

## ABSTRACT

FERNANDO, PAUL R. Adding scalability to IBIS using AMS languages. (Under the direction of Dr. Paul Franzon.)

From 1993 to about 1998, IBIS remained THE digital IO buffer model format. But as the operating frequencies & complexity of I/O buffers increased, IBIS has been left behind in favor of SPICE models, since IBIS is inaccurate or unable to model these advanced buffers. This trend brings the industry back toward a single EDA vendor solution, which is what IBIS was designed to prevent. In an effort to relinquish these shortcomings, multi-lingual model extensions were added to IBIS Version 4.1. Specifically: Berkeley-SPICE, VHDL-AMS and Verilog-AMS files.

These extensions in IBIS 4.1 give IBIS practically unlimited behavioral and structural modeling capabilities as well as more accuracy. The problem is that the AMS languages have been slow in making their way into SI tools and the SI community; mainly due to the associated learning curve, since AMS is relatively new to the SI world.

The solution was to build an AMS library of tool independent basic elements (“element library”) and a separate “template library” which would contain the models of complex buffers (Pre-emphasis, LVDS, DDR2 etc). The templates would be created by instancing elements from the “element library”. An IBIS to AMS converter would convert conventional IBIS files into AMS format and provide the data for the template. The IBIS macro-modeling committee was created in July 2005 with these main goals in mind.

This thesis deals with the new AMS macro-modeling methodology put forth by the IBIS macro-modeling committee and the contributions I made to it as its only student member. My specific contribution was the IBIS to AMS (ibis2ams) converter tool. The thesis also presents the updates I made to the IBIS plotting utility (s2iplt) and the spice to IBIS toolkit (s2ibis). All tools are publicly available on the NCSU ERL website.

**Adding scalability to IBIS using AMS languages**

by

**Paul R. Fernando**

A thesis submitted to the Graduate Faculty of  
North Carolina State University  
In partial fulfillment of the  
Requirements for the degree of  
Master of Science  
In

**Computer Engineering**

Raleigh, NC

2006

Approved by:

---

**Dr. Paul Franzon**  
Chair of Advisory Committee

---

**Dr. Rhett W. Davis**

---

**Dr. Kevin Gard**

## DEDICATION

*To my Parents, Teachers,*

*Family & Friends....*

## **BIOGRAPHY**

Paul Fernando was born in Colombo, Sri Lanka, in October 1982. He joined Louisiana State University in Baton Rouge, LA at age 16 and graduated cum laude in May 2003 majoring BS in Computer Engineering with minors in Mathematics and Computer Science. He received his second BS in Electrical Engineering in December 2003. He interned at Dallas Semiconductors (Maxim Integrated Products) in Dallas, TX in the summer of 2002 where he was introduced to IBIS modeling and NCSU's s2ibis toolkit for the first time.

He began his MS in Computer Engineering at North Carolina State University in Fall 2004 under the supervision of Dr. Paul Franzon. A Summer and Fall co-op in 2005 at Analog Devices, Boston MA, focused, among many other things, on creating a novel flow for fast and accurate IBIS model generation within the company. After working with state of the art IO buffer models at ADI, it became clear that IBIS required major additions to retain accuracy. Since then, he has been working with the IBIS macro-modeling committee consisting of IBIS industry leaders from Intel, Cisco, Cadence, Mentor Graphics, Teraspeed Consulting, SiSoft etc. to develop a new methodology which incorporates AMS languages to IBIS.

He will begin work in May 2006 as a full time employee in the Visual Signal Processing group at Analog Devices, Wilmington, MA.

## ACKNOWLEDGMENTS

A warm “Thank You” to Dr. Paul Franzon for the support and guidance he provided me throughout my Masters program. His help was imperative in gaining my co-op at Analog Devices last year which resulted in me getting my full-time offer even before I finished the co-op. I’d also like to acknowledge the members of my committee, Dr. Rhett Davis and Dr. Kevin Gard. Together with Dr. Maysam Ghovanloo they, among many other things, made my first semester at NCSU a very challenging, exciting and fulfilling one. Ambrish Varma played a vital role during my MS program right from the start. The valuable discussions we had proved to be essential for my thesis.

Thank you, Dr. Bob Evans, for putting me in touch with Mike LaBonte and the IBIS macro-modeling committee. And also for teaching, what is arguably, one of the best Signal Integrity classes on the planet.

I would like to express my gratitude to the entire IBIS macro-modeling committee. Specifically, Arpad Muranyi (Intel), Mike LaBonte (Cisco), Todd Westerhoff (Cisco), Walter Katz (SiSoft), Barry Katz (SiSoft), Ken Willis (Cadence), Ian Dodd (Mentor), Sam Chitwood (Sigrity), Bob Ross (Teraspeed) & Scott McMorow (Teraspeed).

My thanks go out especially to Arpad & Mike. Your help added an industrial dimension to my work. Thanks Arpad, for the extensive feedback while developing ibis2ams.

At the time this thesis was authored, NCSU had no simulator capable of handling AMS languages. Dolphin Integration provided a free license for SMASH, its mixed signal (AMS) simulator worth \$25k. Thank you to Gilles Depeyrot & Thierry Depeyrot of Dolphin Integration.

# TABLE OF CONTENTS

<b>List of Tables</b> .....	<b>vii</b>
<b>List of Figures</b> .....	<b>viii</b>
<b>1. Introduction</b> .....	<b>1</b>
1.1 Why IBIS? .....	1
1.2 What is IBIS (I/O Buffer Information Specification) .....	4
1.3 3-state Driver IBIS data extraction .....	7
1.4 NCSU's involvement with IBIS .....	11
1.5 IBIS Limitations .....	12
1.6 Thesis Organization .....	14
<b>2. The AMS Macro-modeling Concept</b> .....	<b>15</b>
2.1 Introduction .....	15
2.2 The IBIS Macro Modeling Committee (IBIS-macro) .....	19
2.3 The IBIS-AMS Macro-model Hierarchy .....	21
2.4 The simulation algorithm .....	22
<b>3. The IBIS-to-AMS converter</b> .....	<b>26</b>
3.1 Introduction .....	26
3.2 The IBIS-to-AMS Flow .....	27
<b>4. Results</b> .....	<b>32</b>
4.1 Introduction .....	32
4.2 Case study I: Output Buffer .....	33
4.3 Case study II: Pre-emphasis Buffer .....	36
4.4 Case study III: Equalized Receiver .....	42
<b>5. Conclusions &amp; Future Research</b> .....	<b>45</b>
5.1 Conclusions .....	45
5.2 Future Research .....	46
<b>References</b> .....	<b>47</b>

<b>Appendix A: IBIS Macro Library Elements .....</b>	<b>50</b>
<b>Appendix B: IBIS to Verilog-A converter script .....</b>	<b>52</b>
<b>Appendix C: The Verilog-AMS “IBIS_IO” model description .....</b>	<b>73</b>
<b>Appendix D: Simulating the “IBIS_IO” model in Dolphin Smash .....</b>	<b>86</b>
<b>Appendix E: Case Study I: Output Buffer simulation files .....</b>	<b>88</b>
<b>Appendix F: Case Study II: Pre-emphasis Buffer simulation files .....</b>	<b>90</b>
<b>Appendix G: Case Study III: Equalized Receiver simulation files .....</b>	<b>94</b>
<b>Appendix H: IBIS plotting utility (s2iplt) .....</b>	<b>97</b>
<b>Appendix I: A Perl/Tk GUI for s2ibis .....</b>	<b>101</b>

## LIST OF TABLES

Table A.1: Basic Elements .....	50
Table A.2: Sources .....	50
Table A.3: Operators .....	51
Table A.4: Active Devices .....	51
Table A.5: Other Models .....	51

## LIST OF FIGURES

Figure 1.1: Moore's Law . . . . .	2
Figure 1.2: 3-state buffer structure . . . . .	7
Figure 1.3: PU & PD curves . . . . .	8
Figure 1.4: PC & GC Curves . . . . .	9
Figure 2.1: Current IBIS timeline . . . . .	15
Figure 2.2: Proposed IBIS timeline . . . . .	16
Figure 2.3: AMS Hierarchy . . . . .	21
Figure 2.4: IBIS Simulation Circuit . . . . .	23
Figure 3.1: Simulation Process Diagram . . . . .	26
Figure 3.2: ibis2ams Convert Manu . . . . .	27
Figure 3.3: ibis2ams Select IBIS file Dialog . . . . .	27
Figure 3.4: ibis2ams Select model to convert. . . . .	28
Figure 3.5: ibis2ams Select process corner . . . . .	29
Figure 3.6: ibis2ams Select output file name . . . . .	29
Figure 3.7: Sample Verilog-AMS output file . . . . .	30
Figure 3.8: Sample VHDL-AMS output file . . . . .	31
Figure 4.1 Conversion Flow . . . . .	33
Figure 4.2 Test Circuit . . . . .	33
Figure 4.3 SMASH GUI . . . . .	34
Figure 4.4 Output Comparison . . . . .	35
Figure 4.5 Pre-emphasis Buffer Schematic . . . . .	37
Figure 4.6 Pre-emphasis Buffer VHDL-AMS code (Part 1) . . . . .	38

Figure 4.7 Pre-emphasis Buffer VHDL-AMS code (Part 2) . . . . .	39
Figure 4.8 Pre-emphasis Buffer Test Circuit . . . . .	40
Figure 4.9: Pre-emphasis Buffer SMASH GUI . . . . .	40
Figure 4.10: Pre-emphasis Buffer SMASH Output . . . . .	41
Figure 4.11: Receiver Schematic . . . . .	42
Figure 4.12: Receiver VHDL-AMS code . . . . .	43
Figure 4.13: Receiver SMASH Output . . . . .	44
Figure D.1: SMASH GUI . . . . .	86
Figure D.2: SMASH output for Risetime = 0.1ns . . . . .	87
Figure D.3: SMASH output for Risetime = 0.5ns . . . . .	87
Figure H.1: s2iplt GUI main interface . . . . .	98
Figure H.2: s2iplt in50v Power Clamp Curve . . . . .	99
Figure H.3: s2iplt sample PostScript file . . . . .	100
Figure I.1: s2ibis GUI interface . . . . .	102

# 1. Introduction

## 1.1 Why IBIS?

The fastest entity in the universe is an electromagnetic (EM) wave traveling in a vacuum. This velocity is defined as  $C$  ( $3.0 \times 10^8$  m/s). Typically, signals on a PCB travel about half  $C$ . Thus, it takes about 100ps for light to travel from your nose to your eyes [1]. In the latest electronic systems a signal skew of a few 100ps would be sufficient to corrupt the information it carries. Given that  $C$  is a universal constant it's easy to see that modern electronics is reaching a ceiling in terms of signaling speed. Adding to the complexity is that at high frequencies, conductors no longer act as simple wires. They begin to act as transmission lines exhibiting high-frequency effects. And as frequencies increase, so do these complex effects. Transmission line conditions occur when the physical size of a circuit reaches the wavelength of the highest frequency of interest in a signal. For digital systems, higher digital frequencies require sharp edge rates which contain high frequency components (harmonics). Thus, modern board simulations require TL analysis. Nowadays, even clock trees within a single chip require this analysis. Simulating for these high frequency effects involves the use of complex 3-D field solvers. If these effects are unaccounted for in high speed circuit design, the results will almost certainly be disastrous. For this reason, simulation takes a massive role in validating modern designs as timing gets increasingly constrained. It is said that all simulations are inaccurate, but some simulations are useful. The goal in simulation is to obtain the greatest accuracy; the tradeoffs being time, manpower and so on.

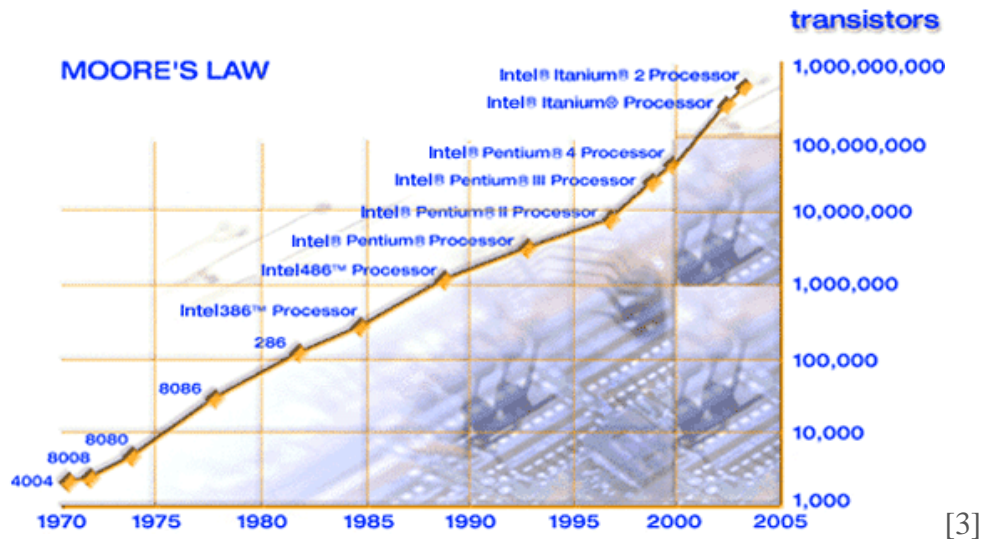


Figure 1.1: Moore's Law

When the integrated circuit was invented by Jack Kilby in 1958, circuit design split into two disciplines: Discrete (board-level) and IC (chip-level). Historically, chip level circuit design has targeted performance (see Fig 1.1), while board level circuit design has targeted cost. The result has been that while signaling speeds within a microchip have been increasing exponentially with respect to time, this has not been the case at the board level (chip to chip). Thus, a majority of the speed bottleneck in modern systems is at the board level. For example, since the signaling between a CPU and the RAM (bus speed) is too slow, it has become mandatory to use on-chip memory (Cache) running at the CPU's core frequency. Modern processors have more than 4Mb of cache and, based on current trends, increases are expected. This is also one of the reasons for the recent multi-core trend in CPUs. Even though these developments take a massive overhead they have been necessary due to the board level speed bottleneck.

Another distinguishing factor here is that IC level design is (usually) completed within a single company, while board level design involves components manufactured by a variety of companies. For example, ASUS designs a motherboard using an Intel CPU, Micron DRAM, Nvidia GPU etc. For optimum system performance ASUS would require complete design information about each component before working on its (system level) design. While sharing design information with a company is complex in itself, sharing this information between companies would be disastrous to the supplier company's business. The netlist would contain process information giving details about the latest FABs & processes not to mention complete design information. For these (and many more) reasons, companies would rather not disclose such proprietary information. Thus, the industry standard circuit simulator, Spice, does not fulfill board-level simulation requirements, since it provides a complete description of the design as well as process information. Another factor was that Spice is component centric, and therefore, simulation of massive circuits takes a long time. In the early 1990s it became clear that a new modeling technique was in order. The technique introduced to fulfill this requirement was IBIS.

## 1.2 What is IBIS (I/O Buffer Information Specification)

IBIS was developed by Intel<sup>®</sup> Corporation in the early 1990s. IBIS version 1.0 was issued in June 1993 along with the creation of the IBIS Open Forum. The IBIS Open Forum comprises of EDA vendors, computer manufacturers, semiconductor vendors, universities, and end-users. It proposes updates and reviews, revises standards, and organizes summits related to IBIS. It promotes IBIS models and provides useful documentation and tools on the IBIS website. In 1995, the IBIS Open Forum teamed up with the Electronic Industries Alliance (EIA). The main facets of IBIS are that [2]:

1. It protects proprietary information. (No circuit information or process parameters are disclosed)
2. Non-linear aspects of I/O are modeled as well as package parasitics and ESD structure.
3. It is much faster (upto 25x) than other transistor level simulators such as SPICE.
4. The IBIS model can be provided before silicon. (Pre tape-out)
5. Supported by most SI industry tools.

Given these advantages, IBIS has become the standard for board level Signal Integrity (SI) simulations.

An IBIS model consists of tabular data made up of current and voltage values at the output and input pins, as well as the voltage and time relationship at the output pins under rising or falling switching conditions (with varying output load conditions). This tabulated data represents the ‘behavior’ (including non-linear characteristics) of the buffer. IBIS only deals with digital buffers & signals.

An IBIS model is (primarily) used by an SI tool to calculate fundamental signal integrity concerns in transmission lines that connect different components on a PCB.

Potential problems that can be analyzed by means of these simulations include the degree of energy reflected back to the driver from the wave that reaches the receiver due to mismatched impedance in the line; crosstalk; ground and power bounce; overshoot; undershoot; and line termination analysis, among others [4].

IBIS is an accurate model since it takes into account nonlinear aspects of the I/O structures, the ESD structures, and the package parasitics. It has several advantages over other traditional models such as SPICE. Thus, for example, the simulation time can be up to 25 times less, and IBIS does not have the non-convergence problem SPICE does. In addition, IBIS can be run on any industry-standard tool since most Electronic Design Automation (EDA) vendors support the IBIS specification.

IBIS files can be generated using bench measurements or by running (SPICE) simulations. The second method is more frequently used since it takes the least amount of overhead from the model maker's perspective. A second reason is that the bench results are directly impacted by the accuracy of the measurement equipment used as well as other environmental effects. A third reason is that IBIS files created from simulations can be provided to customers before tape-out. Therefore, customers get a head start on system level design before first silicon.

IBIS models are characterized under three corner conditions. The "slow" corner is characterized under minimum supply voltage, maximum temperature and slow process parameters (device models). The "fast" corner is characterized under maximum supply voltage, minimum temperature and fast process parameters (device models). The "typical" corner is characterized under nominal conditions.

The IBIS specification supports many kinds of drivers: Input, Output, I/O, 3-state, Open\_drain, I/O\_open\_drain, Open\_sink, I/O\_open\_sink, Open\_source, I/O\_open\_source, Input\_ECL, Output\_ECL, I/O\_ECL, 3-state\_ECL, etc [6]. IBIS does not support analog buffers.

In the next section, a 3-state driver will be used as an example to explain what simulations must be run to provide the data for an IBIS file.

### 1.3 3-state Driver IBIS data extraction

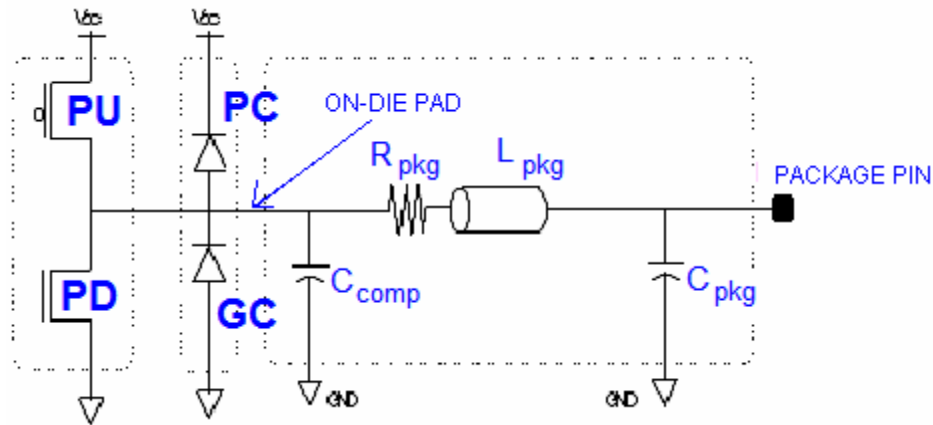


Figure 1.2: 3-state buffer structure

The basic structure of a 3-state driver is given above.

The PU & PD MOS devices form the 3-state driver: the output is hi when only the PU PMOS is on; the output is low when only the PD PMOS is on; the output is hi-z when both MOS devices are off. The package pin will be floating in the final case.

The PC & GC diodes form the ESD (Electro-static discharge) protection mechanism used to protect the chip from high voltages that may appear on the pad due to external environmental factors.

$C_{comp}$  lumps the capacitance of all the devices on the die pad. This capacitance can be approximated by adding the capacitances of the PU, PD, PC & GC components on the (on-die) pad node. This value is typically in the single digit Pico Farad range. Unfortunately, NCSU's s2ibis utility cannot extract this value.

The [package] RLC values reflect the (lumped) parasitic RLC that is present in the bond wire that connects the die pad to the package pin. This information is usually provided

by the package manufacturer. Alternatively, the [Package Model] keyword may be used to provide an RLC matrix for more accuracy.

The data captured in an IBIS file consists of the following (Note: package parasitics & C\_comp are not included when running these simulations):

1. Pullup and Pulldown V-I tables

With the output set **high**, the output node is swept from  $-V_{cc}$  to  $2V_{cc}$  while the corresponding current through the output node is measured. This V-I table forms the **Pullup** curve. For the **Pulldown** curve, the output is set **low** and the same measurement is repeated. For the Pullup curve, the voltages must be made  $V_{cc}$  relative by subtracting the values from  $V_{cc}$ . Finally, the clamp effects (obtained from the next set of simulations) must be subtracted from these results to remove the clamping effects.

