] states
WARNING @line line_number – [Rc Series] will be used in both [On] and [Off] states
WARNING @line line_number – [Series Current] will be used only in the [Off] state
WARNING @line line_number – [Series MOSFET] will be used only in the [On] state.
```

Probable Cause: Separate sets of these `[Model Spec]` keywords can be specified for the [On] state and [Off] state of an IBIS Series_switch. To support dynamic modeling of the switch, Allegro SI allows only a single topology. A single circuit is constructed containing all series elements. `[Series MOSFET]` elements are enabled only in the [On] state, and `[Series Current]` elements are enabled only in the [Off] state.

Suggested Solution: Verify that the switch behavior will be as intended given these conditions. Possibly reorganize the Series_switch elements in the IBIS file accordingly. For example, if a 10 Ohm resistor is used in the [On] state and a 1MOhm resistor is used in the [Off] state, replace the [On] state resistor with an equivalent `[Series MOSFET]`.

Error Messages from ibis2signoise

ERROR @ line line_number – Syntax error The lexical token was <*character-string*>.

Probable Cause: The IBIS file contains a syntax error at the specified line. Possible lexical errors include misspelled keywords.

Suggested Solution: Edit the specified line in the IBIS file.

ERROR @line line_number – Rise_on_dly may not be negative
ERROR @line line_number – Fall_on_dly may not be negative
ERROR @line line_number – Rise_off_dly may not be negative
ERROR @line line_number – Fall_off_dly may not be negative
ERROR @line line_number – Rise_off_dly must be greater than Rise_on_dly, if set
ERROR @line line_number – Fall_off_dly must be greater than Fall_on_dly, if set

Probable Cause: Incorrect time values in a `[Driver Schedule]` section.

Suggested Solution: Edit the time values in the IBIS file.

ERROR @line line_number – IbisDevice 'MODEL' has no PackageModel SeriesPinMapping skipped

Probable Cause: An IBIS `[Component]` that has `[Series Pin Mapping]` references a `[Package Model]`, but no definition of the `[Package Model]` is found.

Suggested Solution: Either remove the `[Package Model]` reference, or include the `[Package Model]` definition in the same IBIS file as the IBIS `[Component]`.

ERROR @line line_number – No SubCircuits in 'MODEL' CircuitModels – SeriesPinMapping skipped.
ERROR @line line_number – PackageModel 'MODEL' has no CircuitModels –SeriesPinMapping skipped.

Probable Cause: The `[Package Model]` referenced by an IBIS `[Component]` may be a matrix style model. To implement `[Series Pin Mapping]` a CircuitModels PackageModel with a valid SubCircuits section is required.

Suggested Solution: Correct the PackageModel so that it has a valid CircuitModels section with SubCircuits. Optionally, the PackageModel reference may be removed from the IBIS `[Component]` that has `[Series Pin Mapping]`. In this case a PackageModel contain the `[Series Pin Mapping]` elements will be created automatically.

ERROR @line line_number – No data for package model.

Probable Cause: The dimension of an IBIS `[Define Package Model]` is absent.

Suggested Solution: Edit the IBIS file and add the correct dimension data.

ERROR @line line_number – No definition found for SeriesPin model. 'MODEL'

Probable Cause: A `[Series Pin]` section references a Series model, but the named model is not found in the IBIS file.

Suggested Solution: Add the IBIS Series `[Model]` to the file containing the `n [Component]` with `[Series Pin]` section.

ERROR @line line_number – PackageModel 'MODEL' for IbisDevice 'MODEL' is not found

Probable Cause: The IBIS `[Component]` references a `[Package Model]` for which no corresponding `[Define Package Model]` is found.

Suggested Solution: Add the `[Define Package Model]` to the same IBIS file that contains the referencing `[Component]`.

```
ERROR – Syntax error, position unknown
```

Probable Cause: You should not see this error message. The `ibis2signoise` translator has encountered an unanticipated problem.

Suggested Solution: Contact Cadence Customer Support.

```
ERROR @line line_number – Unable to create EBD <MODEL> component model because component <MODEL> is not
found.
```

Probable Cause: An internal `[Component]` referenced in an EBD is not found.

It is required if the `-ebdcomp` option is given, so that the EBD can be implemented as a PackageModel associated with the primary internal IbisDevice component.

Suggested Solution: Place the IBIS file for the `[Component]` in the same directory as the EBD file. Make sure the component IBIS file has the filename as stated in the EBD file.

```
ERROR @line line_number – Unable to create PackageModel for IbisDevice 'MODEL'
```

Probable Cause: Unexpected error.

Suggested Solution: Contact Cadence support.

```
ERROR @line line_number – Unable to identify component name for EBD <MODEL_PKG> component model.
```

Probable Cause: The EBD does not contain exactly one internal component with driving pins. The -ebdcomp option requires that the EBD contain one active component that has drivers. All other internal components must be passive, with no pins that can drive an output.

Suggested Solution: Change the `[Model]` assignments of pins on internal components that are not intended to be active drivers. If the EBD correctly contains multiple driving components, then the `-ebdcomp` option cannot be used.

```
ERROR @line line_number – Unrecognized [Model Spec] sub-parameter <PARAM>
```

Probable Cause: A sub-parameter appearing after a `[Model Spec]` is not one of: *`Vinh`*, *`Vinl`*, *`Vinh+`*, *`Vinl+`*, *`Vinh-`*, *`Vinl-`*, *`S_overshoot_high`*, *`S_overshoot_low`*, *`D_overshoot_high`*, *`D_overshoot_low`*, *`D_overshoot_time`*, *`Pulse_high`*, *`Pulse_low`*, *`Pulse_time`*, or *`Vmeas`*. There may be a misspelling.

Suggested Solution: Check your spelling and edit the IBIS file at that line.

```
ERROR @line line_number – Use of [LC Series] with no [C Series]
ERROR @line line_number – Use of [RC Series] with no [C Series]
ERROR @line line_number – Use of [Rl Series] with no [L Series]
```

Probable Cause: A `[C Series]` or `[L Series]` section is missing from the IBIS file.

Suggested Solution: Add the required primary series element.

```
ERROR @line line_number – [Number of Pins] missing from package model.
```

Probable Cause: A package model is not followed by a `[Number Of Pins]` line.

Suggested Solution: Add the required line, or move it so that it follows the beginning of the package model.