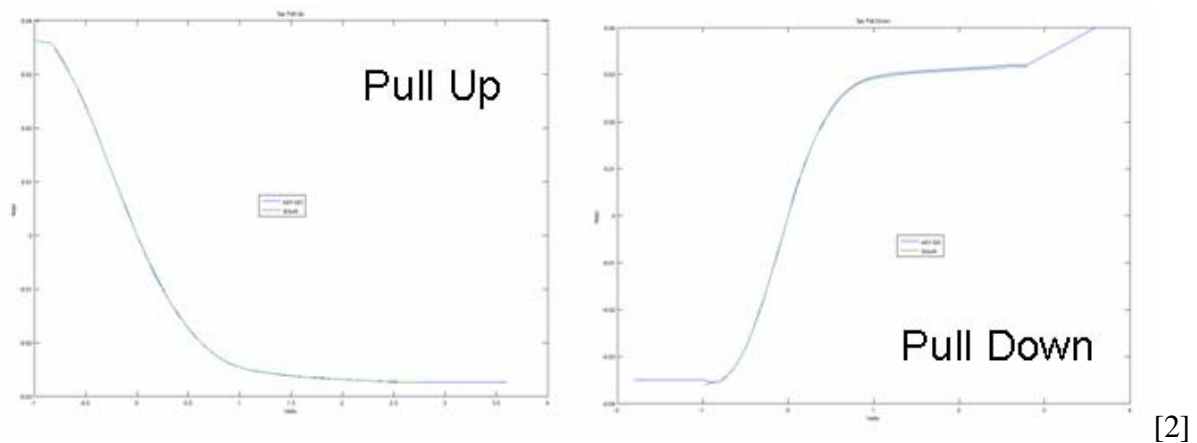


Figure 1.3: PU & PD Curves

## 2. Power clamp and Ground clamp V-I tables

The output is set to hi-z for these measurements. The output is swept from  $-V_{cc}$  to  $V_{cc}$  for the Ground clamp curve, while the output is swept from  $V_{cc}$  to  $2V_{cc}$  for the Power clamp curve. As in the Pullup case, the Power clamp curve must be made  $V_{cc}$  relative by subtracting each the value from  $V_{cc}$ .

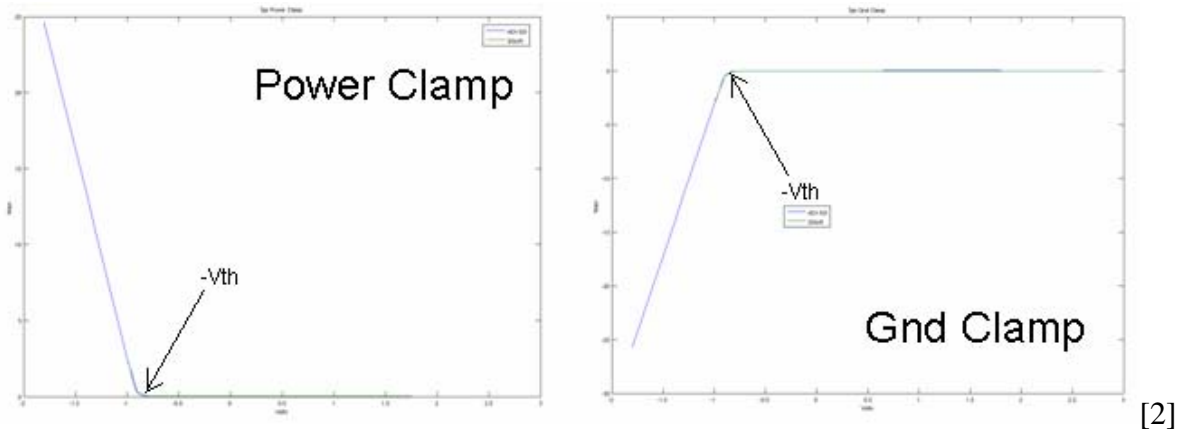


Figure 1.4: PC & GC Curves

## 3. Ramp rate ( $dV/dt$ )

The output is connected to a shunted 50Ohm resistor (standard load). The output is switched from low to high and high to low and the voltage-time variation from the 20% voltage point to the 80% voltage point is measured for each case. This is used to calculate the  $dV/dt$  for the rising and falling case respectively.

## 4. Rising and Falling V-time tables

The setup is similar to the Ramp rate setup but the resistor value may be varied. In this case the output voltage is measured with respect to time. The measurements are repeated with the resistor connected to  $V_{cc}$  (a pullup

configuration). Thus, at least four of these tables are **required** (i.e. rising with pulldown resistor, rising with pullup resistor, falling with pulldown resistor & falling with pullup resistor.) for a complete set of data.

NCSU s2ibis is commonly used to convert buffer spice netlists to IBIS format. It supports various flavors of spice.

## **1.4 NCSU's involvement with IBIS**

NCSU has been at the forefront of IBIS development since its conception over a decade ago.

The initial specification for IBIS (version 1.0) was issued in June 1993 with the formation of the IBIS Open Forum. The industry's first Spice to IBIS conversion utility, S2IBIS version 1 was released in July 1994 by the Electronics Research Lab at NCSU. Since then, NCSU has released S2IBIS2 and S2IBIS3 which met the demands of IBIS specifications version 2 & 3 respectively.

The current version, S2IBIS3 was written in Java and is capable of handling non-convergence issues in spice. It is platform independent. It is compatible with the following spice simulators: Hspice, Pspice, Spice2, Spice3, Spectre & Eldo. It also handles the IBIS v3.2 specific keywords such as [series MOSFET], [series pin mapping] and [series switch groups].

In 1995 NCSU introduced the IBIS plotting utility (s2iplt).

S2IBIS has clearly proven to be THE industry standard tool for spice to IBIS conversion; making NCSU the primary university involved with IBIS.

As part of this thesis, s2iplt (IBIS wave plotting utility) & s2ibis (Spice to IBIS converter) were updated to feature GUI interfaces & other functions.

## 1.5 IBIS Limitations

Since IBIS as a standard began to evolve in the 1990s it has gone through several updates. IBIS Version 4.1 (Released in February 2004) is the current standard at the time of this publication. Even though these updates have added new keywords to improve the scalability & accuracy of IBIS, the time lag between the emergence of new I/O buffer technologies, their support in the IBIS specification and support in IBIS simulators has continued to grow. As the operating frequencies of I/O buffers increase, many model makers and/or users leave IBIS behind in favor of SPICE models. Since most semiconductor vendors support their models with a single vendor's version of SPICE, this trend brings the industry back towards a single EDA vendor solution, which is what IBIS was designed to prevent.

It is quite clear that IBIS is currently having a hard time keeping up with advanced technology modeling (For example modeling pre/de emphasis buffers, LVDS & DDR2). However, as mentioned in section 1.2, behavioral modeling has significant advantages over SPICE.

In version 4.1, multi-lingual model extensions were added to IBIS so that it could recognize Berkeley-SPICE (Version 3F5), VHDL-AMS (IEEE Std. 1076.1-1999) and Verilog-AMS (IEEE 1364-2001) files. These files can be specified using the [external circuit] keyword. These extensions in IBIS 4.1 give IBIS practically unlimited behavioral and structural modeling capabilities as well as more accuracy. The problem is that the AMS languages are slow in making their way into SI tools and the SI community. There are many reasons for this problem [7]:

1. Model developers must learn a new language (\*-AMS)
2. Not many board level SI simulators support \*-AMS yet.

3. Few \*-AMS models exist.

However, there is a current need for modeling and simulating advanced buffers.

The purpose of this thesis is to provide an industry standard methodology that helps alleviate (if not solve) the above problem.

## **1.6 Thesis Organization**

The remainder of this thesis is organized as follows:

**Chapter 2** describes the AMS Macro-modeling concept.

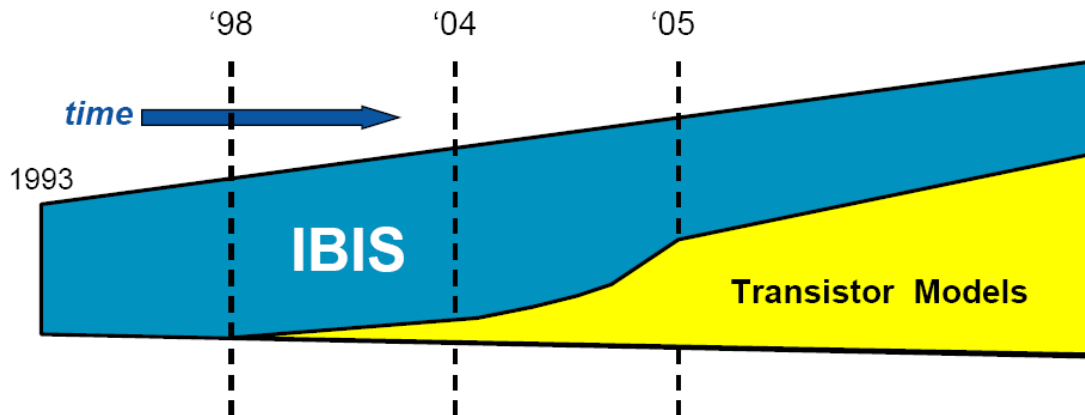
**Chapter 3** gives examples and results in the form of case studies.

**Chapter 4** provides Conclusions & Future Research.

As part of this thesis, s2iplt (IBIS wave plotting utility) & s2ibis (Spice to IBIS converter) were updated to feature GUI interfaces & other functions. These are described in Appendices H & I respectively.

## 2. The AMS Macro-modeling Concept

### 2.1 Introduction



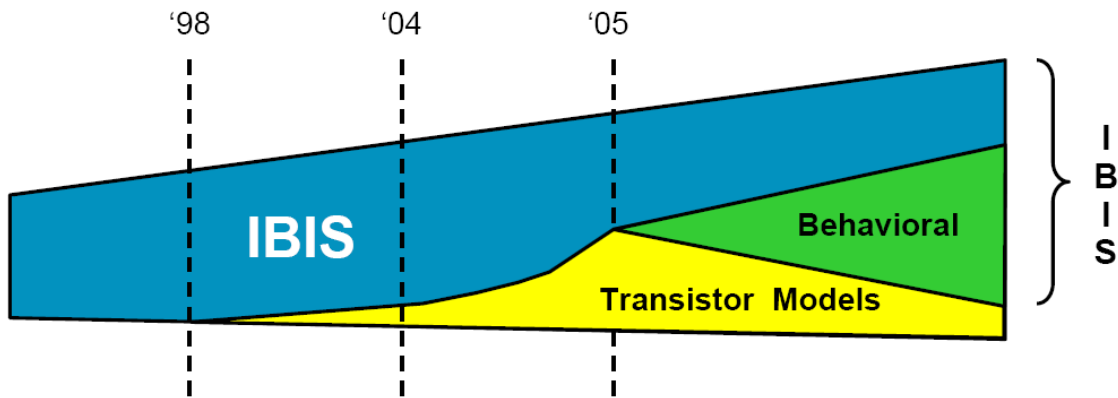
[8]

Figure 2.1: Current IBIS timeline

From 1993 to about 1998 IBIS remained THE digital IO buffer model format [8]. But as operating frequencies continued to increase, IBIS became less accurate in most cases and outright unusable in others. IBIS is running out of steam, and if nothing is done about this situation, the industry will end up with nothing but SPICE (transistor) models (figure 4.1). But clearly, SPICE is not a viable long term solution- mainly due to its long simulation times & proprietary information [8].

Simply adding new keywords to IBIS will not be a viable solution when it comes to keeping up with the ever faster evolution of complicated buffers. Clearly the future is in the adoption of flexible modeling languages, such as the ones accepted by IBIS 4.1. Specifically, Berkeley-SPICE (Version 3F5), VHDL-AMS (IEEE Std. 1076.1-1999) and Verilog-AMS (IEEE 1364-2001)

Thus, the evolution of IBIS is expected to encompass behavioral modeling and continue as shown in (figure 4.2). Simple buffers will continue to be modeled in ‘normal’ IBIS while state of the art high speed buffers (where accuracy is more important) will be modeled using behavioral languages. It is also possible that IBIS models will be purely behavioral in the not-too-distant future.



[8]

Figure 2.2: Proposed IBIS timeline

Behavioral modeling offers several benefits:

1. Fast simulation times
2. IP protection
3. Template based (therefore easier and faster to create)
4. Many tools support it (Verilog & VHDL are very popular)
5. Direct link to IC design

The most popular behavioral modeling languages are Verilog & VHDL. Traditionally, behavioral modeling has only supported the digital domain. This wouldn't be sufficient for IBIS since its table's exhibit analog behavior. Recently, Verilog & VHDL were

enhanced with Analog Mixed Signal (AMS) extensions which added capabilities of modeling analog & mixed signal behavior. Given these extensions, Verilog-AMS & VHDL-AMS were chosen to strengthen the IBIS paradigm. For high-speed buffer modeling, \*-AMS combines the behavioral-modeling capability of IBIS with the unrestricted topology description functionality of Spice [11].

AMS supports a variety of modeling styles - a buffer can be described purely behaviorally, but also in a structural manner with a circuit netlist and semiconductor process description. In addition, the netlist portion of the \*-AMS languages is very similar to the familiar SPICE netlists, and therefore it can be advantageous to describe the macro models in a familiar style.

The question is: how can we bridge the gap that exists today between these languages (Verilog & VHDL AMS) and the need for modeling advanced buffers?

Many tools today allow a modeling technique which is referred to as macro modeling. With this technique, a complex buffer may be constructed from one or more IBIS models along with some additional controlled voltage or current sources, delays, etc... This technique is gaining popularity because it helps to overcome well known IBIS limitations. However, since macro modeling is currently done differently in each simulator, this is again a tool dependent solution. On the other hand, due to its capabilities and immediate availability, macro modeling would be useful to the industry if this capability was available in a standard, tool independent form.

The solution was to build an AMS library of tool independent basic elements (“element library”). A separate “template library” would contain the models of complex buffers (Pre-emphasis, LVDS, DDR2 etc) which would be created by instancing elements

from the “element library”. An IBIS to AMS converter would convert conventional IBIS files into AMS format and provide the data for the template.

A new committee was needed to solve these issues. Thus, the IBIS macro modeling committee was formed in July 2005.

## 2.2 The IBIS Macro Modeling Committee (IBIS-macro)

Macro-modeling requires a standard, tool independent format. The solution is a library of standard \*-AMS "building blocks" that are defined and developed in \*-AMS to support behavioral macro modeling. It was to this end that the IBIS Macro Modeling Committee (IBIS-macro) was formed [9]. This library would also make it easier for model developers to transition into developing \*-AMS based models. These "building blocks" would include items like the IBIS B-element model, controlled voltage and current sources, and functions similar to features found in many SPICE simulators.

However, only a few EDA tools support \*-AMS modeling today, raising the question of whether the new models could be supported by simulators without \*-AMS support. This could be done by performing an automated netlist syntax translation and element mapping, assuming the standard building block library contains a set of functions commonly found in SPICE simulators.

Such a standard library would be useful for two reasons. First, this would give model developers a set of functions they are already familiar with. This will simplify the task of utilizing the \*-AMS languages, since model developers can use \*-AMS purely as a structural modeling language, calling out predefined elements, interconnecting them and passing the parameters (data) they need. The second potential benefit of the building block library is realized by EDA vendors whose tools don't yet support \*-AMS. By keeping the building block library small, well-defined and simple, the set of functions that can be mapped into EDA tools don't have to be natively supported for \*-AMS. An EDA tool without native \*-AMS support could "map" the building blocks into elements supported in the simulator and

use the instantiations and connectivity from the \*-AMS model to reproduce the model's overall structure [9].

The key to success here is a well defined, controlled building block library than can be successfully mapped into different EDA tools. This is another task of the IBIS Macro Modeling ("IBIS 2010") Committee. This group is represented by members from Intel, Cisco, Cadence, Teraspeed Consulting, SiSoft & NCSU. I am the only student representative on this committee.

Full AMS models (as opposed to template (library) based macro-models- which is the subject of this thesis) will probably appear once all EDA vendors support AMS completely. Even then, a library such as this would be significantly advantageous to model makers in terms of time savings and ease of use.

The following section describes the structure and functionality of the aforementioned library in detail.

### 2.3 The IBIS-AMS Macro-model Hierarchy

The current, complete IBIS macro library components are listed in Appendix B. The hierarchy is defined as follows [7],[10]:

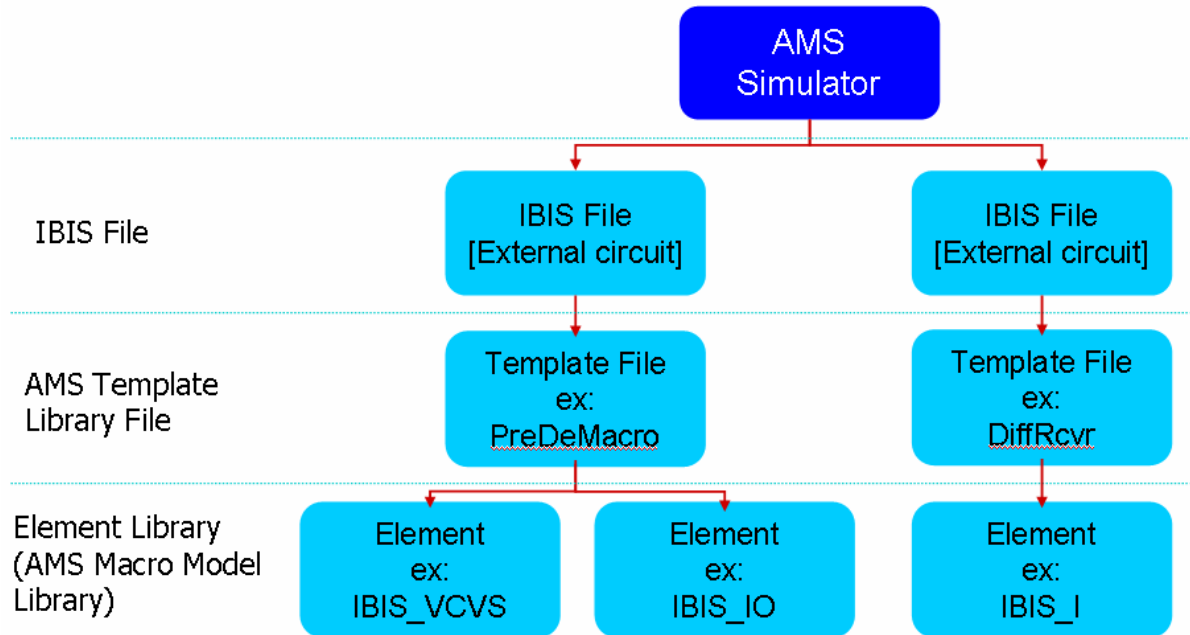


Figure 2.3: AMS Hierarchy

The [external circuit] keyword would be in the IBIS file and would specify the AMS Template file to be used. Please see the IBIS 4.1 specification [6] for more details. The template files are predefined as part of the “template library”. For example, supposing we wish to simulate a pre-emphasis buffer. Then the IBIS file would contain “[external circuit] PreDeMacro.va” pointing the simulator to the template library file, PreDeMacro.va. The PreDeMacro.va file would contain an AMS description of the pre-emphasis buffer by instantiating Macro library elements (Appendix B) from the “element library”. A more complete explanation will be given in the next chapter by means of several case studies.

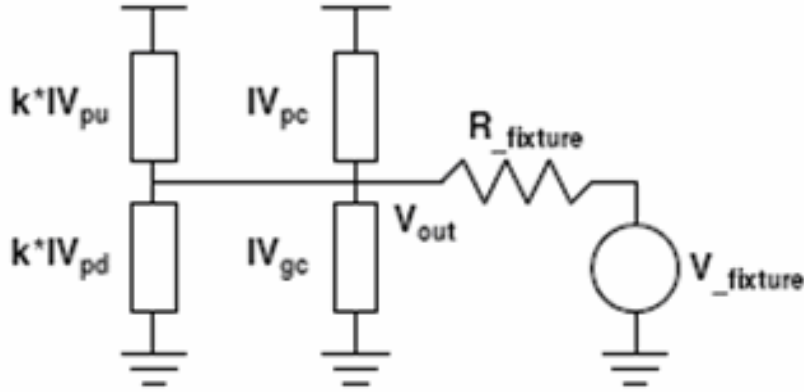
## 2.4 The Simulation algorithm

As shown in Chapter 1, an IBIS model contains several tables of data describing a buffers behavior. An IBIS compatible simulator must make sense of this information so that many load conditions may be simulated accurately using this limited behavioral description. Each simulator has its own unique algorithm for approaching the issue so that simulation time & accuracy are optimized. The algorithm used in the AMS library will be described in this section. The implementation of the algorithm in an IBIS IO buffer model is provided in Appendix E.

This algorithm makes use of the V-I curves (PullUp, PullDown, Power Clamp, GND Clamp) to calculate the steady-state (DC) characteristics of the buffer. The  $V_t$  (Rising & Falling waveforms) are used to calculate the transient behavior. This algorithm (& most others) disregards the ramp rate. Due to backward compatibility the ramp rate is still specified in IBIS files as part of the standard.

The algorithm uses the  $V_t$  curves to scale the PU & PD I-V curves with respect to time to account for the partially on/off transistor characteristics during transients. The tables that store these scaling values will be called the  $K^*$  tables. The derivation of the  $K^*$  tables are described below.

An output buffer is shown below. The  $IV^*$  values denote each respective IV curve provided. Let 'k' be the scaling coefficient used to scale the PU & PD data.



[13]

Figure 2.4: IBIS Simulation Circuit

Assume that waveform1 and waveform2 are **Rising Waveforms** obtained with two different  $V_{\text{fixture}}$  values (usually  $V_{CC}$  and GND). Since they are rising, the PU transistor will be turning on while the PD transistor will be turning off.

The node equations for the above circuit will be as follows:

$$0 = (K_{pu\_on}(t) \bullet IV_{PU1}) + IV_{PC1} - (K_{pd\_off}(t) \bullet IV_{PD1}) - IV_{GC1} - I_{out1}(t)$$

$$0 = (K_{pu\_on}(t) \bullet IV_{PU2}) + IV_{PC2} - (K_{pd\_off}(t) \bullet IV_{PD2}) - IV_{GC2} - I_{out2}(t)$$

Where,

$$I_{out} = \frac{V_{out} - V_{\text{fixture}}}{R_{\text{fixture}}}$$

Since, typically,  $V_{\text{fixture}1}=0$  &  $V_{\text{fixture}2}=V_{CC}$  (Requirement:  $V_{\text{fixture}1} < V_{\text{fixture}2}$ ; for easy programmability),

$$I_{out1}(t) = \frac{V_{out}(t) - 0}{R_{\text{fixture}}} = \frac{V_{out}(t)}{R_{\text{fixture}}} \qquad I_{out2}(t) = \frac{V_{out}(t) - V_{CC}}{R_{\text{fixture}}}$$

Therefore the only unknowns in these sets of equations are  $K_{pu\_on}(t)$  &  $K_{pd\_off}(t)$

These two unknowns can be solved for each value of  $t$  using the two equations, 2 unknowns process.

These equations yield,

$$K_{pu\_on}(t) = \frac{[I_{PD2} \bullet (I_{out1} + I_{GC1} - I_{PC1})] + [I_{PD1} \bullet (I_{PC2} - I_{GC2} - I_{out2})]}{(I_{PU1} \bullet I_{PD2}) - (I_{PD1} \bullet I_{PU2})}$$

$$K_{pd\_off}(t) = \frac{[I_{PU2} \bullet (I_{out1} + I_{GC1} - I_{PC1})] + [I_{PU1} \bullet (I_{PC2} - I_{GC2} - I_{out2})]}{(I_{PU1} \bullet I_{PD2}) - (I_{PD1} \bullet I_{PU2})}$$

Similarly for the **Falling Waveforms** (where the PU transistor is turning off while the PD transistor is turning on):

$$K_{pd\_on}(t) = \frac{[I_{PU2} \bullet (I_{out1} + I_{GC1} - I_{PC1})] + [I_{PU1} \bullet (I_{PC2} - I_{GC2} - I_{out2})]}{(I_{PU1} \bullet I_{PD2}) - (I_{PD1} \bullet I_{PU2})}$$

$$K_{pu\_off}(t) = \frac{[I_{PD2} \bullet (I_{out1} + I_{GC1} - I_{PC1})] + [I_{PD1} \bullet (I_{PC2} - I_{GC2} - I_{out2})]}{(I_{PU1} \bullet I_{PD2}) - (I_{PD1} \bullet I_{PU2})}$$

Note that in this case (typically)  $V_{fixture1}=V_{cc}$  &  $V_{fixture2}=0$ . The requirement is that  $V_{fixture1} > V_{fixture2}$  for easy programmability.

The end results of all these calculations are the following arrays:

$$K_{pd\_on}(t), K_{pd\_off}(t), K_{pu\_on}(t), K_{pu\_off}(t)$$

These arrays are computed once at simulation startup. When the simulation is in progress the original equations are used to compute  $I_{out}$ .

Going back to the original equations, for Rising and Falling cases respectively,

$$I_{out}(t) = (K_{pu\_on}(t) \bullet IV_{PU}) + IV_{PC} - (K_{pd\_off}(t) \bullet IV_{PD}) - IV_{GC}$$

$$I_{out}(t) = (K_{pu\_off}(t) \bullet IV_{PU}) + IV_{PC} - (K_{pd\_on}(t) \bullet IV_{PD}) - IV_{GC}$$

All the variables in the right (RHS) are known; therefore  $I_{out}$  can be easily calculated.

Please see Appendix E for a complete implementation.

# 3. The IBIS-to-AMS converter

## 3.1 Introduction

As shown in the pre-emphasis buffer (above) example, the model maker customizes the templates in the template library so that ‘real’ data is passed for simulation. Since the templates are in Verilog or VHDL format, the next step is to use the IBIS to AMS converter (GUI) script to convert raw IBIS files in to Verilog-AMS or VHDL-AMS format. The script is listed in Appendix D.

The customized template file, the AMS files created using the script, combined with the macro library form the input set for the AMS simulator.

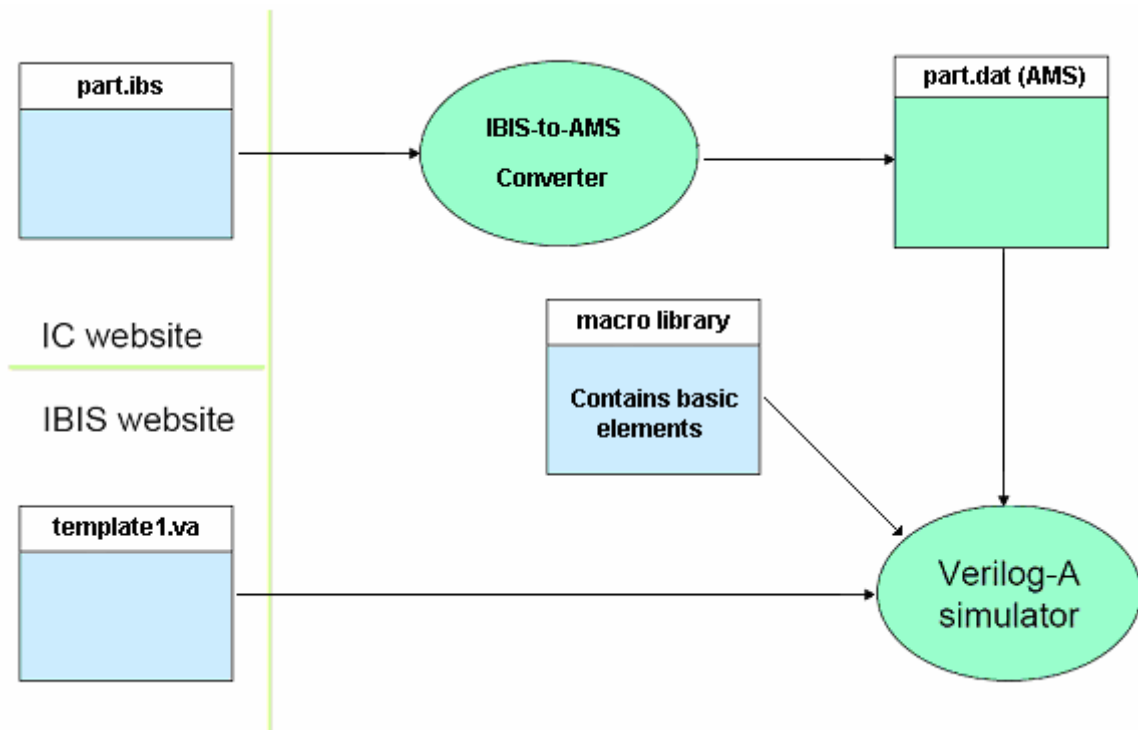
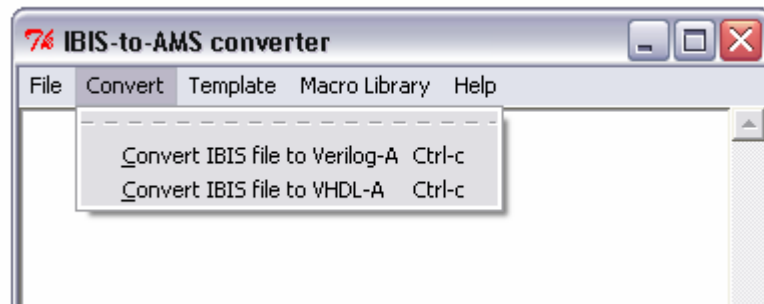


Figure 3.1: Simulation Process Diagram

The functionality of the script is explained in more detail in the following section.

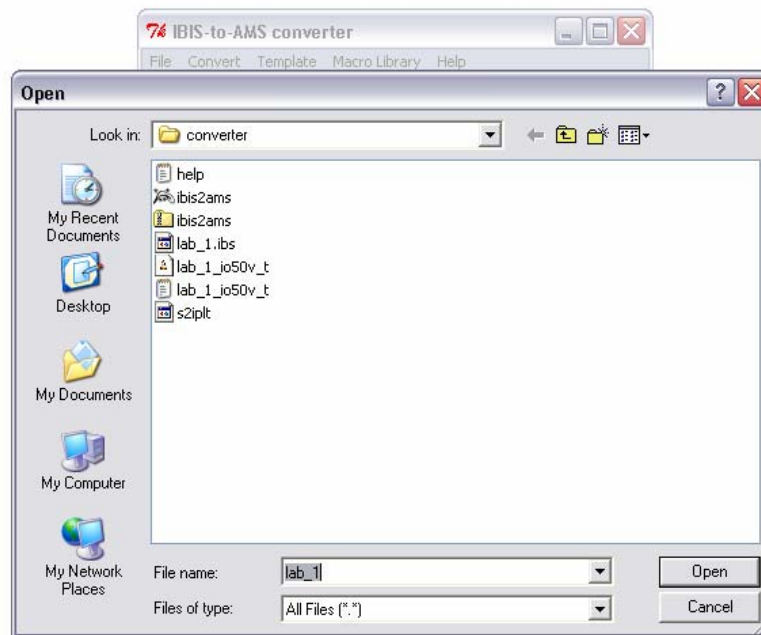
### 3.2 The IBIS-to-AMS Flow

The `ibis2ams.pl` converter script was written in Perl/Tk. It can be obtained from the IBIS-macro website. Converting IBIS files to AMS files involves using the ‘convert’ menu in the GUI. The menu contents are as shown in the following figure.



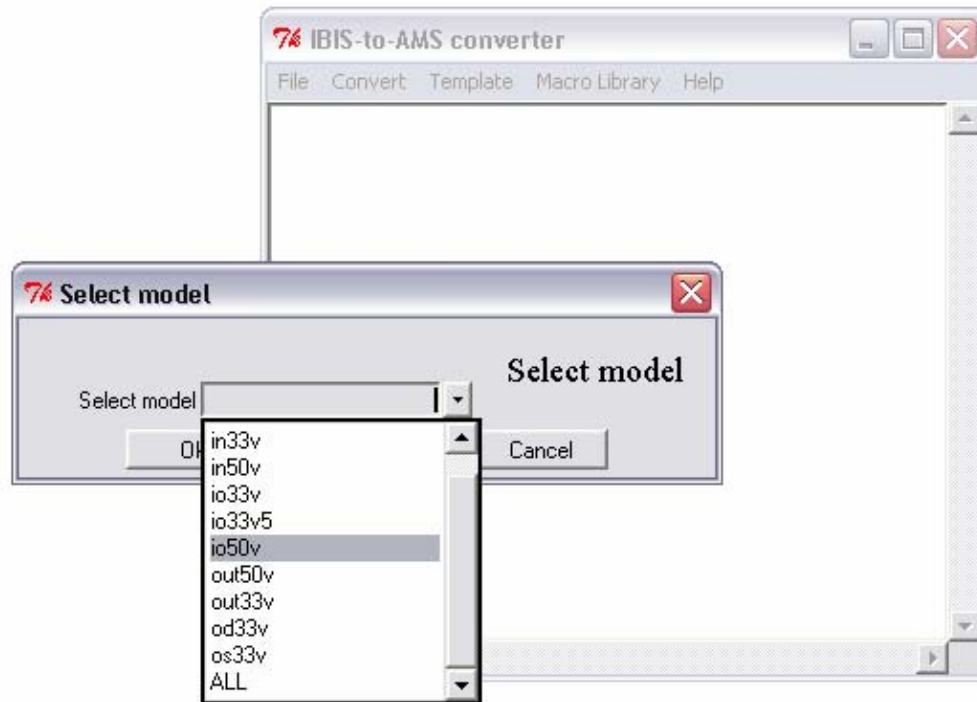
**Figure 3.2: ibis2ams Convert Menu**

For example, consider the case where we want to convert the ‘io50v’ model (typical corner) in `lab_1.ibs` to Verilog-AMS. The user would select ‘Convert IBIS file to Verilog-A’. This would bring up the following dialog to select the input IBIS file.



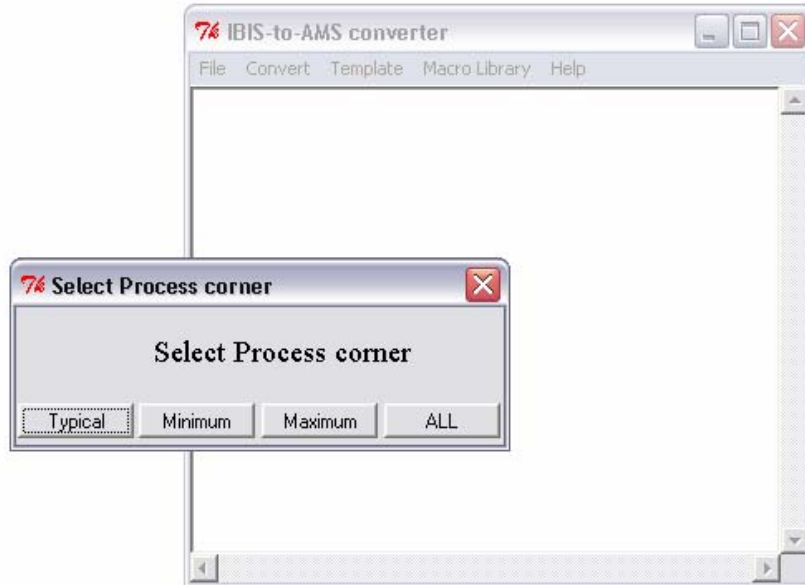
**Figure 3.3: ibis2ams Select IBIS file Dialog**

After the IBIS file is selected the following dialog queries the user to select the model (in the IBIS file) that needs conversion. Note that selecting 'ALL' would convert all the models (and corners) in the IBIS file to AMS. For this example, 'io50v' was selected.



**Figure 3.4: ibis2ams Select model to convert**

The next dialog asks the user to select the process corner (typical, minimum or maximum) for data extraction.

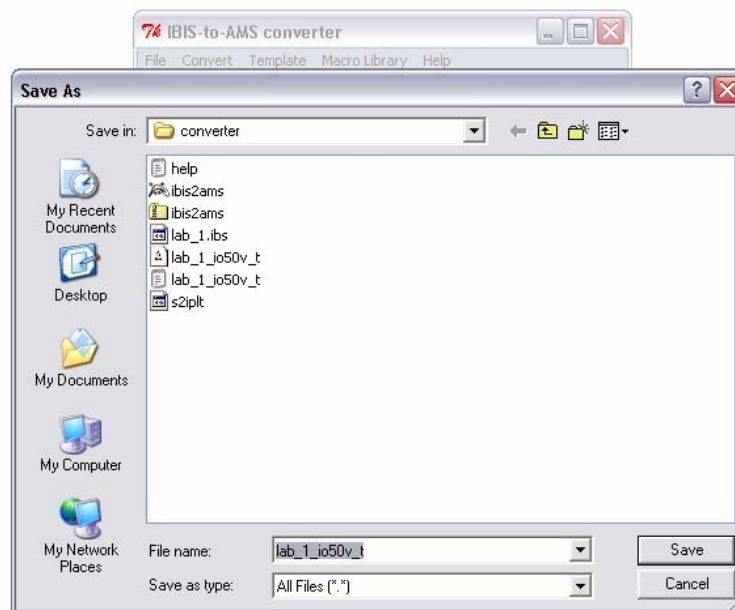


**Figure 3.5: ibis2ams Select process corner**

The next dialog asks the user to provide the AMS output file name. A default file name is provided by the program with the following format:

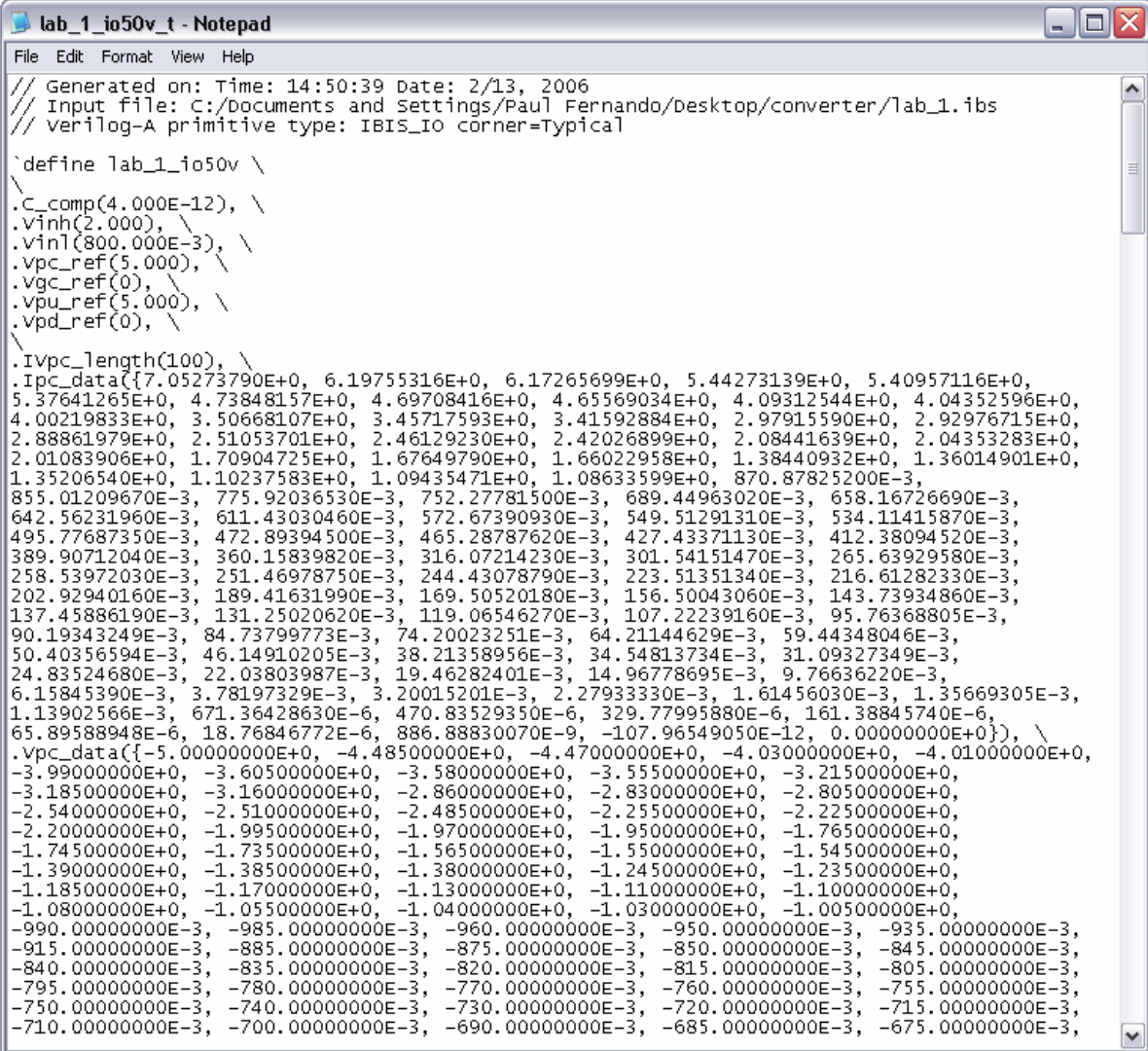
<IBIS\_file\_name>\_<Model\_name>\_<corner>.dat

Where <corner> is 't' (typical), 'n' (minimum) or 'x' (maximum)



**Figure 3.6: ibis2ams Select output file name**

A part of the output Verilog-AMS file is shown below.