```
ERROR @line line_number – [C Series] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [Driver Schedule] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [L Series] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [LC Series] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [Model Spec] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [Number of Sections] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [Off] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [On] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [R Series] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [RC Series] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [RL Series] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [Series Current] not allowed in pre-IBIS 3.0 file
ERROR @line line_number – [Series MOSFET] not allowed in pre-IBIS 3.0 file
```

Probable Cause: The `[IBIS Version]` of the file is incorrect.

Suggested Solution: Change `[IBIS Version]` to 3.2.

QUAD to DML

Viewing VI and VT Curve Waveforms

Use the `-curvedir` option to have `quad2signoise` create waveforms for each VI curve and VT curve found in the translated IBIS file. The waveforms are produced by `dmlcheck`, so they will not be produced if you use the `-d` option with `quad2signoise`. If the curve directory does not exist, it is created.

One waveform file is created for each VI curve and VT curve in the translated IBIS file. Waveform filenames have the form `IOCellName_CurveName.sim`.

Each waveform file contains 6 waveforms — the original minimum, typical, and maximum curves and the repaired minimum, typical, and maximum curves.

Use SigWave to view the waveform files.

In cases where `quad2signoise` has modified a curve to fix some problem, the change should be apparent when you view the curve waveforms. The fixed waveforms are identical to the original waveforms for unmodified curves.

Warning and Error Messages from quad2signoise

The `quad2signoise` translation utility displays warning and error messages that are generated during the translation.

If warnings or errors are found, they are displayed in the shell window that you used to start the application. The line numbers refer to lines in `outfile.temp`. An output file is not created.

If there are only warning messages, the output file is generated. The reported problems exist, but did not prevent the output file from being created. If there are error messages, the output file cannot be generated until the reported problems are fixed. For details on warning and error messages, their probable cause and suggested solution, see “Warning Messages from quad2signoise” on page 183 and “Error Messages from the quad2signoise” on page 184.

Warning Messages from quad2signoise

```
WARNING: Unable to open the log file log_filename for writing. quad2signoise.log will be taken as the
default log file.
```

Probable Cause: A permission or some other system problem prevents the `quad2signoise` translator from writing the log file that you specified. The translator writes a log file named `quad2signoise.log` in the current directory.

Suggested Solution: Check permissions for the path you specified for the log file.

```
WARNING: Unable to open the log file quad2signoise.log for writing. Log will be displayed on the standard output.
```

Probable Cause: A permission or some other system problem prevents the `quad2signoise` translator from writing the `quad2signoise.log` file in the current directory.

Warning and error messages appear only in the window the utility is running in.

Suggested Solution: Check permissions for the current directory.

Error Messages from the quad2signoise

```
ERROR: Unable to open file output_dml_filename for writing.
```

Probable Cause: A permission or some other system problem prevents the `quad2signoise` translator from writing the DML device model library.

Suggested Solution: Check permissions for the path to the current directory or the path you specified for the output file.

```
ERROR: Unable to allocate memory.
```

Probable Cause: The translator does not have enough memory.

Suggested Solution: Check your system.

```
ERROR: Unable to open file input_QUAD_filename for reading.
```

Probable Cause: A permission or some other system problem prevents the `quad2signoise` translator from reading the QUAD file.

Suggested Solution: Check permissions for the path to the directory that contains the Quad File

Translating ESpice Files to Generic Spice Files

You can use the `spc2spc` utility to translate a Cadence proprietary Spice (or ESpice) input file to a generic SPICE input file compatible with Spice2G, Spice3, Cadence SPECTRE or HSpice. Using `spc2spc`, you can translate ESpice input file to a generic Spice format you can use for benchmarking.

To translate a ESpice file to generic Spice:

➤ Enter the `spc2spc` command at the operating system command line in the following syntax:

```
Spc2spc [options] <input file>
```

The usual input file will be `main.spc`.

Available options for this command are listed in the following tables.

Spc2spc Output Options

Option	Function
-mapOut=<*filename***>**	Create a connectivity map file with the specified name.
-spectreOut=<*filename***>**	Translate to Spectre language format.
-spiceOut=<*filename***>**	Translate to Spice language format.

Spc2spc General Options

Option	Function
-commentInfo	Copy comments from source file to output file.
-dir= <*directory***>**	Create sub-directory <*directory*> into which all output files will be written. This is particularly useful when the welement option is enabled, since a large number of output files may be created.
-fileInfo	Add comments to the output file indicating the name of the source file and the line number for each translated statement.
-flatten	Flatten the circuit so no sub-circuits remain.
-h	Send help text to stdout.

Spc2spc General Options

Option	Function
-nelement	Do not translate ESpice transmission line n elements. The default action is to translate ESpice n elements into resistor, capacitor, inductor (RLC) ladder networks.
-rename= **<***node,elem,subckt***>**	Rename nodes, elements or sub-circuits. The placeholder text, *node*, *elem* and *subckt* must be replaced by one of the following format codes: alphanum- Truncate name to 8 characters and add a numeric uniquifier. asis- Do not change the name. clean - Change non-alphanumeric characters to '_' numeric- Change identifier to a number.
-hs	Name conversions apply for *Hspice.print* character limitations. *name.tab* will be created as a name lookup table.
-sourceInfo	Include original text of source file as comments in the output file.
-version	Send software version text to stdout.
-welement	Translate finite length ESpice transmission line n elements into w elements. This option creates a RLGC definition file for each distinct n element type.
-wminlen= **<***length***>**	Modify processing of w element option so that only transmission lines greater than the specified length are translated to w elements. Shorter transmission lines will be converted into RLC ladder networks. The length may be specified with units (for example, 12mils). If no units are supplied, the length is assumed to be in meters.

Spc2spc General Options

Option	Function

-hspctr	Special option for Spice/Spectre Mixed Language Spectre simulator. Use this to translate *most* of the netlist into Hspice syntax. The following exceptions apply:
	■ Sections enclosed between special control statements are passed literally; for example:
	`BEGIN_LITERAL_SPECTRE`
	`simulator lang=spectre`
	`R1 (1 2) resistor 50`
	`simulator lang=spice`
	`R2 12 50`
	`*spectre: R3 1 2 resistor 150`
	`.....`
	`END_LITERAL_SPECTRE`
	■ S-Parameter data files in native Spectre format; for example:
	`simulator lang=spice`
	`X1 (1 0 2 0 3 0) sp_data`
	`.MODEL sp_data NPORT file=”3p.data`
	`*interp=rational`
	This option also:
	Forces the `-rename clean` option for element and subcircuit names
	Generates a print node conversion table
	Changes the default output option to `-spiceOut=main_gen.spectre`

Spc2spc Options for Tuning RLC Ladder Generation

Option	Function
-s	Convert single-line transmission line n elements into Spice t elements.

Spc2spc Options for Tuning RLC Ladder Generation

Option	Function
-t= <*Rise/Fall Time in ps*>	Set the minimum rise or fall time in picoseconds. RLC ladder networks are only accurate in modeling transmission lines up to a limited frequency. The frequency limit can be increased by increasing the number of ladder elements, but this will slow simulation speed. When this option is supplied, ladder network generation will be optimized to work with edge rates up to that specified.
-x= <*segment delay in ps*>	Set the maximum RLC segment delay in picoseconds. This option can be used instead of the -t option to ensure there are sufficient ladder sections to accurately simulate up to the frequency of interest.

The `spc2spc` translator reads the specified input file, usually `main.spc`. It will also read files *#included* into this file. In the case of a standard `main.spc`, it *#includes* the complete ESpice circuit hierarchy for a test case. This includes the components file (`comps.spc`), the interconnect file (`intercon.spc`), the stimulus file (`stimulus.spc`), and, when transmission files are present, the RLGC model files (`dlink_RLGC.inc`, `comps_RLGC.inc`, and `ntl_RLGC.inc`).

After translation, the complete ESpice file hierarchy is combined into one Spice input deck, `main_gen.spc`.

Spc2spc Translation Rules

- `spc2spc` does not translate behavioral models. Upon encountering a behavioral model, `spc2spc` comments out the call to the behavioral model and describes, within comments, the node connections for the behavioral model.
- If behavioral models are present in your `comps.spc` file, you must make the appropriate model/stimulus connections to the behavioral model's IO nodes. In most cases you can accomplish this by connecting the stimulus sub-circuit node to the behavioral model's former IO node directly.
- `spc2spc` models transmission lines as distributed lumped element models with the `-s` option, or in the case of single-line transmission lines, as T-models. By default, all transmission lines are modeled as distributed RLC element lines. Coupled transmission

lines are modeled with mutual capacitances and inductor coupling coefficients (k-factors).

- `spc2spc` determines the granularity of the distributed, lumped sections based on the modal propagation velocities in the transmission lines and the minimum transmission tile from the stimulus file. The propagation delay for a section is constrained to be 1/20th of the minimum transmission time in the system. This limits the lumped element approximation error to 4%. You can use a command line option (`-x=tau`) to override the default per-section delay to any non-zero value. You can use another command line option (`-t=tr`) to override the minimum rise and fall time used in the lumped section delay calculations.
- `spc2spc` breaks inductive resistors (resistive inductors) into two series elements. It keeps skin-effect resistors as parallel elements to the series R-L elements when they are present.

A

PackagedDevice Examples

IbisDevice Template

```
("DMLfilename.dml"
   (PackagedDevice
    ("<Enter name of model here>"
     (ModelVersion "<Enter text here>")     ; optional but recommended
     (ModelSource "<Enter text here>")      ; optional but recommended
     (ModelDate "<Enter text here>")        ; optional but recommended
     (Manufacturer "<Enter text here>")     ; optional but recommended
     (EstimatedPinParasitics
       (R
         (minimum "<Enter value here>")
         (typical "<Enter value here>")
         (maximum "<Enter value here>") )
       (L
         (minimum "<Enter value here>")
         (typical "<Enter value here>")
         (maximum "<Enter value here>") )
       (C
         (minimum "<Enter value here>")
         (typical "<Enter value here>")
         (maximum "<Enter value here>") ) )
     (ModelSelector
       (<Enter symbolic IOCell name1 here>
          (<Enter IOCell nameA here> "<Enter description string here>")
          (<Enter IOCell nameB here> "<Enter description string here>") )
       (<Enter symbolic IOCell name2 here>
          (<Enter IOCell nameC here> "<Enter description string here>")
          (<Enter IOCell nameD here> "<Enter description string here>") ))
     (ModelSelectorDefault
          (<Enter symbolic IOCell name1 here> "<Enter default IOCell nameA here>")
          (<Enter symbolic IOCell name2 here> "<Enter default IOCell nameC here>"))
     (IbisPinMap
      (<Enter pin number here>
        (signal "<Enter signal name here>")
        (bus "<Enter bus name here>")
        (signal_model "<Enter signal model name here>")
        (ground_bus "<Enter ground bus name here>")
        (ground_clamp_bus "<Enter ground clamp bus name here>")
        (ground_pin "<Enter ground pin name here>")
```

```
        (power_bus "<Enter power bus name here>")
        (power_clamp_bus "<Enter clamp bus name here>")
        (power_pin "<Enter power pin name here>")
        (R "<Enter value here>")
        (L "<Enter value here>")
        (C "<Enter value here>")
        (WireNumber "<Enter pin number here>")
        (BufferDelay
          (<Enter the name of the IOCell model here>
            (Rise
               (minimum "<Enter value here>")
               (typical "<Enter value here>")
               (maximum "<Enter value here>") )
            (Fall
               (minimum "<Enter value here>")
               (typical "<Enter value here>")
               (maximum "<Enter value here>") ) ) ) )
     (DiffPair
      (<Enter pin number here>
      (InversePin <Enter pin number here> )
      (LogicThresholds
        (<Enter Input or Output>
           (High
               (minimum "<Enter value here>")
               (typical "<Enter value here>")
               (maximum "<Enter value here>")
           (Low
               (minimum "<Enter value here>")
               (typical "<Enter value here>")
               (maximum "<Enter value here>") ) ) )
      (LaunchDelay
               (minimum "<Enter value here>")
               (typical "<Enter value here>")
               (maximum "<Enter value here>") ) )
     )
     )
)
```

DML File of GUI Example (Figures 2-1 and 2-2) for IbisDevice

```
("exampleA2.dml"
    (PackagedDevice
     ("p14u1_diffPair"
         ("EstimatedPinParasitics"
          ("R"
              ("typical" "250m")
              ("minimum" "225m")
              ("maximum" "275m")
          )
          ("L"
```