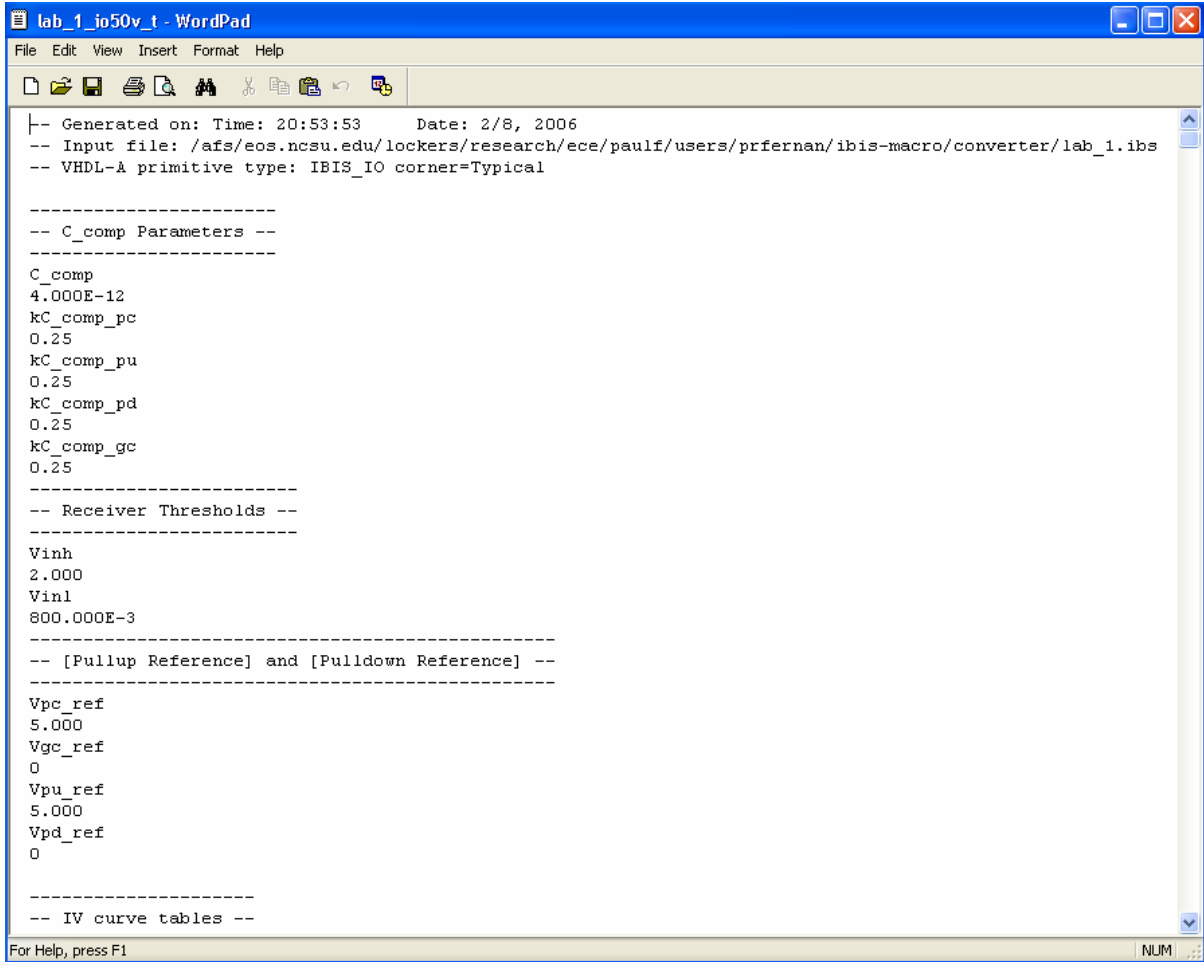


```
lab_1_io50v_t - Notepad
File Edit Format View Help
// Generated on: Time: 14:50:39 Date: 2/13, 2006
// Input file: C:/Documents and Settings/Paul Fernando/Desktop/converter/lab_1.ibs
// verilog-A primitive type: IBIS_IO corner=Typical

`define lab_1_io50v \
.C_comp(4.000E-12), \
.vinh(2.000), \
.vinl(800.000E-3), \
.vpc_ref(5.000), \
.vgc_ref(0), \
.vpu_ref(5.000), \
.vpd_ref(0), \
.Ivpc_length(100), \
.Ipc_data({7.05273790E+0, 6.19755316E+0, 6.17265699E+0, 5.44273139E+0, 5.40957116E+0,
5.37641265E+0, 4.73848157E+0, 4.69708416E+0, 4.65569034E+0, 4.09312544E+0, 4.04352596E+0,
4.00219833E+0, 3.50668107E+0, 3.45717593E+0, 3.41592884E+0, 2.97915590E+0, 2.92976715E+0,
2.88861979E+0, 2.51053701E+0, 2.46129230E+0, 2.42026899E+0, 2.08441639E+0, 2.04353283E+0,
2.01083906E+0, 1.70904725E+0, 1.67649790E+0, 1.66022958E+0, 1.38440932E+0, 1.36014901E+0,
1.35206540E+0, 1.10237583E+0, 1.09435471E+0, 1.08633599E+0, 870.87825200E-3,
855.01209670E-3, 775.92036530E-3, 752.27781500E-3, 689.44963020E-3, 658.16726690E-3,
642.56231960E-3, 611.43030460E-3, 572.67390930E-3, 549.51291310E-3, 534.11415870E-3,
495.77687350E-3, 472.89394500E-3, 465.28787620E-3, 427.43371130E-3, 412.38094520E-3,
389.90712040E-3, 360.15839820E-3, 316.07214230E-3, 301.54151470E-3, 265.63929580E-3,
258.53972030E-3, 251.46978750E-3, 244.43078790E-3, 223.51351340E-3, 216.61282330E-3,
202.92940160E-3, 189.41631990E-3, 169.50520180E-3, 156.50043060E-3, 143.73934860E-3,
137.45886190E-3, 131.25020620E-3, 119.06546270E-3, 107.22239160E-3, 95.76368805E-3,
90.19343249E-3, 84.73799773E-3, 74.20023251E-3, 64.21144629E-3, 59.44348046E-3,
50.40356594E-3, 46.14910205E-3, 38.21358956E-3, 34.54813734E-3, 31.09327349E-3,
24.83524680E-3, 22.03803987E-3, 19.46282401E-3, 14.96778695E-3, 9.76636220E-3,
6.15845390E-3, 3.78197329E-3, 3.20015201E-3, 2.27933330E-3, 1.61456030E-3, 1.35669305E-3,
1.13902566E-3, 671.36428630E-6, 470.83529350E-6, 329.77995880E-6, 161.38845740E-6,
65.89588948E-6, 18.76846772E-6, 886.88830070E-9, -107.96549050E-12, 0.00000000E+0}), \
.Vpc_data({-5.00000000E+0, -4.48500000E+0, -4.47000000E+0, -4.03000000E+0, -4.01000000E+0,
-3.99000000E+0, -3.60500000E+0, -3.58000000E+0, -3.55500000E+0, -3.21500000E+0,
-3.18500000E+0, -3.16000000E+0, -2.86000000E+0, -2.83000000E+0, -2.80500000E+0,
-2.54000000E+0, -2.51000000E+0, -2.48500000E+0, -2.25500000E+0, -2.22500000E+0,
-2.20000000E+0, -1.99500000E+0, -1.97000000E+0, -1.95000000E+0, -1.76500000E+0,
-1.74500000E+0, -1.73500000E+0, -1.56500000E+0, -1.55000000E+0, -1.54500000E+0,
-1.39000000E+0, -1.38500000E+0, -1.38000000E+0, -1.24500000E+0, -1.23500000E+0,
-1.18500000E+0, -1.17000000E+0, -1.13000000E+0, -1.11000000E+0, -1.10000000E+0,
-1.08000000E+0, -1.05500000E+0, -1.04000000E+0, -1.03000000E+0, -1.00500000E+0,
-990.00000000E-3, -985.00000000E-3, -960.00000000E-3, -950.00000000E-3, -935.00000000E-3,
-915.00000000E-3, -885.00000000E-3, -875.00000000E-3, -850.00000000E-3, -845.00000000E-3,
-840.00000000E-3, -835.00000000E-3, -820.00000000E-3, -815.00000000E-3, -805.00000000E-3,
-795.00000000E-3, -780.00000000E-3, -770.00000000E-3, -760.00000000E-3, -755.00000000E-3,
-750.00000000E-3, -740.00000000E-3, -730.00000000E-3, -720.00000000E-3, -715.00000000E-3,
-710.00000000E-3, -700.00000000E-3, -690.00000000E-3, -685.00000000E-3, -675.00000000E-3,
```

Figure 3.7: Sample Verilog-AMS output file

The VHDL-AMS file of the same model would look as follows.



```
lab_1_io50v_t - WordPad
File Edit View Insert Format Help
|--- Generated on: Time: 20:53:53      Date: 2/8, 2006
-- Input file: /afs/eos.ncsu.edu/lockers/research/ece/paulf/users/prfernan/ibis-macro/converter/lab_1.ibs
-- VHDL-A primitive type: IBIS_IO corner=Typical

-----
-- C_comp Parameters --
-----
C_comp
4.000E-12
kC_comp_pc
0.25
kC_comp_pu
0.25
kC_comp_pd
0.25
kC_comp_gc
0.25
-----
-- Receiver Thresholds --
-----
Vinh
2.000
Vinl
800.000E-3
-----
-- [Pullup Reference] and [Pulldown Reference] --
-----
Vpc_ref
5.000
Vgc_ref
0
Vpu_ref
5.000
Vpd_ref
0
-----
-- IV curve tables --
-----
For Help, press F1
```

Figure 3.8: Sample VHDL-AMS output file

## 4. Results

### 4.1 Introduction

Three case studies were performed to prove the accuracy, scalability and overall viability of the methodology.

1. Output Buffer: A standard output buffer was simulated in hspice (transistor level) and AMS and compared.
2. Pre-emphasis Buffer: This buffer can be described in traditional IBIS; however, the tap delay would be hard coded. This example is based on the code written by Arpad Muranyi. He had performed the simulation in Verilog-AMS; I translated the code to VHDL-AMS for Dolphin SMASH compatibility.
3. Equalized Receiver: This buffer cannot be modeled in traditional IBIS. This example was done to prove the AMS methodology's scalability.

## 4.2 Case study I: Output Buffer

A standard output buffer that's completely compatible with traditional IBIS will be used in this example for proof of concept. The buffer is an Intel buffer provided by Michael Mirmak and includes the pre-driver & all parasitics. The ESD diodes were removed due to lack of a device model.

The flow used to create the AMS file describing the output buffer is given below. The example will use SMASH VHDL-AMS as the simulation tool.

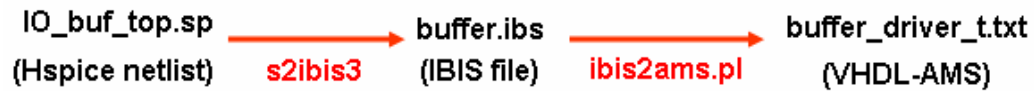


Figure 4.1 Conversion Flow

First, the Intel hspice netlist is converted to an IBIS file using the NCSU s2ibis utility. (All the simulation files for this case study are provided in Appendix G)

Once the IBIS file is created, it must be converted to VHDL-AMS format using the ibis2ams converter provided in Appendix D (and described in Chapter 3). This output file will be used for the SMASH VHDL-AMS simulations.

The following circuit is simulated in SMASH. The instance X1 will be represented by the VHDL-AMS file which was extracted above.

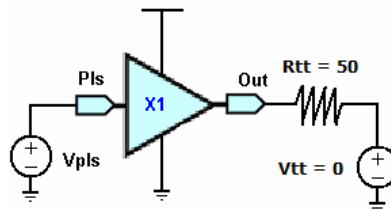


Figure 4.2 Test Circuit

The SMASH GUI for the simulation is shown below:

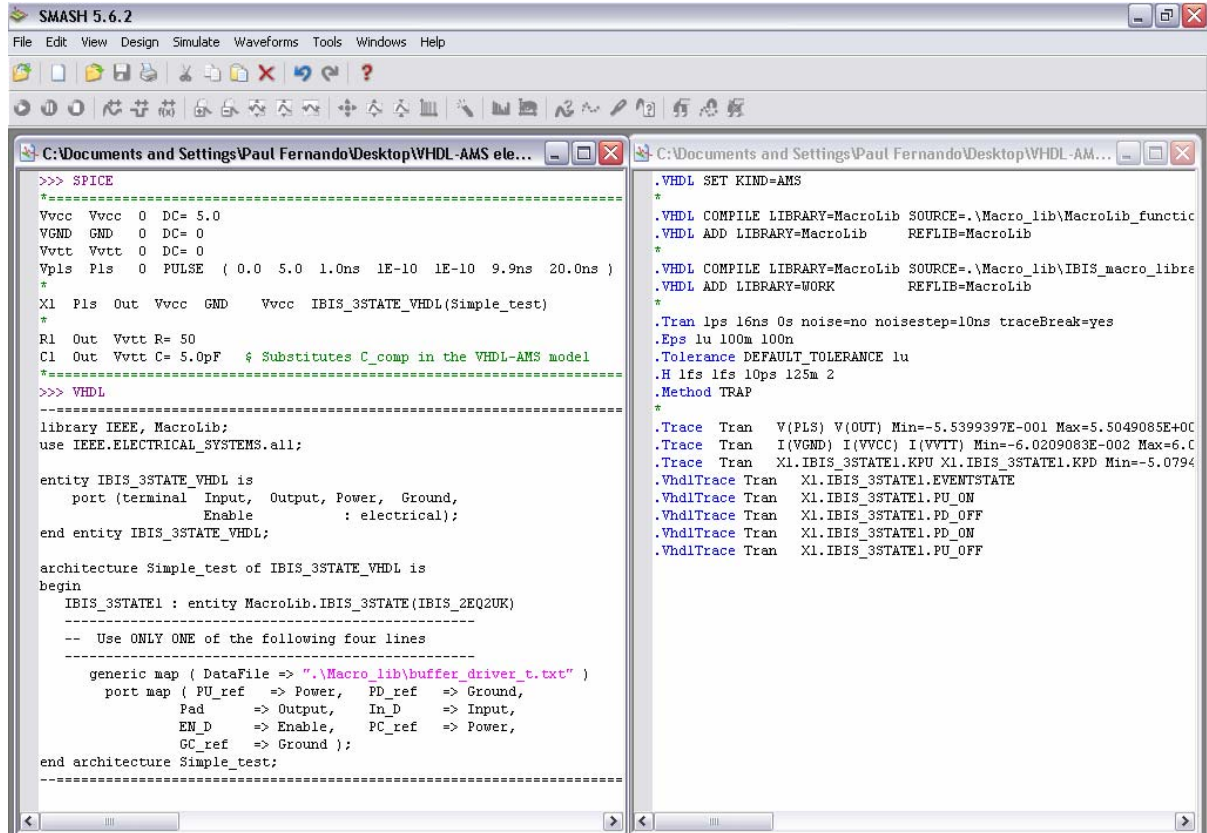
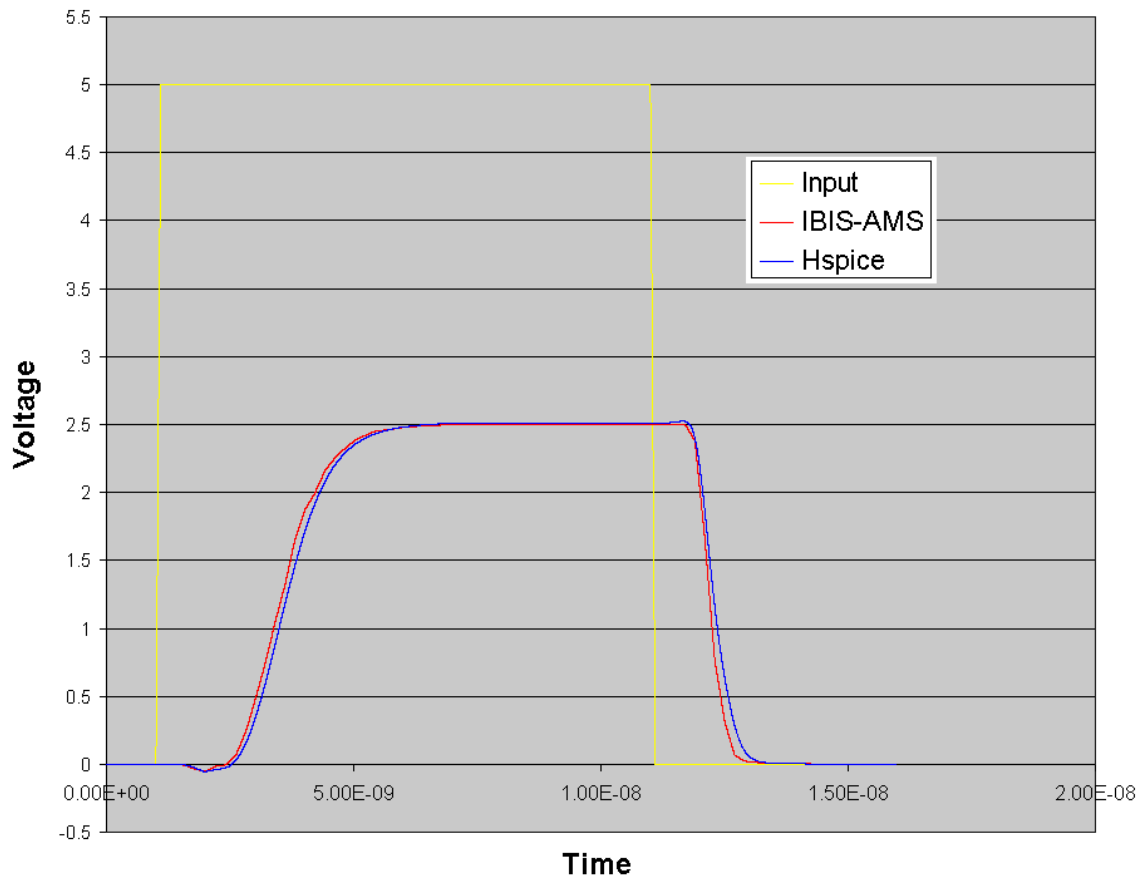


Figure 4.3 SMASH GUI

The complete circuit description is given in the left plane. Notice the division between the spice & VHDL sections. The Spice section describes the circuit given above and the VHDL section defines the buffer sub-circuit (it also calls the VHDL-AMS data file which was extracted). The right plane contains the simulator 'use' file which provides simulator directives.

The results of the simulation are given below, plotted against the transistor level Hspice results:



**Figure 4.4 Output Comparison**

The variation is clearly due to an underestimation of  $C_{comp}$  in the IBIS model. Note that s2ibis doesn't calculate  $C_{comp}$ . It simply defaults to 5pF leaving the model maker to tweak the value. The matching between the transistor level hspice simulation & the SMASH AMS simulation is quite good. The model-maker simply has to tweak the  $C_{comp}$  value to get a better match.

### 4.3 Case study II: Pre-emphasis Buffer

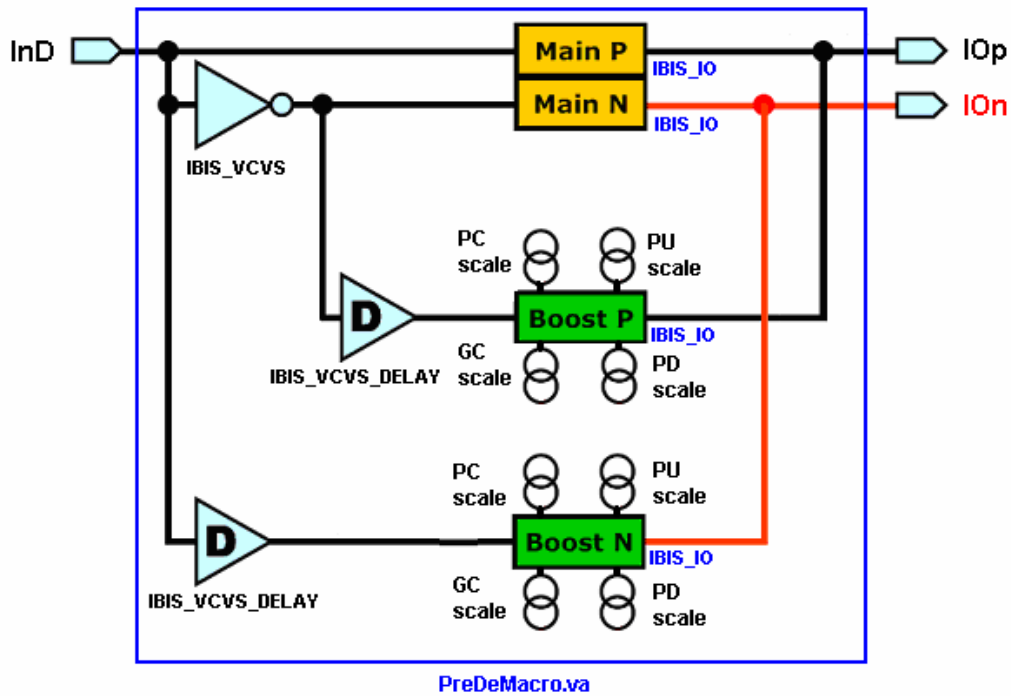
A T-line (and package parasitics) presents LPF behavior. The high frequency components are attenuated due to the T-lines physical characteristics. A high-frequency pulse must have sufficient amplitude at the receiver for valid detection. The transmitter equalization technique pre-emphasizes the high-frequency components of a signal and minimizes the t-line attenuation problem. Pre-emphasis is done by means of a simple digital filter. A single tap pre-emphasis buffer as the one shown below would have four static states, depending on the states of the main & boost circuits. At a transition, both circuits are switched on for one cycle to emphasize the rising or falling edge. At other points in time where there is no transition the boost circuit remains off. Thus, its behavioral model would have to keep track of previous data that is been input to the buffer. Modeling this device in IBIS using current techniques would require the use of 'tricks' &/or the [driver schedule] keyword making it a good candidate for macro-modeling. [12]

IBIS versions before 4.1 could not model a pre-emphasis buffer at all due to its output dependency on previous output values. [driver schedule] was added in IBIS 4.1 but the drawback was that the tap delay had to be hard coded in the IBIS file. Therefore, each time the input clock frequency was varied, the user had to manually edit the IBIS file to reflect the change in tap delay. Using the new methodology, the user can simply supply the delay as a parameter through the main simulation file.

A schematic of the template is given below for ease of understanding. It can be seen that the 2-tap pre-emphasis buffer template would have to contain the following elements:

1. An inverter
2. Two ideal delays

3. Four IBIS\_OUTPUT (Appendix B) models for the drivers
4. Eight IBIS\_I (current sources) to scale the boost buffers currents – The current scaling for the boost circuit is achieved by scaling the supply currents provided to the boost buffers. The eight supply nodes are: both (P & N) boost buffers pullup\_ref, powerclamp\_ref, pulldown\_ref & groundclamp\_ref; hence the need for 8 current sources.