```
        ("typical" "15n")
        ("minimum" "12n")
        ("maximum" "18n")
    )
    ("C"
        ("typical" "18p")
        ("minimum" "15p")
        ("maximum" "20p")
    )
   )
   ("IbisPinMap"
    ("14"
        ("signal" "EN1#")
        ("signal_model" "CDSDefaultInput")
        ("WireNumber" "14")
        ("ground_bus" "gndbus")
        ("power_bus" "pwrbus")
        ("R" "220m")
        ("L" "5.2n")
        ("C" "2.4p")
    )
    ("13"
        ("signal" "RAS0-")
        ("signal_model" "CDSDefaultOutput")
        ("ground_bus" "gndbus")
        ("ground_clamp_bus" "gndbus")
        ("power_bus" "pwrbus")
        ("power_clamp_bus" "pwrbus")
        ("R" "200m")
        ("L" "5n")
        ("C" "2p")
        ("WireNumber" "13")
        ("BufferDelay"
         ("CDSDefaultOutput"
             ("Rise"
              ("minimum" "6.91381e-10")
              ("typical" "1.37573e-09")
              ("maximum" "1.80825e-09")
             )
             ("Fall"
              ("minimum" "4.57694e-10")
              ("typical" "9.45875e-10")
              ("maximum" "1.23181e-09")
             )
         )
        )
    )
    ("12"
        ("signal" "RAS0+")
        ("signal_model" "CDSDefaultOutput")
        ("WireNumber" "12")
```

```
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "209m")
    ("L" "6n")
    ("C" "2.5p")
    ("BufferDelay"
     ("CDSDefaultOutput"
         ("Rise"
          ("minimum" "6.91354e-10")
          ("typical" "1.37571e-09")
          ("maximum" "1.80822e-09")
         )
         ("Fall"
          ("minimum" "4.57667e-10")
          ("typical" "9.45849e-10")
          ("maximum" "1.23179e-09")
         )
     )
    )
)
("11"
    ("signal" "DOUT1-")
    ("signal_model" "CDSDefaultOutput")
    ("WireNumber" "11")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "190m")
    ("L" "4.5n")
    ("C" "2p")
    ("BufferDelay"
     ("CDSDefaultOutput"
         ("Rise"
          ("minimum" "6.91344e-10")
          ("typical" "1.3757e-09")
          ("maximum" "1.80821e-09")
         )
         ("Fall"
          ("minimum" "4.57657e-10")
          ("typical" "9.4584e-10")
          ("maximum" "1.23178e-09")
         )
     )
    )
)
("10"
    ("signal" "DOUT1+")
    ("signal_model" "CDSDefaultOutput")
    ("WireNumber" "10")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "190m")
```

```
    ("L" "4.5n")
    ("C" "2p")
    ("BufferDelay"
     ("CDSDefaultOutput"
         ("Rise"
          ("minimum" "6.91279e-10")
          ("typical" "1.37564e-09")
          ("maximum" "1.80815e-09")
         )
         ("Fall"
          ("minimum" "4.57593e-10")
          ("typical" "9.4578e-10")
          ("maximum" "1.23172e-09")
         )
     )
    )
)
("9"
    ("signal" "DMG")
    ("signal_model" "CDSDefaultOutput")
    ("WireNumber" "9")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "200m")
    ("L" "4.9n")
    ("C" "2.2p")
    ("BufferDelay"
     ("CDSDefaultOutput"
         ("Rise"
          ("minimum" "6.91373e-10")
          ("typical" "1.37573e-09")
          ("maximum" "1.80824e-09")
         )
         ("Fall"
          ("minimum" "4.57685e-10")
          ("typical" "9.45867e-10")
          ("maximum" "1.23181e-09")
         )
     )
    )
)
("7"
    ("signal" "DIN2+")
    ("signal_model" "CDSDefaultInput")
    ("WireNumber" "7")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "220m")
    ("L" "5.2n")
    ("C" "2.4p")
)
```

```
("6"
    ("signal" "DIN2-")
    ("signal_model" "CDSDefaultInput")
    ("WireNumber" "6")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "200m")
    ("L" "4.9n")
    ("C" "2.2p")
)
("5"
    ("signal" "DIN1+")
    ("signal_model" "CDSDefaultInput")
    ("WireNumber" "5")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "190m")
    ("L" "4.5n")
    ("C" "2p")
)
("4"
    ("signal" "DIN1-")
    ("signal_model" "CDSDefaultInput")
    ("WireNumber" "4")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "190m")
    ("L" "4.5n")
    ("C" "2p")
)
("3"
    ("signal" "DMR")
    ("signal_model" "CDSDefaultInput")
    ("WireNumber" "3")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "190m")
    ("L" "4.5n")
    ("C" "2p")
)
("2"
    ("signal" "CAS0")
    ("signal_model" "CDSDefaultIO")
    ("WireNumber" "2")
    ("ground_bus" "gndbus")
    ("power_bus" "pwrbus")
    ("R" "190m")
    ("L" "4.5n")
    ("C" "2p")
    ("BufferDelay"
     ("CDSDefaultIO"
```

```
          ("Rise"
           ("minimum" "6.31133e-10")
           ("typical" "8.82682e-10")
           ("maximum" "1.29079e-09")
          )
          ("Fall"
           ("minimum" "3.8404e-10")
           ("typical" "5.69679e-10")
           ("maximum" "8.02297e-10")
          )
      )
     )
 )
 ("8"
     ("bus" "gndbus")
     ("signal_model" "GND")
     ("R" "150m")
     ("L" "7n")
     ("C" "3p")
     ("WireNumber" "8")
 )
 ("1"
     ("bus" "pwrbus")
     ("signal_model" "POWER")
     ("R" "175m")
     ("L" "8n")
     ("C" "2p")
     ("WireNumber" "1")
 )
)
("DiffPair"
 ("5"
     ("InversePin" "4")
     ("LogicThresholds"
      ("Input"
          ("High"
           ("minimum" "0")
           ("typical" "60mv")
           ("maximum" "200mV")
          )
          ("Low"
           ("minimum" "-300mV")
           ("typical" "-80mv")
           ("maximum" "0")
          )
      )
     )
 )
 ("7"
     ("InversePin" "6")
     ("LogicThresholds"
```

```
      ("Input"
          ("High"
           ("minimum" "0")
           ("typical" "60mv")
           ("maximum" "200mV")
          )
          ("Low"
           ("minimum" "-300mV")
           ("typical" "-80mv")
           ("maximum" "0")
          )
      )
     )
)
("10"
     ("InversePin" "11")
     ("LogicThresholds"
      ("Output"
          ("High"
           ("minimum" "4.5")
           ("typical" "4.8")
           ("maximum" "5.1")
          )
          ("Low"
           ("minimum" "-5.2")
           ("typical" "-4.8")
           ("maximum" "-4.5")
          )
      )
     )
     ("LaunchDelay"
      ("typical" "0.3ns")
      ("minimum" "-1ns")
      ("maximum" "1ns")
     )
)
("12"
     ("InversePin" "13")
     ("LogicThresholds"
      ("Output"
          ("High"
           ("minimum" "4.5")
           ("typical" "4.8")
           ("maximum" "5.1")
          )
          ("Low"
           ("minimum" "-5.2")
           ("typical" "-4.8")
           ("maximum" "-4.5")
          )
      )
```

```
            )
            ("LaunchDelay"
             ("typical" "0.3ns")
             ("minimum" "-1ns")
             ("maximum" "1ns")
            )
         )
        )
        ("Manufacturer" "TI")
        ("PackageModel" "CDS_Pkg14DIP")
    )
   )
)
```

DML File of GUI Example (Figure 2-3) for IbisDevice

```
("exampleA3.dml"
 (PackagedDevice
   ("74ALS08"
        ("IbisPinMap"
         ("1"
             ("ground_bus" "gndbusGROUND")
             ("ground_pin" "7")
             ("power_bus" "pwrbusVCC")
             ("power_pin" "14")
             ("signal" "A")
             ("signal_model" "Z114_IN")
         )
         ("10"
             ("ground_bus" "gndbusGROUND")
             ("ground_pin" "7")
             ("power_bus" "pwrbusVCC")
             ("power_pin" "14")
             ("signal" "B")
             ("signal_model" "Z114_IN")
         )
         ("11"
             ("ground_bus" "gndbusGROUND")
             ("ground_pin" "7")
             ("power_bus" "pwrbusVCC")
             ("power_pin" "14")
             ("signal" "Y")
             ("signal_model" "Z114_OUT")
         )
         ("12"
             ("ground_bus" "gndbusGROUND")
             ("ground_pin" "7")
             ("power_bus" "pwrbusVCC")
             ("power_pin" "14")
             ("signal" "A")
```

```
    ("signal_model" "Z114_IN")
)
("13"
    ("ground_bus" "gndbusGROUND")
    ("ground_pin" "7")
    ("power_bus" "pwrbusVCC")
    ("power_pin" "14")
    ("signal" "B")
    ("signal_model" "Z114_IN")
)
("14"
    ("bus" "pwrbusVCC")
    ("signal" "VCC")
    ("signal_model" "POWER")
)
("2"
    ("ground_bus" "gndbusGROUND")
    ("ground_pin" "7")
    ("power_bus" "pwrbusVCC")
    ("power_pin" "14")
    ("signal" "B")
    ("signal_model" "Z114_IN")
)
("3"
    ("ground_bus" "gndbusGROUND")
    ("ground_pin" "7")
    ("power_bus" "pwrbusVCC")
    ("power_pin" "14")
    ("signal" "Y")
    ("signal_model" "Z114_OUT")
)
("4"
    ("ground_bus" "gndbusGROUND")
    ("ground_pin" "7")
    ("power_bus" "pwrbusVCC")
    ("power_pin" "14")
    ("signal" "A")
    ("signal_model" "Z114_IN")
)
("5"
    ("ground_bus" "gndbusGROUND")
    ("ground_pin" "7")
    ("power_bus" "pwrbusVCC")
    ("power_pin" "14")
    ("signal" "B")
    ("signal_model" "Z114_IN")
)
("6"
    ("ground_bus" "gndbusGROUND")
    ("ground_pin" "7")
    ("power_bus" "pwrbusVCC")
```

```
            ("power_pin" "14")
            ("signal" "Y")
            ("signal_model" "Z114_OUT")
          )
          ("7"
            ("bus" "gndbusGROUND")
            ("signal" "GROUND")
            ("signal_model" "GND")
          )
          ("8"
            ("ground_bus" "gndbusGROUND")
            ("ground_pin" "7")
            ("power_bus" "pwrbusVCC")
            ("power_pin" "14")
            ("signal" "Y")
            ("signal_model" "Z114_OUT")
          )
          ("9"
            ("ground_bus" "gndbusGROUND")
            ("ground_pin" "7")
            ("power_bus" "pwrbusVCC")
            ("power_pin" "14")
            ("signal" "A")
            ("signal_model" "Z114_IN")
          )
        )
        ("Manufacturer" "TI")
        ("PackageModel" "zaaa14")
        )
     )
    )
)
```

ESpice Template

```
("DMLfilename.dml"
  (PackagedDevice
   ("<Enter name of model here>"
     (ModelVersion "<Enter text here>")     ; optional but recommended
     (ModelSource "<Enter text here>")      ; optional but recommended
     (ModelDate "<Enter text here>")        ; optional but recommended
     (Manufacturer "<Enter text here>")     ; optional but recommended
     (PinConnections
     (<Enter pin number/name here> <Enter pin number/name here>) )
     (ESpice "
     .subckt <Enter name of model here> <Enter pin numbers/names here>
     ")
     )
   )
)
```

DML File of GUI Example (Figure 2-4) for ESpice

```
("exampleA4.dml"
 (PackagedDevice
  ("RPAK2C_8R_SM_10K"
   (ESpice ".subckt RPAK2C_8R_SM_10K 5 1 2 3 4 6 7 8 9
    R1 1 5 10000
    R2 2 5 10000
    R3 3 5 10000
    R4 4 5 10000
    R5 6 5 10000
    R6 7 5 10000
    R7 8 5 10000
    R8 9 5 10000
.ends RPAK2C_8R_SM_10K
" )
   (PinConnections
    (1 5 )
    (2 5 )
    (3 5 )
    (4 5 )
    (6 5 )
    (7 5 )
    (8 5 )
    (9 5 ) ) ) ) )
```

B

PackageModel Examples

PackageModel Template