Original Diagram borrowed from M. Mirmak

**Figure 4.5 Pre-emphasis Buffer Schematic**

The VHDL-AMS template file which describes the Pre-emphasis buffer is provided below. Note that the parameters ‘ScaleBoost’ and ‘TapDelay’ can be provided by the main simulation file. Parameter passing in this manner provides a great deal of flexibility which cannot be found in IBIS alone.

```

>>> VHDL
-----
library IEEE, MacroLib;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity IBIS_PreDeMacro_VHDL is
  generic (ScaleBoost: real := -0.9;
           TapDelay: real := 10.0e-9);
  port (terminal InD, IOp, IOn, PCref, PUref, PDref, GCref, En : electrical);
end entity IBIS_PreDeMacro_VHDL;

architecture Simple_test of IBIS_PreDeMacro_VHDL is

terminal Dref, InNM, InPB, InNB : electrical;
terminal PUrefPB, PDrefPB, PCrefPB, GCrefPB : electrical;
terminal PUrefNB, PDrefNB, PCrefNB, GCrefNB : electrical;

begin
  Dig1 : entity MacroLib.IBIS_V ----- Voltage supply for buffers
    generic map ( Vdc => 1.0 )
    port map ( p => Dref, n => PDref);

  Inv1 : entity MacroLib.IBIS_VCVS ----- Inverter
    port map ( p => Dref, n => InNM, ps => InD, ns => PDref);

  PosM : entity MacroLib.IBIS_IO(IBIS_2EQ2UK) ----- Main P
    generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt" )
    port map ( PU_ref => PUref,
              PD_ref => PDref, Pad => IOp,
              In_D => InD, EN_D => En,
              PC_ref => PCref, GC_ref => GCref );

  NegM : entity MacroLib.IBIS_IO(IBIS_2EQ2UK) ----- Main N
    generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt" )
    port map ( PU_ref => PUref,
              PD_ref => PDref, Pad => IOn,
              In_D => InNM, EN_D => En,
              PC_ref => PCref, GC_ref => GCref );

```

Figure 4.6 Pre-emphasis Buffer VHDL-AMS code (Part 1)

```

-----TAP
Dly1 : entity MacroLib.IBIS_VCVS_DELAY
  generic map ( TD => TapDelay )
  port map ( p => InNB, n => PDref, ps => InD, ns => PDref);
Dly2 : entity MacroLib.IBIS_VCVS_DELAY
  generic map ( TD => TapDelay )
  port map ( p => InPB, n => PDref, ps => InNM, ns => PDref);
PosB : entity MacroLib.IBIS_IO(IBIS_2EQ2UK)
  generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt"
    ,kI_pc => ScaleBoost, kI_pu => ScaleBoost,
    --
    -- kI_pd => ScaleBoost, kI_gc => ScaleBoost
  )
  port map ( PU_ref => PUFrefPB,
    PD_ref => PDrefPB, Pad => IOp,
    In_D => InPB, EN_D => En,
    PC_ref => PCrefPB, GC_ref => GCrefPB );
NegB : entity MacroLib.IBIS_IO(IBIS_2EQ2UK)
  generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt"
    ,kI_pc => ScaleBoost, kI_pu => ScaleBoost,
    --
    -- kI_pd => ScaleBoost, kI_gc => ScaleBoost
  )
  port map ( PU_ref => PUFrefNB,
    PD_ref => PDrefNB, Pad => IOn,
    In_D => InNB, EN_D => En,
    PC_ref => PCrefNB, GC_ref => GCrefNB );

IpcP : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => PCref, n => IOp, ps => PCref, ns => PCrefPB);
IpuP : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => PUFref, n => IOp, ps => PUFref, ns => PUFrefPB);
IpdP : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => PDref, n => IOp, ps => PDref, ns => PDrefPB);
IgcP : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => GCref, n => IOp, ps => GCref, ns => GCrefPB);

IpcN : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => PCref, n => IOn, ps => PCref, ns => PCrefNB);
IpuN : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => PUFref, n => IOn, ps => PUFref, ns => PUFrefNB);
IpdN : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => PDref, n => IOn, ps => PDref, ns => PDrefNB);
IgcN : entity MacroLib.IBIS_CCCS
  generic map (Scale => ScaleBoost)
  port map ( p => GCref, n => IOn, ps => GCref, ns => GCrefNB);
end architecture Simple_test;
-----

```

**Ideal delays (for boost input)**

**Boost P**

**Boost N**

**Current scaling for Boost tap**

Figure 4.7 Pre-emphasis Buffer VHDL-AMS code (Part 2)

The following circuit was implemented in SMASH. The subcircuit X1 is described in VHDL-AMS as shown above.

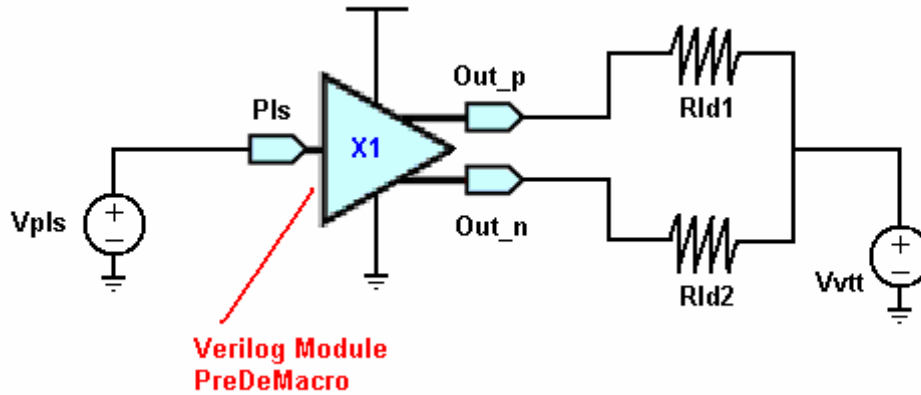


Figure 4.8 Pre-emphasis Buffer Test Circuit

The SMASH GUI is shown below.

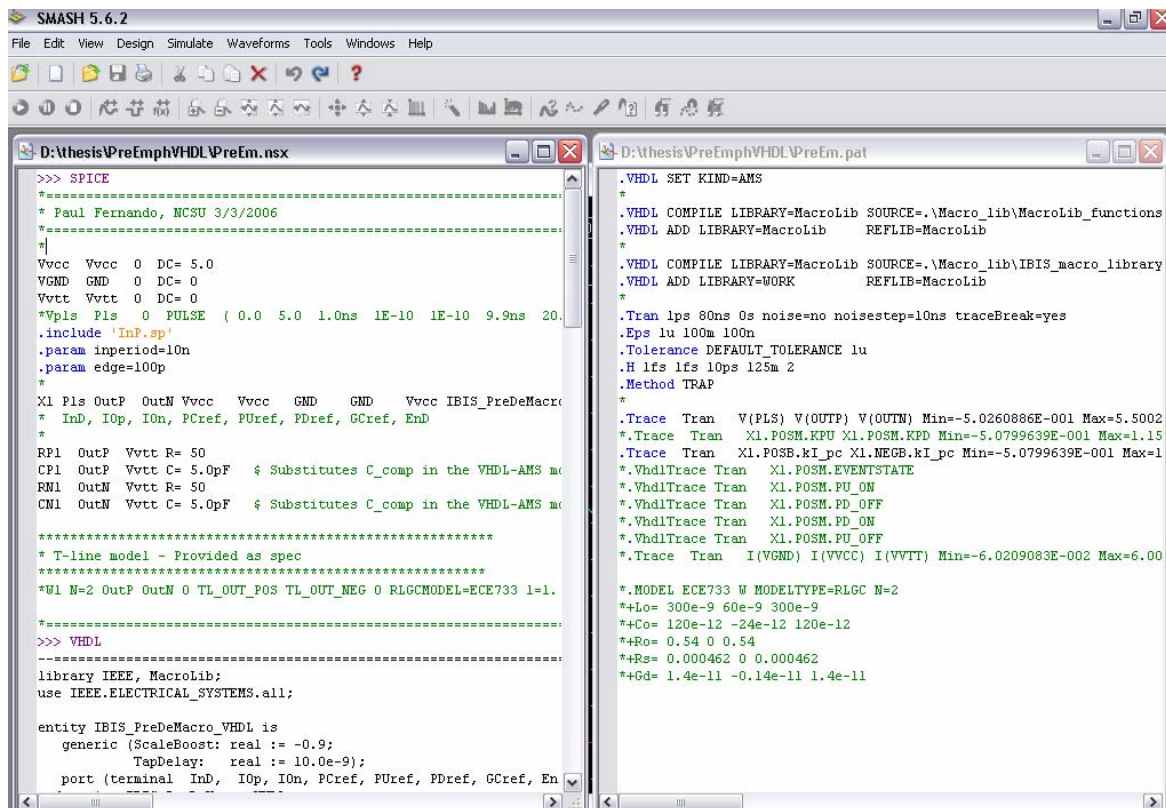


Figure 4.9: Pre-emphasis Buffer SMASH GUI

The SMASH VHDL-AMS simulation results are shown below for an arbitrary input stream. Depending on the value chosen for 'ScaleBoost', the tap will emphasize the signal at switching points.

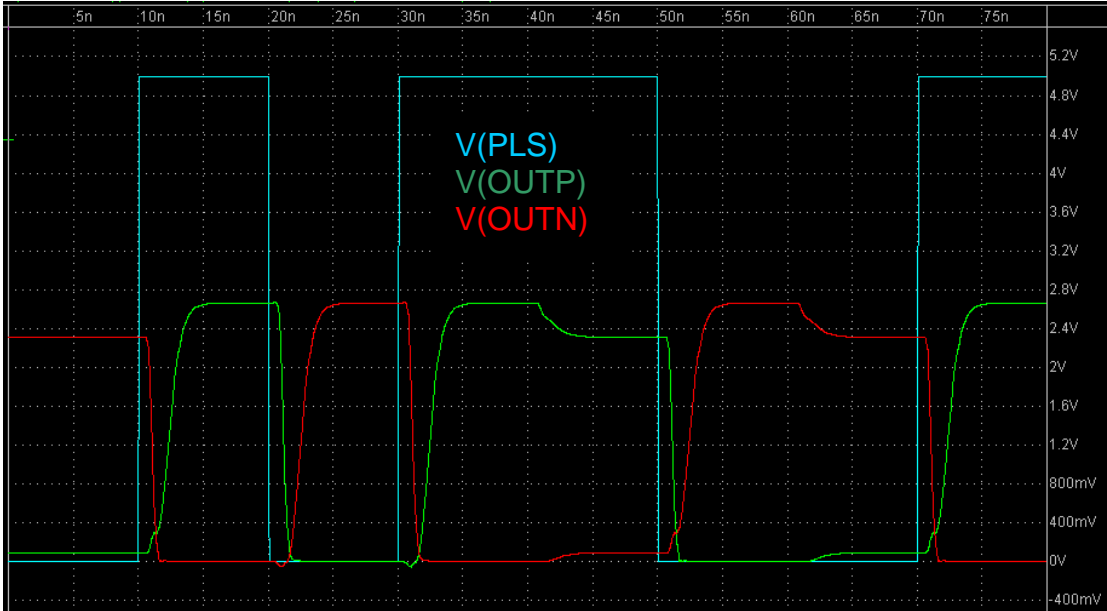


Figure 4.10: Pre-emphasis Buffer SMASH Output

#### 4.4 Case study III: Equalized Receiver

The schematic of the receiver is shown below. This example doesn't make use of IBIS data. It simply provides a behavioral description of a differential input pair with receiver end equalization and amplification.

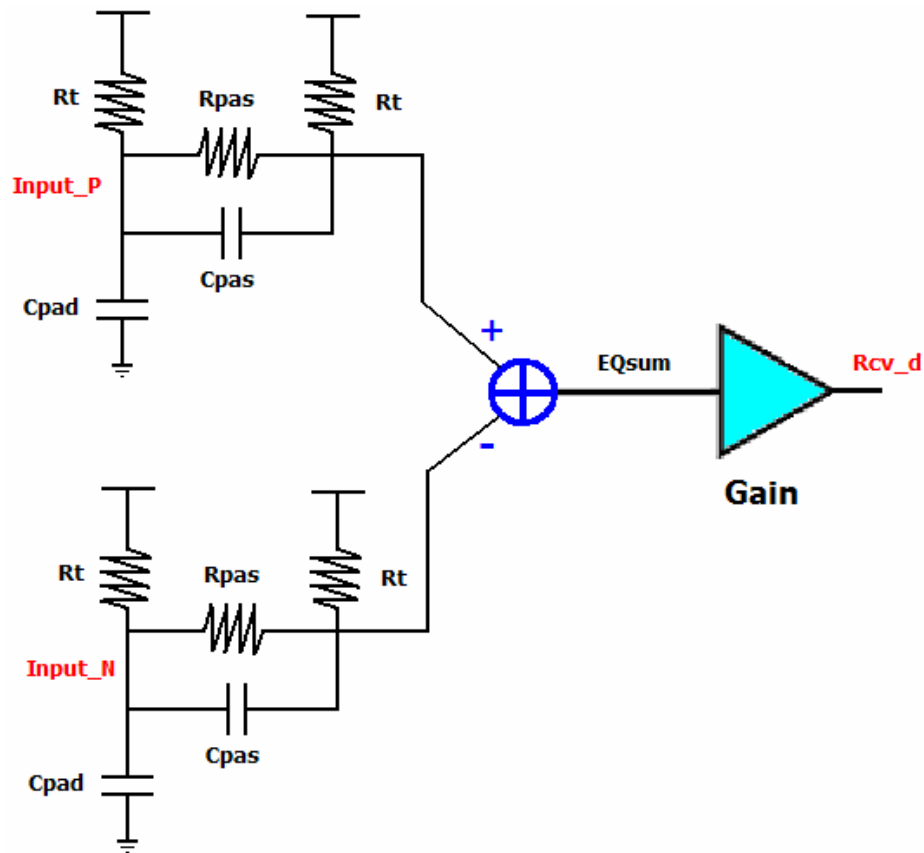


Figure 4.11: Receiver Schematic

This receiver includes a passive equalizer and an amplifier. The equalization RC is determined by Rpas & Cpas. Gain is equal to  $(R_{pas} + R_{t2}) * gain / R_{t2}$

The VHDL-AMS code for the above circuit is given below. Note that the resistor, capacitor and gain settings are all programmable.

```

>>> VHDL
library IEEE, MacroLib;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity Eq_Rcvr is
  generic (Cpad: real := 1.0e-12;    Rt: real := 50.0;
           pas_on: real := 1.0;     Rpas: real := 130.0;
           Cpas: real := 0.5e-12;   Rt2: real := 1000.0;
           gain: real := 4.0);
  port (terminal PC_ref, GC_ref, Input_p, Input_n, Rcv_D : electrical)
end entity Eq_Rcvr;

architecture Simple_test of Eq_Rcvr is
  terminal EQ_p, EQ_n, EQSUM : electrical;

begin

  RT1P : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => Input_p, n => PC_ref);
  CP : entity MacroLib.IBIS_C
    generic map ( Cval => Cpad )
    port map ( p => Input_p, n => PC_ref);
  RPASP : entity MacroLib.IBIS_R
    generic map ( Rval => Rpas )
    port map ( p => Input_p, n => EQ_p);
  CPASP : entity MacroLib.IBIS_C
    generic map ( Cval => Cpas*pas_on )
    port map ( p => Input_p, n => EQ_p);
  RT2P : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => EQ_p, n => PC_ref);

  RT1N : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => Input_n, n => PC_ref);
  CN : entity MacroLib.IBIS_C
    generic map ( Cval => Cpad )
    port map ( p => Input_n, n => GC_ref);
  RPASN : entity MacroLib.IBIS_R
    generic map ( Rval => Rpas )
    port map ( p => Input_n, n => EQ_n);
  CPASN : entity MacroLib.IBIS_C
    generic map ( Cval => Cpas*pas_on )
    port map ( p => Input_n, n => EQ_n);
  RT2N : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => EQ_n, n => PC_ref);

  -- This voltage source derives the differential output of the 2 passive
  -- and then amplifies it by the gain parameter.

  EQSUM1 : entity MacroLib.IBIS_VCVS_SUM
    port map ( p =>EQSUM,      n =>GC_ref,
              ps1 =>EQ_p,     ns1 =>GC_ref,
              ps2 =>GC_ref,   ns2 =>EQ_n);
  EQOUT : entity MacroLib.IBIS_VCVS
    generic map ( Scale => (Rpas + Rt2)*(gain/Rt2) )
    port map ( p =>Rcv_d,      n =>GC_ref,
              ps =>EQSUM,     ns =>GC_ref);
end architecture Simple_test;

```

Parameters

IN\_P Passive Equalizer

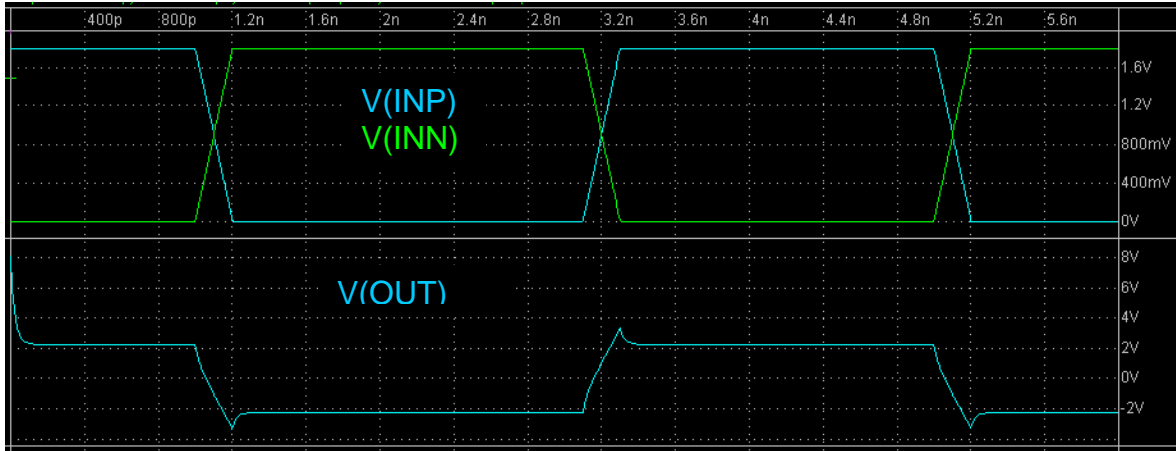
IN\_N Passive Equalizer

Summer

Gain Stage

Figure 4.12: Receiver VHDL-AMS code

The resultant waveform for an ideal differential input is provided below.



**Figure 4.13: Receiver SMASH Output**

The Transmission line model in SMASH isn't complete yet, therefore I couldn't simulate a 'real' condition.

## 5. Conclusions & Future Research

### 5.1 Conclusions

As the operating frequencies & complexity of I/O buffers increased, many model makers and/or users leave IBIS behind in favor of SPICE models; since IBIS is inaccurate or unable to model these advanced buffers. Since most semiconductor vendors support their models with a single vendor's version of SPICE, this trend brings the industry back towards a single EDA vendor solution, which is what IBIS was designed to prevent. In an effort to relinquish these shortcomings, in IBIS version 4.1, multi-lingual model extensions were added to IBIS so that it could recognize Berkeley-SPICE, VHDL-AMS and Verilog-AMS files.

A Macro model library was developed by the IBIS-macro committee to model state of the art buffers by combining IBIS & AMS languages. Several templates were written to model state of the art IO buffers which couldn't be modeled by traditional IBIS. A GUI based tool was developed (ibs2ams) and extensively tested to convert standard IBIS files into AMS formats. Combining the Macro library, the conversion tool and templates, several simulations were done to prove the methods accuracy, scalability and overall viability. As a byproduct, we have a programmable IBIS simulator written in AMS which can be updated as we choose.

GUI tools were created in Perl/Tk for s2iplt (IBIS wave plotting utility) & s2ibis (Spice to IBIS converter).

## 5.2 Future Research

As mentioned in the conclusion section above, a result of this effort is that we end up with a completely programmable IBIS simulator in AMS. One line of action is to attempt to solve the SSN issues with IBIS. The first step would probably be to implement BIRD 95 in the AMS library simulation algorithm [21].

Another line of action would be to create more template files; for LVDS, impedance compensated buffers, true differential, buffers with deterministic jitter insertion, Decision Feedback Equalization, FIR equalized receiver models etc. Running accuracy studies for the above template files would also be a useful avenue.