```
("DMLfilename.dml"
   (PackageModel               ;This could also be <name of package model>
     (ModelVersion "<Enter text here>")
     (ModelSource "<Enter text here>")
     (ModelDate "<Enter text here>")
     (Manufacturer "<Enter text here>")
     (PinNameToNumber
       ("<Enter pin name here>" "Enter pin number here")
       ("<Enter pin name here>" "Enter pin number here") )
     (RLGC
       ("<Enter frequency value here>"
         (R
          (<Enter format of RLGC here>  ; BandedSymmetricMatrix /
                                        ; SymmetricMatrix /
                                        ; SparseSymmetricMatrix
            (band "<Enter B_NUMBER here>")
            (dimension "<Enter D_NUMBER here>")
            (data "<Enter data_values here>") ) )
         (L
          (<Enter format of RLGC here>  ; BandedSymmetricMatrix /
                                        ; SymmetricMatrix /
                                        ; SparseSymmetricMatrix
            (band "<Enter B_NUMBER here>")
            (dimension "<Enter D_NUMBER here>")
            (data "<Enter data_values here>") ) )
         (C
          (<Enter format of RLGC here>  ; BandedSymmetricMatrix /
                                        ; SymmetricMatrix /
                                        ; SparseSymmetricMatrix
            (band "<Enter B_NUMBER here>")
            (dimension "<Enter D_NUMBER here>")
            (data "<Enter data_values here>") ) ) )
     )
     (CircuitModels
        (SingleLineCircuits
     ("<Enter pin_number here>"
        (SubCircuitName <name_of_subckt>)
        (SubCircuitName <name_of_subckt>) )
```

```
    )
    (SubCircuits "
     .subckt <name_of_subckt> <input> <output>
    ")
    (CoupledLineCircuits
     ("<Enter pin_number here>"
      (Terminals <terminal_number.in terminal_number.out>)
      (SubCircuitName <name_of_subckt>)
      (SubCircuitName <name_of_subckt>) )
    (SubCircuits "
     .subckt <name_of_subckt> <input> <output>
    ")
  )
 )
)
```

C

BoardModel Examples

BoardModel Template

```
("DMLfilename.dml"
   (BoardModel
     (ModelVersion "<Enter text here>")
     (ModelSource "<Enter text here>")
     (ModelDate "<Enter text here>")
     (Manufacturer "<Enter text here>")
     ("<Enter name of board model here>"
      (PinMap "<Enter number of pins here>")
        ("<Enter pin number here>" "<Enter device name here> <Enter signal name/
        number here>" "<Enter device name here> <Enter signal name/number here>") )
      (CircuitModels
       (SingleLineCircuits
        ("<Enter pin_number here>"
          (SubCircuitName <name_of_subckt>)
          (SubCircuitName <name_of_subckt>) )
       )
       (SubCircuits "
        .subckt <name_of_subckt> <input> <output>
       ")
       (CoupledLineCircuits
        ("<Enter pin_number here>"
          (Terminals <terminal_number.in terminal_number.out>)
          (SubCircuitName <name_of_subckt>)
          (SubCircuitName <name_of_subckt>) )
        (SubCircuits "
         .subckt <name_of_subckt> <input> <output>
        ")
     (Comps
        ("<Enter device name here>"
         (Pins
          ("<Enter pin number/name here>"
               (pinuse "<Enter text here" )
               (dcvolt "<Enter value here>") )
         (signal_model "<Enter name of signal model here>") )
     )
  ))
```

D

AnalogOutput Examples

AnalogOutput Template

```
("DMLfilename.dml"
  (AnalogOutput
     ("<Enter name of model here>"
     (ModelVersion "<Enter text here>")      ; optional but recommended
     (ModelSource "<Enter text here>")       ; optional but recommended
     (ModelDate "<Enter text here>")         ; optional but recommended
     (Manufacturer "<Enter text here>")      ; optional but recommended
     (GroundVoltage "<Enter value here>")
     (PowerVoltage "<Enter value here>")
     (TheveninCircuit
      ".subckt <Enter Spice sub-circuit definition here>
      <Enter other Spice parameter here>") )
     (Rise
      (IDL
       ".subckt <Enter Spice sub-circuit definition here>
       <Enter other Spice parameter here>")
      (WaveFile "<Enter name of file including its path here>") )
     (Fall
      (IDL
       ".subckt <Enter Spice sub-circuit definition here>
       <Enter other Spice parameter here>")
      (WaveFile "<Enter name of file including its path here>") )
     (Pulse
      (IDL
       ".subckt <Enter Spice sub-circuit definition here>
       <Enter other Spice parameter here>")
      (WaveFile "<Enter name of file including its path here>") )
     (InvertedPulse
      (IDL
       ".subckt <Enter Spice sub-circuit definition here>
       <Enter other Spice parameter here>")
      (WaveFile "<Enter name of file including its path here>") )
     )
  )
)
```

DML File of GUI Example (Figure 2-9)

```
("exampleE2.dml"
 (AnalogOutput
  (AnalogOutputExample
   (TheveninCircuit
".subckt denman_TheveninCircuit vcc in ground out
R1 in out 1
.ends denman_TheveninCircuit
" )
   (Pulse
    (WaveFile "C:\PSD_Data\old_tut\900MHz.wave" )
    (IDL ".subckt denman_Pulse vout vref
VAWB vout vref PWL(
+ 0 0
+ 1e-010 1.60748
+ 2e-010 2.71448
+ 3e-010 2.97634
+ 4e-010 2.31154
+ 5e-010 0.927051
+ 6e-010 -0.74607
+ 7e-010 -2.18691
+ 8e-010 -2.94686
+ 9e-010 -2.78933
+ 1e-009 -1.76336
+ 1.1e-009 -0.188371
+ 1.2e-009 1.44526
+ 1.3e-009 2.62892
+ 1.4e-009 2.99408
+ 1.5e-009 2.42705
+ 1.6e-009 1.10437
+ 1.7e-009 -0.562145
+ 1.8e-009 -2.05364
+ 1.9e-009 -2.90575
+ 2e-009 -2.85317
+ 2.1e-009 -1.91227
+ 2.2e-009 -0.375998
+ 2.3e-009 1.27734
+ 2.4e-009 2.53298
+ 2.5e-009 3
+ 2.6e-009 2.53298
+ 2.7e-009 1.27734
+ 2.8e-009 -0.376002
+ 2.9e-009 -1.91227
+ 3e-009 -2.85317
+ 3.1e-009 -2.90575
+ 3.2e-009 -2.05364
+ 3.3e-009 -0.562142
+ 3.4e-009 1.10438
+ 3.5e-009 2.42705
+ 3.6e-009 2.99408
```

```
+ 3.7e-009 2.62892
+ 3.8e-009 1.44526
+ 3.9e-009 -0.188374
+ 4e-009 -1.76336
+ 4.1e-009 -2.78933
+ 4.2e-009 -2.94686
+ 4.3e-009 -2.1869
+ 4.4e-009 -0.746067
+ 4.5e-009 0.927054
+ 4.6e-009 2.31154
+ 4.7e-009 2.97634
+ 4.8e-009 2.71448
+ 4.9e-009 1.60748
+ 5e-009 -3.35308e-006
+ 5.1e-009 -1.60748
+ 5.2e-009 -2.71448
+ 5.3e-009 -2.97634
+ 5.4e-009 -2.31154
+ 5.5e-009 -0.927047
+ 5.6e-009 0.746073
+ 5.7e-009 2.18691
+ 5.8e-009 2.94686
+ 5.9e-009 2.78933
+ 6e-009 1.76335
+ 6.1e-009 0.188367
+ 6.2e-009 -1.44526
+ 6.3e-009 -2.62892
+ 6.4e-009 -2.99408
+ 6.5e-009 -2.42705
+ 6.6e-009 -1.10437
+ 6.7e-009 0.562148
+ 6.8e-009 2.05364
+ 6.9e-009 2.90575
+ 7e-009 2.85317
+ 7.1e-009 1.91227
+ 7.2e-009 0.375995
+ 7.3e-009 -1.27734
+ 7.4e-009 -2.53299
+ 7.5e-009 -3
+ 7.6e-009 -2.53298
+ 7.7e-009 -1.27733
+ 7.8e-009 0.376005
+ 7.9e-009 1.91228
+ 8e-009 2.85317
+ 8.1e-009 2.90575
+ 8.2e-009 2.05364
+ 8.3e-009 0.562138
+ 8.4e-009 -1.10438
+ 8.5e-009 -2.42705
+ 8.6e-009 -2.99408
+ 8.7e-009 -2.62892
```

```
+ 8.8e-009 -1.44526
+ 8.9e-009 0.188378
+ 9e-009 1.76336
+ 9.1e-009 2.78933
+ 9.2e-009 2.94686
+ 9.3e-009 2.1869
+ 9.4e-009 0.746064
+ 9.5e-009 -0.927057
+ 9.6e-009 -2.31154
+ 9.7e-009 -2.97634
+ 9.8e-009 -2.71448
+ 9.9e-009 -1.60747
+ 1e-008 6.70615e-006
+ )
.ends denman_Pulse
" ) ) ) )
```

E

DesignLink Examples

DesignLink Template

```
("DMLfilename.dml"
  (DesignLink
    ("<Enter name of design link model here>"
     (ModelVersion "<Enter text here>")      ; optional but recommended
     (ModelSource "<Enter text here>")       ; optional but recommended
     (ModelDate "<Enter text here>")         ; optional but recommended
     (Manufacturer "<Enter text here>")      ; optional but recommended
     (Connections
      ("<Enter connection name here>"
        (Length "<Enter value here>" )
               (PinMap
          (<Enter wire number here> "<Enter From design name here> <Enter From
          component refdes here> Enter From pin number here>" > "<Enter to design
          name here> <Enter To component refdes here> Enter To pin number here>") )
          (Cable "<Enter name of cable model here>" )
        (RLGC
         ("<Enter frequency value here>"
           (R
             (<Enter format of RLGC here>    ; BandedSymmetricMatrix /
                                  ; SymmetricMatrix /
                                  ; SparseSymmetricMatrix
             (band "<Enter B_NUMBER here>")
             (dimension "<Enter D_NUMBER here>")
             (data "<Enter data_values here>") ) )
           (L
             (<Enter format of RLGC here>    ; BandedSymmetricMatrix /
                                  ; SymmetricMatrix /
                                  ; SparseSymmetricMatrix
             (band "<Enter B_NUMBER here>")
             (dimension "<Enter D_NUMBER here>")
             (data "<Enter data_values here>") ) )
           (C
             (<Enter format of RLGC here>    ; BandedSymmetricMatrix /
                                  ; SymmetricMatrix /
                                  ; SparseSymmetricMatrix
             (band "<Enter B_NUMBER here>")
             (dimension "<Enter D_NUMBER here>")
             (data "<Enter data_values here>") ) ) )
```

```
     (CircuitModels
        (SingleLineCircuits
         (“<Enter pin_number here>”
           (SubCircuitName <name_of_subckt>)
           (SubCircuitName <name_of_subckt>) ) )
     (CoupledLineCircuits
     (“<Enter pin_number here>”
         (Terminals <terminal_number.in terminal_number.out>)
         (SubCircuitName <name_of_subckt>)
         (SubCircuitName <name_of_subckt>) )
     (SubCircuits “
       .subckt <name_of_subckt> <input> <output>
     ”) ) ) ) )
   (Drawings
     (“<Enter design name here>” “<Enter drawing name here>” )
     (“<Enter design name here>” “<Enter drawing name here>” ) ) )
     )
  )
)
```

DML File of GUI Example (Figure 2-11) for DesignLink

```
(“exampleF2.dml”
 (DesignLink
   ("BRD1_U1_to_BRD2_U2"
    ("Connections"
     ("cable_1"
         ("Length" "0.05")
         ("PinMap"
          ("1" "BRD1 U1 4" "BRD2 U2 4")
          ("2" "BRD1 U1 5" "BRD2 U2 5")
          ("3" "BRD1 U1 6" "BRD2 U2 6")
          ("4" "BRD1 U1 7" "BRD2 U2 7")
         )
         ("Cable" "FourWireCable")
     )
    )
    ("Drawings"
     ("BRD1" "blm2_pos.brd")
     ("BRD2" "blm2_pos.brd")
    )
)
```

F

Cable Model Examples

Cable Template

```
("DMLfilename.dml"
  (Cable
     (ModelVersion "<Enter text here>")      ; optional but recommended
     (ModelSource "<Enter text here>")       ; optional but recommended
     (ModelDate "<Enter text here>")         ; optional but recommended
     (Manufacturer "<Enter text here>")      ; optional but recommended
     ("<Enter name of cable model here>"
       (Wires "<Enter number of wires here>")
       (PinNameToNumber
          ("<Enter pin name here>" "Enter pin number here")
          ("<Enter pin name here>" "Enter pin number here") )
       (RLGC
        ("<Enter frequency value here>"
         (R
           (<Enter format of RLGC here> ; BandedSymmetricMatrix /
                                        ; SymmetricMatrix /
                                        ; SparseSymmetricMatrix
           (band "<Enter B_NUMBER here>")
           (dimension "<Enter D_NUMBER here>")
           (data "<Enter data_values here>") ) )
         (L
           (<Enter format of RLGC here> ; BandedSymmetricMatrix /
                                        ; SymmetricMatrix /
                                        ; SparseSymmetricMatrix
           (band "<Enter B_NUMBER here>")
           (dimension "<Enter D_NUMBER here>")
           (data "<Enter data_values here>") ) )
         (C
           (<Enter format of RLGC here> ; BandedSymmetricMatrix /
                                        ; SymmetricMatrix /
                                        ; SparseSymmetricMatrix
           (band "<Enter B_NUMBER here>")
           (dimension "<Enter D_NUMBER here>")
           (data "<Enter data_values here>") ) ) )
       (CircuitModels
        (SingleLineCircuits
         ("<Enter pin_number here>"
          (SubCircuitName <name_of_subckt>)
```