# REFERENCES

- [1] Hall, Hall, McCall, "High speed Digital Systems Design", IEEE Press
- [2] P. Fernando, "IBIS Modeling", Presentation at Analog Devices, Boston MA
- [3] <http://www.intel.com/technology/silicon/mooreslaw/>
- [4] M. Casamayor, "A first approach to IBIS models", Analog Devices Application note.
- [5] <http://www.freelists.org/archives/IBIS-macro/>
- [6] IBIS Version 4.1 (I/O Buffer Information Specification), January 2004
- [7] Muranyi, "IBIS 4.1 Macros for simulator independent models", DAC IBIS summit, June 2005.
- [8] D. Telian, "Modeling complex I/O with IBIS 4.1", IBIS summit, January 2005.
- [9] NCSU s2ibis3 manual.
- [10] Westerhoff, "IBIS 4.1 Macromodel Library for simulator independent modeling", Designcon East IBIS summit, September 2005.
- [11] <http://www.eetimes.com/news/design/columns/eda/showArticle.jhtml?articleID=17407954>
- [12] ECE733 class project, [http://www4.ncsu.edu/~prfernan/Xtra\\_files/733%20proj2.doc](http://www4.ncsu.edu/~prfernan/Xtra_files/733%20proj2.doc)  
IBIS Modeling Cookbook v4.0, <http://www.eda.org/pub/ibis/cookbook/cookbook-v4.pdf>
- [13] Arpad Muranyi, "Accuracy of IBIS models with reactive loads", DesignCon 2006, Santa Clara, CA
- [14] Giacotto & Muranyi, "A VHDL-AMS buffer model using IBIS v3.2 data", DesignConEast 2003, Marlborough, MA
- [15] NCSU ECE744 Notes, Spring 2005, Dr. Bob Evans

- [16] Ian Dodd , “IBIS - Addressing Challenges in Behavior and Measurement”, DesignCon, February 2006.
- [17] Arpad Muranyi, “Introduction to the IBIS Macro Model Library”, DesignCon, February 2006.
- [18] Todd Westerhoff, “Current Status - IBIS 4.1 Macro Library for Simulator Independent Modeling”, DesignCon, February 2006.
- [19] Michael Mirmak, “IBIS and Behavioral Modeling”, DesignCon, February 2006.
- [20] Lance Wang and ZhangMin Zhong, “Using IBIS for SI Analysis”, Asian IBIS Summit, Shenzhen, China, December 2005
- [21] Yang, Park, et al, “Multi-buffer SSN Simulation using BIRD95”, DAC 2005, Anaheim, CA

# Appendices

# Appendix A

## IBIS Macro Library Elements

### Basic Elements

Table A.1: Basic Elements

	Fixed	Voltage Controlled	Current Controlled
R	IBIS_R	IBIS_VCR	IBIS_CCR
L	IBIS_L	IBIS_VCL	IBIS_CCL
C	IBIS_C	IBIS_VCC	IBIS_CCC

### Sources

Table A.2: Sources

	Fixed	Voltage Controlled	Current Controlled	Voltage Controlled with Delay	Current Controlled with Delay
Voltage Source	IBIS_V	IBIS_VCVS	IBIS_CCVS	IBIS_VCVS_DELAY	IBIS_CCVS_DELAY
Current Source	IBIS_I	IBIS_VCCS	IBIS_CCCS	IBIS_VCCS_DELAY	IBIS_CCCS_DELAY

## Operators

Table A.3: Operators

	Current Controlled Current (with scaler)	Voltage Controlled Voltage (with scaler)
Adders	IBIS_CCCS_SUM	IBIS_VCVS_SUM
Multipliers	IBIS_CCCS_MULT	IBIS_VCVS_MULT
Dividers	IBIS_CCCS_DIV	IBIS_VCVS_DIV
	Voltage Controlled Current (with scaler)	Current Controlled Voltage (with scaler)
Adders	IBIS_VCCS_SUM	IBIS_CCVS_SUM
Multipliers	IBIS_VCCS_MULT	IBIS_CCVS_MULT
Dividers	IBIS_VCCS_DIV	IBIS_CCVS_DIV

## Active Devices

Table A.4: Active Devices

IBIS Input buffer	IBIS_INPUT
IBIS Output buffer	IBIS_OUTPUT
IBIS Input/Output buffer	IBIS_IO
IBIS Tri-state buffer	IBIS_3STATE
IBIS Open-sink buffer	IBIS_OPENSINK
IBIS I/O Open-sink buffer	IBIS_IO_OPENSINK
IBIS Open-source buffer	IBIS_OPENSOURCE
IBIS I/O Open-source buffer	IBIS_IO_OPENSOURCE

## Other Models

Table A.5: Other Models

Inductive Coupler	IBIS_K
Transmission Line	IBIS_T

# Appendix B

## IBIS to Verilog-A converter script

This script was written to convert standard IBIS files to AMS format.

```
#!/usr/local/bin/perl
#
# Paul Fernando NCSU, IBIS-MACRO 01/17/2006
# prfernan@ncsu.edu
#
# IBIS-to-VerilogA converter script v0.1 (GUI capabilities added) -
1/30/06
# IBIS-to-VerilogA converter script v0.2 (VHDL-A(MS) parsing added) -
2/1/06
# IBIS-to-VerilogA converter script v0.3 (windows "pwd" bug fixed) -
2/2/06
# IBIS-to-VerilogA converter script v0.4 (more bug fixes) - 2/3/06
# IBIS-to-VerilogA converter script v0.5 (Updated 'convert' to handle ALL
models & corners) - 2/6/06
# IBIS-to-VerilogA converter script v0.6 (references/v_range, Rise/Fall
waveform ordering) - 2/8/06
#
# This program is used to convert ibis file to verilog/vhdl A(MS) files.
# This program requires Perl/Tk. http://www.perl.com/download.csp
# Note: only the "convert" menu works so far
#
#####
#####
use Tk 800.000;
use subs qw/file_menuitems convert_menuitems template_menuitems
lib_menuitems help_menuitems/;
require Tk::BrowseEntry;
require Tk::LabEntry;
use Tk::DialogBox;
use Tk::Dialog;
use Tk::ErrorDialog;

$Help_file="help.txt";
$default_IBIS_file="lab_1.ibs";

##### Create scroller_bar

my $mw = MainWindow->new(-title => "IBIS-to-AMS converter");
my $t = $mw->Scrolled("Text", -width => 50, -height => 20, -scrollbars =>
'se',
                    -wrap => 'none')->pack(-expand => 1, -fill => 'both');
    # scrollbars on south & east
$t->configure(-state => 'disabled');
    # disallows user typing in canvas
```

```

##### Create the menubar

$mw->configure(-menu => my $menubar = $mw->Menu);

map {$menubar->cascade( -label => '~' . $_->[0], -menuitems => $_->[1] )}
  ['File', file_menuitems],          # Each of these functions is
defined below, in the next section
  ['Convert', convert_menuitems],    # Each of these functions
is defined below, in the next section
  ['Template', template_menuitems],  # Each of these functions
is defined below, in the next section
  ['Macro Library', lib_menuitems],  # Each of these functions
is defined below, in the next section
  ['Help', help_menuitems];

if ($^O eq 'MSWin32') {
  $current_dir=`cd`; chomp($current_dir);
  $current_dir=$current_dir."\\";
  my $syst = $menubar->cascade(-label => '~System');
  my $dir = 'dir | sort | more';
  $syst->command(
    -label    => $dir,
    -command => sub {system $dir},
  );
} else{
  $current_dir=`pwd`; chomp($current_dir);
  $current_dir=$current_dir."/";
}
print "Running on $^O"."\n";
print "Current directory is $current_dir\n";

MainLoop;

#####
#####
#####
#####
#####

#####
#####
# File Menu
#####
#####
sub file_menuitems {

  # Create the menu items for the File menu.

  my($motif, $bisque) = (1, 0);
  my $new_image_format = 'png';

  [
    [qw/command ~Quit -accelerator Ctrl-q -command/ => sub {

```

```

        if($mw->Dialog(-title=>'Exit?', -text=>'Confirm exit', -
buttons=>['Exit', 'Return'], -default_button=>'Return', -
bitmap=>'warning')
        ->Show eq "Exit") { &exit }]],
    ];

} # end file_menuitems

#####
#####
# This function converts a model in a normal ibis file to AMS format
#####
#####
sub convert_menuitems {

    # Create the menu items for the File menu.

    my($motif, $bisque) = (1, 0);
    my $new_image_format = 'png';

    [
        ['command', '~Convert IBIS file to Verilog-A', qw/-accelerator
Ctrl-c/,
        -command => sub{ &ibs_to_AMS("verilog");}],
        ['command', '~Convert IBIS file to VHDL-A', qw/-accelerator
Ctrl-c/,
        -command => sub{ &ibs_to_AMS("vhdl");}],
    ];
} # end convert_menuitems

#####
#####
# opens an AMS template file and links its data
#####
#####
sub template_menuitems {
    # Create the menu items for the template menu

    my($motif, $bisque) = (1, 0);
    my $new_image_format = 'png';

    [
        ['command', 'Open ~template file', qw/-accelerator Ctrl-t/,
        -command => sub{

}],
    ];
} # end template_menuitems

#####
#####
# Read AMS macro library and list its contents
#####
#####
sub lib_menuitems {

```

```

    [
        ['command', '~List macro library modules', qw/-accelerator Ctrl-l/,
        -command => sub{
            if ($in_va_file=$mw->getOpenFile(-
            initialfile=>"$current_dir"."IBIS_macro_library.va")) {
                # $t->configure(-state => 'normal'); # disallows user typing
                # my $f1 = $t->Frame->pack(-side => 'top', -expand => 1, -fill
                =>'y');
                # Read modules in Verilog-A macro library file
                open( INFILEP, "< $in_va_file" ) or die "Can't open $in_va_file";
                $temp_modules="";
                while( $line=<INFILEP> ) {
                    if($line=~m/^(\\s*)module\\s+/i) {
                        @words = split(/\\s+/, $line);
                        push(@macro_lib_modules, $words[1]);
                        print "$words[1]\\n";
                        $temp_modules=$temp_modules."\\n$words[1]";
                    }
                }
                close INFILEP;
                # my $be = $f1->Label(-text => "$temp_modules", -relief => 'groove',
                -width => 50)
                # ->pack(-ipady => 5, -side => 'left');
            }
        }],
    ];
} # end lib_menuitems

#####
#####
# help
#####
#####
sub help_menuitems {
    [
        ['command', '~Help', qw/-accelerator Ctrl-h/, -command =>
        sub{&Editor_module($Help_file)}],'',
        ['command', '~About', qw/-accelerator Ctrl-a/, -command => sub {
            my $db2=$mw->Dialog(-title=> "About", -text=>"IBIS to AMS
            Converter\\nNCSU, IBIS-Macro 2006\\nRunning on
            $^O\\n\\nContact:\\nprfernan@ncsu.edu"."\\n",
            -buttons=>['Ok'], -default_button=> 'Ok', -bitmap=>'info')-
            >Show();}],
    ];
}

#####
#####
# ibs_to_AMS - Oversees the ibis to AMS conversion
#####
#####
sub ibs_to_AMS{

```

```

# Prompt for input ibis file
if ($in_ibs_file=$mw->getOpenFile(-
initialfile=>"$current_dir"."$default_IBIS_file")) {
    # Reinitialize data structs
    &Reset_data_structs; $in_model_name_temp=""; @choices="";

    # Read models(@choices) in ibis file
    open( INFILEP, "< $in_ibs_file" ) or die "Can't open $in_ibs_file";
    while( $line=<INFILEP> ) {
        if($line=~m/^(\\s*)\\[Model\\]\\s+/i) {
            @words = split(/\\s+/, $line);
            push(@choices, $words[1]);
        }
    }
    close INFILEP;
    push(@choices, "ALL");

    # Create dialog for model selection
    my $dbl=$mw->Dialog(-title=> "Select model", -text=>"Select model",
        -buttons=>['Ok', 'Cancel'], -default_button=> 'Ok');
    $dbl->BrowseEntry(-label => "Select model", -choices => \@choices,
        -variable => \\$in_model_name_temp, -justify=>'right')
        ->pack(-ipady => 2, -side => 'bottom', -fill => 'both');
    my $ans=$dbl->Show();

    if($ans eq "Ok" && $in_model_name_temp ne "" && $in_model_name_temp
ne "ALL") {
        $in_model_name=$in_model_name_temp;

        # Parse input file name
        $in_ibs_file1=$in_ibs_file; $in_ibs_file1=~s/.ibs//g;
        ($temp, $in_ibs_file1)=$in_ibs_file1=~m/(.*\\/)(.*)$/;

        # Dialog to get process corner, default: typical
        my $db2=$mw->Dialog(-title=> "Select Process corner", -
text=>"Select Process corner",
            -buttons=>['Typical', 'Minimum', 'Maximum', 'ALL'], -
default_button=> 'Typical');
        my $ans2=$db2->Show();
        if($ans2 eq 'Typical') { $corner="t"; $corner1="Typical";
&ibs_to_AMS_conversion($_[0]); &Reset_data_structs;}
        elsif($ans2 eq 'Minimum') { $corner="n"; $corner1="Minimum";
&ibs_to_AMS_conversion($_[0]); &Reset_data_structs;}
        elsif($ans2 eq 'Maximum') { $corner="x"; $corner1="Maximum";
&ibs_to_AMS_conversion($_[0]); &Reset_data_structs;}
        elsif($ans2 eq 'ALL') {
            $corner="t"; $corner1="Typical";
&ibs_to_AMS_conversion($_[0]); &Reset_data_structs;
            $corner="n"; $corner1="Minimum";
&ibs_to_AMS_conversion($_[0]); &Reset_data_structs;
            $corner="x"; $corner1="Maximum";
&ibs_to_AMS_conversion($_[0]); &Reset_data_structs;
        }
    }
}

```

```

        elsif($ans eq "Ok" && $in_model_name_temp eq "ALL") {
            foreach $in_model_name (@choices) {
                if($in_model_name ne "ALL" && $in_model_name ne "") {
                    # Parse input file name
                    $in_ibs_file1=$in_ibs_file;
$in_ibs_file1=~s/.ibs//g;
                    ($temp,
$in_ibs_file1)=$in_ibs_file1=~m/(.*\/)(.*)$/;

                    $corner="t"; $corner1="Typical";
&ibs_to_AMS_conversion($_[0], "FORCE"); &Reset_data_structs;
                    $corner="n"; $corner1="Minimum";
&ibs_to_AMS_conversion($_[0], "FORCE"); &Reset_data_structs;
                    $corner="x"; $corner1="Maximum";
&ibs_to_AMS_conversion($_[0], "FORCE"); &Reset_data_structs;
                }
            }
        }
    }
}

sub Reset_data_structs{
    $input_state=""; $ibs_model_type="";
    %ibs_Vpu_ref=""; %ibs_Vpd_ref=""; %ibs_Vpc_ref=""; %ibs_Vgc_ref="";
    %ibs_V_tf=""; %ibs_T_tf=""; %ibs_V_tr=""; %ibs_T_tr="";
    %ibs_vrange=(); %ibs_I_pc=""; %ibs_I_gc=""; %ibs_I_pu="";
%ibs_I_pd="";
    $ibs_Vinh=""; $ibs_Vinl=""; $ibs_V_pc=""; $ibs_V_gc="";
$ibs_V_pu=""; $ibs_V_pd="";
}

sub ibs_to_AMS_conversion{
    # Read ibis file and create data structs
    &create_datastructs($in_ibs_file);
    &sort_rising_falling_entries;

    # depending on selection create Verilog or VHDL output file.
    if($_[0] eq "verilog") {
        if($_[1] eq "FORCE") {

            &create_verilog_output_file("$current_dir"."$in_ibs_file1"."$_in_model_name"."$_corner"."dat");
        } elseif($out_file=$mw->getSaveFile(-
initialfile=>"$current_dir"."$in_ibs_file1"."$_in_model_name"."$_corner"."
.dat")) {
            &create_verilog_output_file($out_file);
        }
    }
    elseif($_[0] eq "vhdl") {
        if($_[1] eq "FORCE") {

            &create_vhdl_output_file("$current_dir"."$in_ibs_file1"."$_in_model_name"."$_corner"."txt");
        }
    }
}

```

```

        } elsif($out_file=$mw->getSaveFile(-
initialfile=>"$current_dir"."$in_ibs_file1"."_$in_model_name"."_$corner"."
.txt")) {
                &create_vhdl_output_file($out_file);
        }
    }
    #&Editor_module($out_file); # Open output file in text
viewer
    #&print_all_structs; # Prints all datastructures to screen
}

sub sort_rising_falling_entries{
    my $temp_Vfx_r; my $temp_Rfx_r; my $temp_TR_counter; my
    $temp_ibs_T_tr; my $temp_ibs_V_tr;

    # If V_fixture min & max don't exist copy V_fixture to them
    if($Vfx_r{"1n"} eq "") {$Vfx_r{"1n"}=$Vfx_r{"1t"};};
    if($Vfx_r{"2n"} eq "") {$Vfx_r{"2n"}=$Vfx_r{"2t"};};
    if($Vfx_f{"1n"} eq "") {$Vfx_f{"1n"}=$Vfx_f{"1t"};};
    if($Vfx_f{"2n"} eq "") {$Vfx_f{"2n"}=$Vfx_f{"2t"};};
    if($Vfx_r{"1x"} eq "") {$Vfx_r{"1x"}=$Vfx_r{"1t"};};
    if($Vfx_r{"2x"} eq "") {$Vfx_r{"2x"}=$Vfx_r{"2t"};};
    if($Vfx_f{"1x"} eq "") {$Vfx_f{"1x"}=$Vfx_f{"1t"};};
    if($Vfx_f{"2x"} eq "") {$Vfx_f{"2x"}=$Vfx_f{"2t"};};

    # R1 must have the lower Vfx (of R1 and R2)
    # F1 must have the higher Vfx (of F1 and F2)
    if($ibs_model_type1 ne "IBIS_OPENSINK" && $ibs_model_type1 ne
"IBIS_IO_OPENSINK" && $ibs_model_type1 ne "IBIS_OPENSOURCE" &&
$ibs_model_type1 ne "IBIS_IO_OPENSOURCE") {
        if($Vfx_r{"1".$corner}>$Vfx_r{"2".$corner}) { # swap the two
            #print "Swapped Rising WFs for model:$in_model_name
type:$ibs_model_type1 corner:$corner\n";
            $temp_Vfx_r = $Vfx_r{"1".$corner};
            $temp_Rfx_r = $Rfx_r{"1"};
            $temp_TR_counter=$TR_counter{"1"};
            $temp_ibs_T_tr = $ibs_T_tr{"1"};
            $temp_ibs_V_tr = $ibs_V_tr{"1"};

            $Vfx_r{"1".$corner} = $Vfx_r{"2".$corner};
            $Rfx_r{"1"} = $Rfx_r{"2"};
            $TR_counter{"1"} = $TR_counter{"2"};
            $ibs_T_tr{"1"} = $ibs_T_tr{"2"};
            $ibs_V_tr{"1"}=$ibs_V_tr{"2"};

            $Vfx_r{"2".$corner} = $temp_Vfx_r;
            $Rfx_r{"2"} = $temp_Rfx_r;
            $TR_counter{"2"} = $temp_TR_counter;
            $ibs_T_tr{"2"} = $temp_ibs_T_tr;
            $ibs_V_tr{"2"}=$temp_ibs_V_tr;
        }
        if($Vfx_f{"1".$corner}<$Vfx_f{"2".$corner}) { # swap the two
            #print "Swapped Falling WFs for model:$in_model_name
type:$ibs_model_type1 corner:$corner\n";

```

```

$Vfx_f{$corner}=$Vfx_f{"1"}.${corner};
$Rfx_f{$corner}=$Rfx_f{"1"};
$TF_counter{$corner}=$TF_counter{"1"};
$sibs_T_tf{$corner}=$sibs_T_tf{"1"};
$sibs_V_tf{$corner}=$sibs_V_tf{"1"};

$Vfx_f{"1"}.${corner}=$Vfx_f{"2"}.${corner};
$Rfx_f{"1"}=$Rfx_f{"2"};
$TF_counter{"1"}=$TF_counter{"2"};
$sibs_T_tf{"1"}=$sibs_T_tf{"2"};
$sibs_V_tf{"1"}=$sibs_V_tf{"2"};

$Vfx_f{"2"}.${corner}=$temp_Vfx_f;
$Rfx_f{"2"}=$temp_Rfx_f;
$TF_counter{"2"}=$temp_TF_counter;
$sibs_T_tf{"2"}=$temp_sibs_T_tf;
$sibs_V_tf{"2"}=$temp_sibs_V_tf;
}
}

# If [...] reference[s] are not provided use [voltage range] instead.
if($sibs_Vpu_ref{$corner} eq "") {
$sibs_Vpu_ref{$corner}=$sibs_vrange{$corner}; }
if($sibs_Vpd_ref{$corner} eq "") { $sibs_Vpd_ref{$corner}="0"; }
if($sibs_Vpc_ref{$corner} eq "") {
$sibs_Vpc_ref{$corner}=$sibs_vrange{$corner}; }
if($sibs_Vgc_ref{$corner} eq "") { $sibs_Vgc_ref{$corner}="0"; }
}
#####
#####
# functional subroutines:
#####
#####

#####
#####
# If valid args are provided, parse the model and create internal data
structs
#####
#####
sub create_datastructs{
$file_flag1="0"; %sibs_C_comp=""; $input_state="";
$PC_counter=0; $GC_counter=0; $PU_counter=0; $PD_counter=0;
$TR_counter{"1"}=0; $TR_counter{"2"}=0; $TF_counter{"1"}=0;
$TF_counter{"2"}=0; $current_TR=0; $current_TF=0;
if($in_model_name ne "") {
open( INFILEP, "<$_[0]" ) or die "Can't open $_[0]";
while( $line=<INFILEP> ) {
if($line=~m/^(.*)\[Model\]\s+$/i) {
$file_flag1="1";
#@words = split(/\s+/, $line);
#print "$words[1]\n";
}elsif($file_flag1 eq "1" && $line=~m/^(.*)\[Model\]\s+$/i) {
$file_flag1="0";
}
}
}
}

```

```

}

if($file_flag1 eq "1") {
    #print $line;
    if($line=~m/^(\\s*)\\[/i) {
        $input_state="";
    }

    if($line=~m/^(\\s*)\\|/i) {
        # Comment line
    }elseif($line=~m/^(\\s*)Model_type\\s+/i) {
        @words = split(/\\s+/, $line);
        $sibs_model_type=$words[1];
        if(lc($sibs_model_type) eq "input") {
$sibs_model_type1="IBIS_INPUT"
        }elseif(lc($sibs_model_type) eq "output") {
$sibs_model_type1="IBIS_OUTPUT"
        }elseif(lc($sibs_model_type) eq "i/o") {
$sibs_model_type1="IBIS_IO"
        }elseif(lc($sibs_model_type) eq "3-state") {
$sibs_model_type1="IBIS_3STATE"
        }elseif(lc($sibs_model_type) eq "open_sink") {
$sibs_model_type1="IBIS_OPENSINK"
        }elseif(lc($sibs_model_type) eq "i/o_open_sink") {
$sibs_model_type1="IBIS_IO_OPENSINK"
        }elseif(lc($sibs_model_type) eq "open_source") {
$sibs_model_type1="IBIS_OPENSOURCE"
        }elseif(lc($sibs_model_type) eq "i/o_open_source") {
$sibs_model_type1="IBIS_IO_OPENSOURCE"
        }else { print "Model type $sibs_model_type in model
$in_model_name in file $in_ibis_file is not valid\\n"; exit;}
    }elseif($line=~m/^(\\s*)Vin\\s*=/i) {
        $line=~s/(\\s+)/g; chomp($line);
        $line=&parse_val($line);
        @words = split(/=/, $line);
        $sibs_Vinl=$words[1];
    }elseif($line=~m/^(\\s*)Vinh\\s*=/i) {
        $line=~s/(\\s+)/g; chomp($line);
        $line=&parse_val($line);
        @words = split(/=/, $line);
        $sibs_Vinh=$words[1];
    }elseif($line=~m/^(\\s*)Vmeas\\s*=/i) {
        $line=~s/(\\s+)/g; chomp($line);
        $line=&parse_val($line);
        @words = split(/=/, $line);
        $sibs_Vmeas=$words[1];
    }elseif($line=~m/^(\\s*)Cref\\s*=/i) {
        $line=~s/(\\s+)/g; chomp($line);
        $line=&parse_val($line);
        @words = split(/=/, $line);
        $sibs_Cref=$words[1];
    }elseif($line=~m/^(\\s*)C_comp\\s*/i) {
        $line=&parse_val($line);
        @words = split(/\\s+/, $line);
        $sibs_C_comp{"t"}=$words[1];
    }
}

```

```

        $sibs_C_comp{"n"}=$words[2];
        $sibs_C_comp{"x"}=$words[3];
    }elseif($line=~m/^(\\s*)\\[Voltage Range\\]\\s*/i) {
        @words = split(/\\s+/, $line);
        $sibs_vrange{"t"}=$words[2];
        $sibs_vrange{"t"}=&parse_val($sibs_vrange{"t"});
        $sibs_vrange{"n"}=$words[3];
        $sibs_vrange{"n"}=&parse_val($sibs_vrange{"n"});
        $sibs_vrange{"x"}=$words[4];
        $sibs_vrange{"x"}=&parse_val($sibs_vrange{"x"});
    }elseif($line=~m/^(\\s*)\\[Pullup Reference\\]\\s*/i) {
        @words = split(/\\s+/, $line);
        $sibs_Vpu_ref{"t"}=$words[2];
        $sibs_Vpu_ref{"t"}=&parse_val($sibs_Vpu_ref{"t"});
        $sibs_Vpu_ref{"n"}=$words[3];
        $sibs_Vpu_ref{"n"}=&parse_val($sibs_Vpu_ref{"n"});
        $sibs_Vpu_ref{"x"}=$words[4];
        $sibs_Vpu_ref{"x"}=&parse_val($sibs_Vpu_ref{"x"});
    }elseif($line=~m/^(\\s*)\\[Pulldown Reference\\]\\s*/i) {
        @words = split(/\\s+/, $line);
        $sibs_Vpd_ref{"t"}=$words[2];
        $sibs_Vpd_ref{"t"}=&parse_val($sibs_Vpd_ref{"t"});
        $sibs_Vpd_ref{"n"}=$words[3];
        $sibs_Vpd_ref{"n"}=&parse_val($sibs_Vpd_ref{"n"});
        $sibs_Vpd_ref{"x"}=$words[4];
        $sibs_Vpd_ref{"x"}=&parse_val($sibs_Vpd_ref{"x"});
    }elseif($line=~m/^(\\s*)\\[POWER Clamp Reference\\]\\s*/i) {
        @words = split(/\\s+/, $line);
        $sibs_Vpc_ref{"t"}=$words[3];
        $sibs_Vpc_ref{"t"}=&parse_val($sibs_Vpc_ref{"t"});
        $sibs_Vpc_ref{"n"}=$words[4];
        $sibs_Vpc_ref{"n"}=&parse_val($sibs_Vpc_ref{"n"});
        $sibs_Vpc_ref{"x"}=$words[5];
        $sibs_Vpc_ref{"x"}=&parse_val($sibs_Vpc_ref{"x"});
    }elseif($line=~m/^(\\s*)\\[GND Clamp Reference\\]\\s*/i) {
        @words = split(/\\s+/, $line);
        $sibs_Vgc_ref{"t"}=$words[3];
        $sibs_Vgc_ref{"t"}=&parse_val($sibs_Vgc_ref{"t"});
        $sibs_Vgc_ref{"n"}=$words[4];
        $sibs_Vgc_ref{"n"}=&parse_val($sibs_Vgc_ref{"n"});
        $sibs_Vgc_ref{"x"}=$words[5];
        $sibs_Vgc_ref{"x"}=&parse_val($sibs_Vgc_ref{"x"});
    }elseif($line=~m/^(\\s*)\\[Temperature Range\\]\\s*/i) {
        @words = split(/\\s+/, $line);
        $sibs_trange{"t"}=$words[2];
        $sibs_trange{"t"}=&parse_val($sibs_trange{"t"});
        $sibs_trange{"n"}=$words[3];
        $sibs_trange{"n"}=&parse_val($sibs_trange{"n"});
        $sibs_trange{"x"}=$words[4];
        $sibs_trange{"x"}=&parse_val($sibs_trange{"x"});
    }

}elseif($line=~m/^(\\s*)\\[POWER Clamp\\]/i) {
    $input_state="PC";
}elseif($line=~m/^(\\s*)\\[GND Clamp\\]/i) {
    $input_state="GC";
}

```

```

}elsif($line=~m/^(\\s*)\\[Pullup\\]/i) {
    $input_state="PU";
}elsif($line=~m/^(\\s*)\\[Pulldown\\]/i) {
    $input_state="PD";
}elsif($line=~m/^(\\s*)\\[Falling Waveform\\]/i) {
    $input_state="TF"; $current_TF++;
}elsif($line=~m/^(\\s*)\\[Rising Waveform\\]/i) {
    $input_state="TR"; $current_TR++;
}elsif($input_state ne "") {
    if(($input_state eq "TR" || $input_state eq "TF")
&& $line=~m/_fixture/i) {
        if($line=~m/^V_fixture(\\s*)=/i) {
            $line=~s/(\\s+)/g; chomp($line);
            @words = split(/=/, $line);
            $words[1]=~s/v//ig;
            if($input_state eq "TR") {
                $Vfx_r{$current_TR."t"}=&parse_val($words[1]);
            }
            if($input_state eq "TF") {
                $Vfx_f{$current_TF."t"}=&parse_val($words[1]);
            }
        }elsif($line=~m/^V_fixture_min(\\s*)=/i) {
            $line=~s/(\\s+)/g; chomp($line);
            @words = split(/=/, $line);
            $words[1]=~s/v//ig;
            if($input_state eq "TR") {
                $Vfx_r{$current_TR."n"}=&parse_val($words[1]);
            }
            if($input_state eq "TF") {
                $Vfx_f{$current_TF."n"}=&parse_val($words[1]);
            }
        }elsif($line=~m/^V_fixture_max(\\s*)=/i) {
            $line=~s/(\\s+)/g; chomp($line);
            @words = split(/=/, $line);
            $words[1]=~s/v//ig;
            if($input_state eq "TR") {
                $Vfx_r{$current_TR."x"}=&parse_val($words[1]);
            }
            if($input_state eq "TF") {
                $Vfx_f{$current_TF."x"}=&parse_val($words[1]);
            }
        }elsif($line=~m/^R_fixture/i) {
            $line=~s/(\\s+)/g; chomp($line);
            @words = split(/=/, $line);
            $words[1]=~s/ohm//ig;
            if($input_state eq "TR") {
                $Rfx_r{$current_TR}=&parse_val($words[1]);
            }
            if($input_state eq "TF") {
                $Rfx_f{$current_TF}=&parse_val($words[1]);
            }
        }
    }
}
else {
    # Values section
    chomp($line);
    $line=~s/^(\\s+)/g;
    $line=&parse_val($line);
    @words = split(/\\s+/, $line);
    if($input_state eq "PC") {
        $PC_counter++;
        if($sibs_V_pc ne "") {
            $sibs_V_pc=$sibs_V_pc.", ";
        }
    }
}

```





```

    }
  }
}
}

#####
#####
# Create output .dat file in verilog-A(MS) format
#####
#####
sub create_verilog_output_file{
open( OUTFILEP, "> $_[0]" ) or die "Can't open $_[0]";
  print OUTFILEP "// Generated on: ".$Return_time."\n// Input file:
$in_ibs_file\n";
  print OUTFILEP "// Verilog-A primitive type: $sibs_model_type1
corner=$corner1\n\n";
  print OUTFILEP "`define $in_ibs_file1"."_$in_model_name"."
\\\".\n\\\".\n";
  if($sibs_C_comp{$corner} ne "") {
    print OUTFILEP ".C_comp($sibs_C_comp{$corner}), \\\".\n";
  }
  if($sibs_model_type1 ne "IBIS_OUTPUT" &&$sibs_model_type1 ne
"IBIS_OPENSINK" && $sibs_model_type1 ne "IBIS_OPENSOURCE") {
    if($sibs_Vinh ne "") {
      print OUTFILEP ".Vinh($sibs_Vinh), \\\".\n";
    }
    if($sibs_Vinl ne "") {
      print OUTFILEP ".Vinl($sibs_Vinl), \\\".\n";
    }
  }
  if($sibs_model_type1 ne "IBIS_INPUT") {
    print OUTFILEP ".Vpc_ref($sibs_Vpc_ref{$corner}), \\\".\n";
    print OUTFILEP ".Vgc_ref($sibs_Vgc_ref{$corner}), \\\".\n";
    if($sibs_model_type1 ne "IBIS_OPENSINK" && $sibs_model_type1 ne
"IBIS_IO_OPENSINK") {
      print OUTFILEP ".Vpu_ref($sibs_Vpu_ref{$corner}),
\\\".\n";
      print OUTFILEP ".Vpd_ref($sibs_Vpd_ref{$corner}),
\\\".\n\\\".\n";
    }
  }

  print OUTFILEP ".IVpc_length($PC_counter), \\\".\n";
  print OUTFILEP ".Ipc_data({$sibs_I_pc{$corner}}), \\\".\n";
  print OUTFILEP ".Vpc_data({$sibs_V_pc}), \\\".\n";

  print OUTFILEP ".IVgc_length($GC_counter), \\\".\n";
  print OUTFILEP ".Igc_data({$sibs_I_gc{$corner}}), \\\".\n";
  print OUTFILEP ".Vgc_data({$sibs_V_gc}), \\\".\n\\\".\n";

  if($sibs_model_type1 ne "IBIS_INPUT") {
    if($sibs_model_type1 ne "IBIS_OPENSINK" && $sibs_model_type1 ne
"IBIS_IO_OPENSINK" ) {

```

```

        print OUTFILEP ".IVpu_length($PU_counter), \\\".\n";
        print OUTFILEP ".Ipu_data({$sibs_I_pu{$corner}}),
\\\".\n";
        print OUTFILEP ".Vpu_data({$sibs_V_pu}), \\\".\n";
    }

    if($sibs_model_type1 ne "IBIS_OPENSOURCE" && $sibs_model_type1
ne "IBIS_IO_OPENSOURCE" ) {
        print OUTFILEP ".IVpd_length($PD_counter), \\\".\n";
        print OUTFILEP ".Ipd_data({$sibs_I_pd{$corner}}),
\\\".\n";
        print OUTFILEP ".Vpd_data({$sibs_V_pd}), \\\".\n\\\".\n";
    }
}

if($sibs_model_type1 ne "IBIS_INPUT") {
    print OUTFILEP ".Vfx_r1($Vfx_r{"1\".$corner}), \\\".\n";
    print OUTFILEP ".Rfx_r1($Rfx_r{"1\"}), \\\".\n";
    print OUTFILEP ".VTr1_length($TR_counter{"1\"}), \\\".\n";
    print OUTFILEP ".Tr1_data({$sibs_T_tr{"1\"}}), \\\".\n";
    print OUTFILEP ".Vr1_data({$sibs_V_tr{$corner.\"1\"}}),
\\\".\n\\\".\n";

    print OUTFILEP ".Vfx_f1($Vfx_f{"1\".$corner}), \\\".\n";
    print OUTFILEP ".Rfx_f1($Rfx_f{"1\"}), \\\".\n";
    print OUTFILEP ".VTf1_length($TF_counter{"1\"}), \\\".\n";
    print OUTFILEP ".Tf1_data({$sibs_T_tf{"1\"}}), \\\".\n";
    print OUTFILEP ".Vf1_data({$sibs_V_tf{$corner.\"1\"}}),
\\\".\n\\\".\n";

    if($sibs_model_type1 ne "IBIS_OPENSINK" && $sibs_model_type1 ne
"IBIS_IO_OPENSINK" && $sibs_model_type1 ne "IBIS_OPENSOURCE" &&
$sibs_model_type1 ne "IBIS_IO_OPENSOURCE") {
        print OUTFILEP ".Vfx_r2($Vfx_r{"2\".$corner}),
\\\".\n";

        print OUTFILEP ".Rfx_r2($Rfx_r{"2\"}), \\\".\n";
        print OUTFILEP ".VTr2_length($TR_counter{"2\"}),
\\\".\n";

        print OUTFILEP ".Tr2_data({$sibs_T_tr{"2\"}}), \\\".\n";
        print OUTFILEP ".Vr2_data({$sibs_V_tr{$corner.\"2\"}}),
\\\".\n\\\".\n";

        print OUTFILEP ".Vfx_f2($Vfx_f{"2\".$corner}),
\\\".\n";

        print OUTFILEP ".Rfx_f2($Rfx_f{"2\"}), \\\".\n";
        print OUTFILEP ".VTf2_length($TF_counter{"2\"}),
\\\".\n";

        print OUTFILEP ".Tf2_data({$sibs_T_tf{"2\"}}), \\\".\n";
        print OUTFILEP
".Vf2_data({$sibs_V_tf{$corner.\"2\"}}).\".\n";
    }
}

close OUTFILEP;
}

```

```

#####
#####
# Create output .dat file in VHDL-A(MS) format
#####
#####
sub create_vhdl_output_file{
open( OUTFILEP, "> $_[0]" ) or die "Can't open $_[0]";
  print OUTFILEP "-- Generated on: ".&Return_time."\n-- Input file:
$in_ibs_file\n";
  print OUTFILEP "-- VHDL-A primitive type: $ibs_model_typed
corner=$corner1\n\n";

  if($ibs_C_comp{$corner} ne "") {
    print OUTFILEP "-----". "\n";
    print OUTFILEP "-- C_comp Parameters --". "\n";
    print OUTFILEP "-----". "\n";
    print OUTFILEP "C_comp". "\n$ibs_C_comp{$corner} ". "\n";
    print OUTFILEP "kC_comp_pc\n0.25\n";
    print OUTFILEP "kC_comp_pu\n0.25\n";
    print OUTFILEP "kC_comp_pd\n0.25\n";
    print OUTFILEP "kC_comp_gc\n0.25\n";
  }
  if($ibs_model_typed ne "IBIS_OUTPUT" &&$ibs_model_typed ne
"IBIS_OPENSINK" && $ibs_model_typed ne "IBIS_OPENSOURCE") {
    if($ibs_Vinh ne "") {
      print OUTFILEP "-----". "\n";
      print OUTFILEP "-- Receiver Thresholds --". "\n";
      print OUTFILEP "-----". "\n";
      print OUTFILEP "Vinh\n$ibs_Vinh". "\n";
    }
    if($ibs_Vinl ne "") {
      print OUTFILEP "Vinl\n$ibs_Vinl". "\n";
    }
  }
  if($ibs_model_typed ne "IBIS_INPUT") {
    print OUTFILEP "-----
---". "\n";
    print OUTFILEP "-- [Pullup Reference] and [Pulldown Reference]
--". "\n";
    print OUTFILEP "-----
---". "\n";
    print OUTFILEP "Vpc_ref\n$ibs_Vpc_ref{$corner} ". "\n";
    print OUTFILEP "Vgc_ref\n$ibs_Vgc_ref{$corner} ". "\n";
    if($ibs_model_typed ne "IBIS_OPENSINK" && $ibs_model_typed ne
"IBIS_IO_OPENSINK") {
      print OUTFILEP "Vpu_ref\n$ibs_Vpu_ref{$corner} ". "\n";
      print OUTFILEP "Vpd_ref\n$ibs_Vpd_ref{$corner} ". "\n";
    }
  }
}

print OUTFILEP "\n";

#print OUTFILEP ".IVpc_length($PC_counter), \\". "\n";
$ibs_I_pc{$corner}=~/s/, /\n/g;

```

```

print OUTFILEP "-----". "\n";
print OUTFILEP "-- IV curve tables --". "\n";
print OUTFILEP "-----". "\n";
print OUTFILEP "Ipc_data\n$sibs_I_pc{$corner}." "\n\n";
$sibs_V_pc=~s/, /\n/g;
print OUTFILEP "Vpc_data\n$sibs_V_pc." "\n\n";

#print OUTFILEP ".IVgc_length($GC_counter), \\" "\n";
$sibs_I_gc{$corner}=~s/, /\n/g;
print OUTFILEP "Igc_data\n$sibs_I_gc{$corner}." "\n\n";
$sibs_V_gc=~s/, /\n/g;
print OUTFILEP "Vgc_data\n$sibs_V_gc." "\n\n";

if($sibs_model_type1 ne "IBIS_INPUT") {
    if($sibs_model_type1 ne "IBIS_OPENSINK" && $sibs_model_type1 ne
"IBIS_IO_OPENSINK" ) {
        #print OUTFILEP "IVpu_length($PU_counter), \\" "\n";
        $sibs_I_pu{$corner}=~s/, /\n/g;
        print OUTFILEP "\nIpu_data\n$sibs_I_pu{$corner}." "\n";
        $sibs_V_pu=~s/, /\n/g;
        print OUTFILEP "\nVpu_data\n$sibs_V_pu." "\n";
    }

    if($sibs_model_type1 ne "IBIS_OPENSOURCE" && $sibs_model_type1
ne "IBIS_IO_OPENSOURCE" ) {
        #print OUTFILEP ".IVpd_length($PD_counter), \\" "\n";
        $sibs_I_pd{$corner}=~s/, /\n/g;
        print OUTFILEP "\nIpd_data\n$sibs_I_pd{$corner}." "\n";
        $sibs_V_pd=~s/, /\n/g;
        print OUTFILEP "\nVpd_data\n$sibs_V_pd." "\n";
    }
}
print OUTFILEP "\n";

if($sibs_model_type1 ne "IBIS_INPUT") {
    #print OUTFILEP "`define
$in_ibs_file1"_"$in_model_name"_"_VT_data \\" "\n";
    print OUTFILEP "-----". "\n";
    print OUTFILEP "-- Vt curve tables --". "\n";
    print OUTFILEP "-----". "\n";
    print OUTFILEP "\nVfx_r1\n$Vfx_r{\\"1\"}.$corner}." "\n";
    print OUTFILEP "\nRfx_r1\n$Rfx_r{\\"1\"}." "\n";
    #print OUTFILEP "\nVTr1_length\n$TR_counter{\\"1\"}." "\n";
    $sibs_T_tr{"1"}=~s/, /\n/g;
    print OUTFILEP "\nTr1_data\n$sibs_T_tr{\\"1\"}." "\n";
    $sibs_V_tr{$corner."1"}=~s/, /\n/g;
    print OUTFILEP "\nVr1_data\n$sibs_V_tr{$corner.\\"1\"}." "\n";

    print OUTFILEP "\nVfx_f1\n$Vfx_f{\\"1\"}.$corner}." "\n";
    print OUTFILEP "\nRfx_f1\n$Rfx_f{\\"1\"}." "\n";
    #print OUTFILEP "\nVTf1_length\n$TF_counter{\\"1\"}." "\n";
    $sibs_T_tf{"1"}=~s/, /\n/g;
    print OUTFILEP "\nTf1_data\n$sibs_T_tf{\\"1\"}." "\n";
    $sibs_V_tf{$corner."1"}=~s/, /\n/g;
    print OUTFILEP "\nVf1_data\n$sibs_V_tf{$corner.\\"1\"}." "\n";
}

```

```

        if($sibs_model_typed1 ne "IBIS_OPENSINK" && $sibs_model_typed1 ne
"IBIS_IO_OPENSINK" && $sibs_model_typed1 ne "IBIS_OPENSOURCE" &&
$sibs_model_typed1 ne "IBIS_IO_OPENSOURCE") {
            print OUTFILEP "\nVfx_r2\n$Vfx_r{"2\".$corner}".$n";
            print OUTFILEP "\nRfx_r2\n$Rfx_r{"2\"}".$n";
            #print OUTFILEP
"\nVTr2_length\n$TR_counter{"2\"}".$n";
            $sibs_T_tr{"2"}=~s/, /\n/g;
            print OUTFILEP "\nTr2_data\n$sibs_T_tr{"2\"}".$n";
            $sibs_V_tr{$corner."2"}=~s/, /\n/g;
            print OUTFILEP
"\nVr2_data\n$sibs_V_tr{$corner.\"2\"}".$n";

            print OUTFILEP "\nVfx_f2\n$Vfx_f{"2\".$corner}".$n";
            print OUTFILEP "\nRfx_f2\n$Rfx_f{"2\"}".$n";
            #print OUTFILEP
"\nVTf2_length\n$TF_counter{"2\"}".$n";
            $sibs_T_tf{"2"}=~s/, /\n/g;
            print OUTFILEP "\nTf2_data\n$sibs_T_tf{"2\"}".$n";
            $sibs_V_tf{$corner."2"}=~s/, /\n/g;
            print OUTFILEP
"\nVf2_data\n$sibs_V_tf{$corner.\"2\"}".$n";
        }
    }

close OUTFILEP;
}

#####
#####
# Formats numerical values to exponential notation
#####
#####
sub parse_val{
    my $line; $line=$_[0];
    $line=~s/V//g;
    $line=~s/A//g;
    $line=~s/F//g;
    $line=~s/ohm(\s*)$/g;
    $line=~s/f/E-15/g;
    $line=~s/p/E-12/g;
    $line=~s/n/E-9/g;
    $line=~s/u/E-6/g;
    $line=~s/m/E-3/g;
    $line=~s/k/E+3/g;
    $line=~s/M/E+6/g;
    $line=~s/G/E+9/g;
    $line=~s/T/E+12/g;
    return $line;
}

#####
#####
# Prints all datastructures