```
          (SubCircuitName <name_of_subckt>) ) )
        (CoupledLineCircuits
         ("<Enter pin_number here>"
          (Terminals <terminal_number.in terminal_number.out>)
          (SubCircuitName <name_of_subckt>)
          (SubCircuitName <name_of_subckt>) )
        (SubCircuits "
         .subckt <name_of_subckt> <input> <output>
     ") )
     )
   )
)
```

G

MacroModel Examples

MacroModel Starter File

This subsection includes a starter file that could be used when you need to build a MacroModel. Any arbitrary/complex circuit could be created in the sub-circuit part in the example below. The starter MacroModel implemented here is very simple, embedding an output resistor in the default IbisIOCell. Please make sure that you take note of the following tips when you're building macromodels:

- Watch your parenthesis matching; the tool is picky about these.
- When debugging, look in `tlsim.log` for clues. An error message to the screen like `cycle.msm does not exist` probably means the simulator failed to compile your sub-circuit.
- When testing/modifying the file, be careful. You can remove/re-add the library dml, delete and replace the component, but still not get the updated change. Unfortunately, the safest thing to do is exit and re-enter the tool.

Starter File:

```
("macromodel_starter.dml"    ; This is the filename
;
; Everything down to ‘ (Subcircuits " ’ is wrapper info for your MacroModel to make
; sure the tool uses it correctly, and it is integrated into the UI.
;
(IbisIOCell           ; This says we're doing buffers, not packages etc
  (“starter”          ; This is the name that appears in the library browser
                      ; and must match the name in the .subckt defined below.
   (Model
    (ModelType “IO” ) ; This will decide which SigXp symbol is used
    )
  (Ramp
;
; The dt values below control the input stimulus edge speed.
; They are programmed to 1,2,3 ns rise/fall since tr/f = dt/0.6.
; These must be here, or you will get a divide by zero error.
```

```
;
    (Rise
     (minimum
      (dt "0.6n" ) )
     (typical
      (dt "1.2n" ) )
     (maximum
      (dt "1.8n" ) ) )
    (Fall
     (minimum
      (dt "0.6n" ) )
     (typical
      (dt "1.2n" ) )
     (maximum
      (dt "1.8n" ) ) ) )
;
; Define the various "Thresholds" if they will be used for measurements
;
    (LogicThresholds
     (Input
      (High
       (typical "1.2" ) )
      (Low
       (typical "0.8" ) ) )
     (Output
      (High
       (typical "1.4" ) )
      (Low
       (typical "0.6" ) ) ) )
;
; The following reference voltages will probably be needed to ensure that the correct
; voltages are used.
;
   (PullUp
    (ReferenceVoltage
     (maximum "3" )
     (minimum "1" )
     (typical "2" ) ) )
    (PullDown
     (ReferenceVoltage
      (maximum "0" )
      (minimum "0" )
      (typical "0" ) ) )
    (GroundClamp
     (ReferenceVoltage
      (maximum "0" )
      (minimum "0" )
      (typical "0" ) ) )
    (PowerClamp
     (ReferenceVoltage
      (maximum "1."5 )
```

```
      (minimum “1.5” )
      (typical “1.5” ) ) )
;
   (MacroModel
    (Parameters
     (Buffers
;
; This part is mandatory, but can be faked if not needed.
;
      (“BUFF” “CDSDefaultIO”)))
    (NumberOfTerminals “7” )    ; you can specify the # of terminals
    (MacroType “DiffIO” )       ; not clear on use/value of this - DiffI_SO
;
; If there are other terminals you want available for Custom Measurements AND to be
; editable in the Stimulus Editor, add them here...
;
     (PinTerminalsMap
      (4 “Data” )
      (5 “Enable” ) )
    (SubCircuits”
*
* Please note that the comment character inside the SubCircuits” is an asterisk (*).
* Following is a handy “bdrvr” node cheat sheet:
*      power = 1
*      output = 2
*      ground = 3
*      input = 4
*      enable = 5
*      power-clmap_reference = 6
*      ground clamp reference = 7
*  Node group 8-11 is optional:
*      scale factor: pullup = 8
*      scale factor: pulldown = 9
*      scale factor: powerclamp = 10
*      scale factor: groundclamp = 11
*
.subckt starter 1 2 3 4 5 6 7 ibis_file=ibis_models.inc BUFF=CDSDefaultIO
*
bdrvr 1 20 3 4 5 6 7 model=BUFF file=ibis_file
Rout 2 20 30
*
* You can cause any node to be printed, and give it any name you like as shown here:
*
.node_param 4 name=(name(2).input) print
.node_param 20 name=(name(2).inside_resistor) print
*
.ends starter
" ) ) ) ) )
```

H

DML-Wrapped HSpice Model

```
("CONN5AB"
    ("ESpice"
"
.SUBCKT CONN5AB dc_a bp_a dc_b bp_b
BEGIN_LITERAL_HSPICE
xXYZ5AB 0 dc_a dc_b bp_a bp_b XYZ5AB
.include '..\..\..\Referenced_Files\xyz5ab.cir'
END_LITERAL_HSPICE
.ENDS CONN5AB
")
    ("PinConnections"
     ("bp_a" "dc_a")
     ("bp_b" "dc_b")
     ("dc_a" "bp_a")
     ("dc_b" "bp_b")
    )
    ("XNetConnections"
     ("bp_a" "dc_a")
     ("bp_b" "dc_b")
     ("dc_a" "bp_a")
     ("dc_b" "bp_b")
    )
)

The called Hspice connector circuit is below.
************************************************
.SUBCKT XYZ5AB
+   100      101              102
+           2401             2402
*
************************************************
.ENDS XYZ5AB
************************************************
```