```

```

#####
#####
sub print_all_structs{
    print "// Generated on: ".&Return_time."\n\n";
    print "ibs_model_type=$ibs_model_type\n";
    print "ibs_Vinl=$ibs_Vinl\n";
    print "ibs_Vinh=$ibs_Vinh\n";
    print "ibs_Vmeas=$ibs_Vmeas\n";
    print "ibs_Cref=$ibs_Cref\n";
    foreach $line (%ibs_C_comp) { # prints ibs_C_comp t,n,x
        if($ibs_C_comp{$line} ne "") {
            print "ibs_C_comp_$line = $ibs_C_comp{$line}\t";
        }
    }
    print "\n";
    foreach $line (%ibs_vrange) { # prints ibs_vrange t,n,x
        if($ibs_vrange{$line} ne "") {
            print "ibs_vrange_$line = $ibs_vrange{$line}\t";
        }
    }
    print "\n";
    foreach $line (%ibs_trange) { # prints ibs_trange t,n,x
        if($ibs_trange{$line} ne "") {
            print "ibs_trange_$line = $ibs_trange{$line}\t";
        }
    }
    print "\n";
    print "\n";

#####

    print "ibs_V_pc=$ibs_V_pc\n";          # prints PC voltages
    print "PC_counter=$PC_counter\n";      # prints PC number of datapoints
    foreach $line (%ibs_I_pc) {            # prints PC currents t,n,x
        if($ibs_I_pc{$line} ne "") {
            print "ibs_I_pc_$line = $ibs_I_pc{$line}";
        }
    }
    print "\n";
}
print "\n";
print "\n";

    print "ibs_V_gc=$ibs_V_gc\n";          # GC
    print "GC_counter=$GC_counter\n";
    foreach $line (%ibs_I_gc) {
        if($ibs_I_gc{$line} ne "") {
            print "ibs_I_gc_$line = $ibs_I_gc{$line}";
        }
    }
    print "\n";
}
print "\n";
print "\n";

    print "ibs_V_pu=$ibs_V_pu\n";          # PU
    print "PU_counter=$PU_counter\n";

```

```

foreach $line (%ibs_I_pu) {
    if($ibs_I_pu{$line} ne "") {
        print "ibs_I_pu_$line = $ibs_I_pu{$line}";
    }
    print "\n";
}
print "\n";
print "\n";

print "ibs_V_pd=$ibs_V_pd\n";          # PD
print "PD_counter=$PD_counter\n";
foreach $line (%ibs_I_pd) {
    if($ibs_I_pd{$line} ne "") {
        print "ibs_I_pd_$line = $ibs_I_pd{$line}";
    }
    print "\n";
}
print "\n";
print "\n";

    print "ibs_T_tr{\\"1\"}=$ibs_T_tr{\\"1\"}\n";      # prints rising
waveforms(1) time range
    print "PD_counter=$TR_counter{\\"1\"}\n"; # prints rising
waveforms(1) number of datapoints
    foreach $line (%ibs_V_tr) {          # prints rising
waveforms(1) voltages t,n,x
        if($ibs_V_tr{$line} ne "") {
            print "ibs_V_tr_$line = $ibs_V_tr{$line}";
        }
        print "\n";
    }
    print "\n";
    print "\n";
}

#####
#####
#    Return_time      - formats the time for time stamping the output
files
#####
#####
sub Return_time {
    my $sec; my $min; my $hour; my $mday; my $mon; my $year; my $yday;
my $yday;
    ($sec, $min, $hour, $mday, $mon, $year, $yday, $yday) =
localtime(time);
    $year=$year+1900; $mon=$mon+1;
    return "Time: $hour:$min:$sec\tDate: $mon/$mday, $year";
}

#####
#####
#    Editor_module      - Opens a file for viewing purposes
#####
#####

```

```

sub Editor_module {
    my $line1; my $file_to_open;

    if($_[0]) { $file_to_open=$_[0];}
    else {$file_to_open=$mw->getOpenFile(-initialfile=>"$current_dir")}

    if($file_to_open) {
        my $text_mw = MainWindow->new(-title => "File Viewer. NCSU
2006");
        my $t = $text_mw->Scrolled('Text', -scrollbars=>'se')->pack(-
fill => 'both', -expand => 1);

        $t->insert('end', "");
        $t->markSet('one', '1.3');
        if(open( INIBSFILEP, "< $file_to_open" )) {
            while( $line1=<INIBSFILEP> ) {
                $t->insert('end', $line1);
            }
        }
        close INIBSFILEP;
    }
}

# EOF

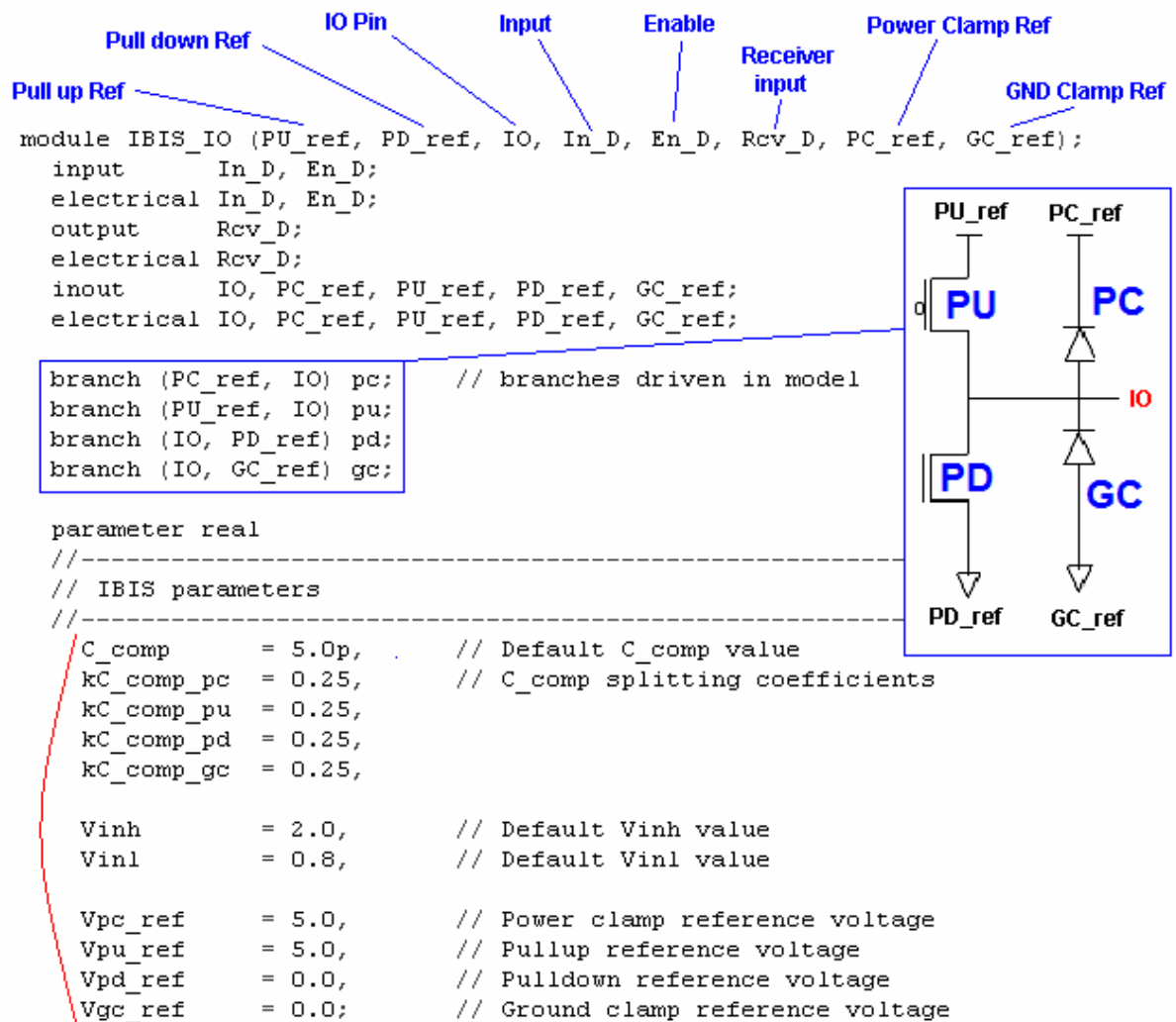
```

# Appendix C

## The Verilog-AMS “IBIS\_IO” model description

This appendix contains the complete Verilog-AMS description of the IBIS\_IO (3-state output and input mode) buffer model. The other models are quite similar. The complete AMS library can be obtained from the IBIS-macro website. The AMS library was developed by the IBIS-macro group & coded by **Arpad Muranyi (Intel)**.

This model (and all the other buffer models in the AMS library) follows the algorithm described in Section 1.4.



**\*All parameters can be set as required when instantiated in the template**

The parameters are set to desired values by the ibis2ams converter’s output files.

Note that all the tables default to 4 points. According to the IBIS specification up to 100 points can be added. These are the tables (parameters) that will be initialized by the ibis2ams converter scripts output files.

Vr1 must have the lower V\_fixture value of Vr1 & Vr2 (as shown below)  
 Vf1 must have the higher V\_fixture value of Vf1 & Vf2 (as shown below)  
 This ordering is done automatically by the ibis2ams script.

```

//-----
// Vectors of the IV curve tables
//-----
parameter integer
  IVpc_length = 4,
  IVpu_length = 4,
  IVpd_length = 4,
  IVgc_length = 4;

parameter real
  Ipc_data[1:IVpc_length] = { 0.08, 0.00, 0.00, 0.00}, Power Clamp I-V Curve
  Vpc_data[1:IVpc_length] = {-5.00, -1.00, 5.00, 10.00},
  Ipu_data[1:IVpu_length] = { 0.10, 0.00, -0.10, -0.20}, Pullup I-V Curve
  Vpu_data[1:IVpu_length] = {-5.00, 0.00, 5.00, 10.00},
  Ipd_data[1:IVpd_length] = {-0.10, 0.00, 0.10, 0.20}, GND Clamp I-V Curve
  Vpd_data[1:IVpd_length] = {-5.00, 0.00, 5.00, 10.00},
  Igc_data[1:IVgc_length] = {-0.08, 0.00, 0.00, 0.00}, Pulldown I-V Curve
  Vgc_data[1:IVgc_length] = {-5.00, -1.00, 5.00, 10.00};
//-----
// Vectors of the Vt curve tables
//-----
parameter integer
  VTr1_length = 4,
  VTr2_length = 4,
  VTf1_length = 4,
  VTf2_length = 4;

parameter real
  Vr1_data[1:VTr1_length] = {0.00, 0.00, 2.50, 2.50},
  Tr1_data[1:VTr1_length] = {0.00, 1.00e-9, 2.00e-9, 3.00e-9},
  Vr2_data[1:VTr2_length] = {2.50, 2.50, 5.00, 5.00},
  Tr2_data[1:VTr2_length] = {0.00, 0.50e-9, 0.80e-9, 3.00e-9},
  Vf1_data[1:VTf1_length] = {5.00, 5.00, 2.50, 2.50},
  Tf1_data[1:VTf1_length] = {0.00, 1.00e-9, 2.00e-9, 3.00e-9},
  Vf2_data[1:VTf2_length] = {2.50, 2.50, 0.00, 0.00},
  Tf2_data[1:VTf2_length] = {0.00, 0.50e-9, 0.80e-9, 3.00e-9};
//-----
parameter real
  Vfx_r1 = 0.0, rising1 has the lower Vfx // V_fixture values
  Vfx_r2 = 5.0, rising2 has the higher Vfx
  Vfx_f1 = 5.0, falling1 has the higher Vfx
  Vfx_f2 = 0.0, falling2 has the lower Vfx

  Rfx_r1 = 50.0 from (0:inf), // R_fixture values
  Rfx_r2 = 50.0 from (0:inf),
  Rfx_f1 = 50.0 from (0:inf),
  Rfx_f2 = 50.0 from (0:inf),
*All parameters can be set as required when instantiated in the template

```

```

//=====
// Non-IBIS parameters
//-----
Max_dt = 1.0e-12 from (0:inf), // Maximum dt between time points
Vth_R = 0.8, // Input threshold for rising edges
Vth_F = 0.2; // Input threshold for falling edges
//-----
// Internal variables
//-----
// Calculate maximum array length for common time and K*** tables
// and declare the arrays
//-----
parameter real Max_t = max(max(max(Tr1_data[VTr1_length], Tr2_data[VTr2_length]), Tf1_data
parameter real Min_t = min(min(min(Tr1_data[1], Tr2_data[1]), Tf1_data[1]), Tf2_data[1]);
parameter integer Max_length = ceil((Max_t-Min_t)/Max_dt) + VTr1_length + VTr2_length + VTf1_lengt

real Tcm[1:Max_length]; // Common time axis array
real Kr1[1:Max_length]; // Scaling coefficient arrays
real Kr2[1:Max_length];
real Kf1[1:Max_length];
real Kf2[1:Max_length];

//-----
// Logic state variables
integer pu_on;
integer pu_off;
integer pd_on;
integer pd_off;

//-----
// Variables to store event time
real Tpu_on_event;
real Tpu_off_event;
real Tpd_on_event;
real Tpd_off_event;

//-----
real kpu; // Output current scaling coefficients
real kpd;

```

The K\* tables are created using the Vt tables to scale the PU & PD I-V curves with respect to time to account for the partially on/off transient characteristics.

The K\* tables are created during the initial\_step. The tables are never modified thereafter.

```

//-----
// variables used in Common_time / Common_wfm / Kcoef "function"
//-----
integer i;
integer Common_length;
integer Index_r1;
integer Index_r2;
integer Index_f1;
integer Index_f2;
integer New_index_r1;
integer New_index_r2;
integer New_index_f1;
integer New_index_f2;
real Old_t;
real New_t;
real Last_t;
real Additional_pts;
real New_dt;

real Vr1_common[1:3]; // Need 3 elements in order to do derivatives
real Vr2_common[1:3];
real Vf1_common[1:3];
real Vf2_common[1:3];

real dVwfm_r1;
real dVwfm_r2;
real dVwfm_f1;
real dVwfm_f2;

real Iout1;
real Iout2;
real Ipd1;
real Ipd2;
real Ipu1;
real Ipu2;
real Igc1;
real Igc2;
real Ipc1;
real Ipc2;

real Term1;
real Term2;
real Den;

```

Parameter declarations end at this point. The rest of the module provides the behavioral description.

The behavioral description begins here (at keyword “analog”)

```
//=====
analog begin

@(initial_step) begin This block runs once at startup to initialize tables and variables
//=====
// This "function" makes a common time axis for all Vt curves and
// calculates the corresponding voltage points with linear interpolation
// and/or linear extrapolation if necessary. After that it converts the
// Vt curves to K coefficients which will be used in the analog equations
// to scale the IV curves. The K coefficients are stored in arrays.
//-----
// Initialize variables
//-----
Common_length = 1;
Index_r1      = 1;
Index_r2      = 1;
Index_f1      = 1;
Index_f2      = 1;
New_index_r1  = VTr1_length;
New_index_r2  = VTr2_length;
New_index_f1  = VTf1_length;
New_index_f2  = VTf2_length;
//-----
// Put the earliest time value of all given time vectors into "Old_t"
//-----
Old_t = min(Tr1_data[1], Tr2_data[1]); Typically all T*_data[1]=0
Old_t = min(Old_t, Tf1_data[1]); i.e. all Vt tables start at 0ns
Old_t = min(Old_t, Tf2_data[1]);
//-----
// Put the latest time value of all given time vectors into "New_t"
//-----
Last_t = max(Tr1_data[VTr1_length], Tr2_data[VTr2_length]);
Last_t = max(New_t, Tf1_data[VTf1_length]);
Last_t = max(New_t, Tf2_data[VTf2_length]);
New_t = Last_t;

Typically all T*_data[VT*_length] are equal to each other and represent the transient sim run-time
```

After the above code, Old\_t is set to the least time point in the Vt tables, usually 0ns. Likewise, New\_t is set to their highest value, for example 3ns (the default for all T\*\_data).

The while block shown in the next page uses the time axis of the Vt tables to initialize the Tcm array which will hold the common time. This array will extend contain times from Old\_t to New\_t spaced by Max\_dt (1ps). Hence, for the example above:

Tcm[1]=0ps, Tcm[2]=1ps, ..... , Tcm[3001]=3000ps

```

//-----
// Loop until latest time value is reached in each given time vector
//-----
while (Old_t < New_t) begin ----- The while blocks sole purpose is to
//----- initialize Tcm[] such that:
// Save the time value from Old_t in Tcm Tcm[1]=min (i.e. original Old_t)
// and increment Common_length counter Tcm[2]=min+Max_dt
//----- Tcm[3]=min+2Max_dt
Tcm[Common_length] = Old_t; .....
Common_length = Common_length + 1; Till Tcm[n]<=max (i.e. original New_t)
//-----
// Find which given time vector(s) have the lowest time value and
// advance their temporary indexes
//-----
if (Tr1_data[Index_r1] <= Old_t) New_index_r1 = Index_r1 + 1;
if (Tr2_data[Index_r2] <= Old_t) New_index_r2 = Index_r2 + 1;
if (Tf1_data[Index_f1] <= Old_t) New_index_f1 = Index_f1 + 1;
if (Tf2_data[Index_f2] <= Old_t) New_index_f2 = Index_f2 + 1;
//-----
// Find the lowest value at the new indexes in the given vector(s)
// and update indexes for next iteration
//-----
if (New_index_r1 <= VTr1_length) begin
    if (Tr1_data[New_index_r1] < New_t) New_t = Tr1_data[New_index_r1];
    Index_r1 = New_index_r1;
end
if (New_index_r2 <= VTr2_length) begin
    if (Tr2_data[New_index_r2] < New_t) New_t = Tr2_data[New_index_r2];
    Index_r2 = New_index_r2;
end
if (New_index_f1 <= VTf1_length) begin
    if (Tf1_data[New_index_f1] < New_t) New_t = Tf1_data[New_index_f1];
    Index_f1 = New_index_f1;
end
if (New_index_f2 <= VTf2_length) begin
    if (Tf2_data[New_index_f2] < New_t) New_t = Tf2_data[New_index_f2];
    Index_f2 = New_index_f2;
end
//-----
// Add points if dt is larger than Max_dt
//-----
if ((New_t-Old_t) > Max_dt) begin
    Additional_pts = ceil((New_t-Old_t)/Max_dt);
    New_dt = (New_t-Old_t) / Additional_pts;
    for (i = 1; i < Additional_pts; i = i + 1) begin
        // $strobe("\ndif = %e\tAdd = %d\tCL = %d\t T = %e\tNew_dt = ", (New_t-Old_t
        Tcm[Common_length] = Tcm[Common_length-1] + New_dt;
        Common_length = Common_length + 1;
    end
end
//-----
// Update variables for next iteration
//-----
Old_t = New_t;
New_t = Last_t;
//-----
end // while loop

```

The interpolation function, `$table_model()`, allows the module to approximate the behavior of a system by interpolating between user-supplied data points.

### Syntax

```
$table_model( table_inputs, table_data_source, table_control_string );
```

The “LL” is the extrapolation control string and is used to extrapolate beyond the supplied data domain. The first “L” denotes “Linear” extrapolation for the start of the dataset, while the second “L” denotes the same for the end of the dataset.

For more info:

<http://eesof.tm.agilent.com/docs/rfdoc2005A/pdf/rfdeverilogaref.pdf>

Verilog-A Reference Manual (page 80), Agilent Technologies, August 2005

```
//=====
// Calculate the scaling coefficients for each time point along Tcm
//-----
// Store the first voltage point in the common waveform variables
//-----
Vr1_common[2] = $table_model(Tcm[1], Tr1_data, Vr1_data, "LL");
Vr2_common[2] = $table_model(Tcm[1], Tr2_data, Vr2_data, "LL");
Vf1_common[2] = $table_model(Tcm[1], Tf1_data, Vf1_data, "LL");
Vf2_common[2] = $table_model(Tcm[1], Tf2_data, Vf2_data, "LL");

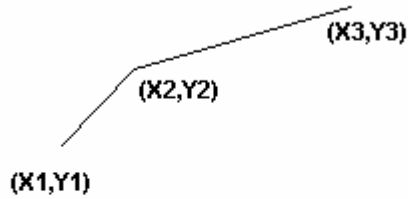
for (i = 1; i <= Common_length; i = i + 1) begin
//-----
// Store the next point (for the derivative calculations)
//-----
if (i < Common_length) begin
    Vr1_common[3] = $table_model(Tcm[i+1], Tr1_data, Vr1_data, "LL");
    Vr2_common[3] = $table_model(Tcm[i+1], Tr2_data, Vr2_data, "LL");
    Vf1_common[3] = $table_model(Tcm[i+1], Tf1_data, Vf1_data, "LL");
    Vf2_common[3] = $table_model(Tcm[i+1], Tf2_data, Vf2_data, "LL");
end

//$strobe("\nTcm = %e\tR1 = %e\tR2 = %e\tF1 = %e\tF2 = %e", Tcm[i], Vr1_con
//-----
// Calculate the derivative of each waveform for C_comp compensation
//-----
if ((i <= 1) || (i >= Common_length)) begin
    dVwfm_r1 = 0.0; // Force the end point derivatives to zero
    dVwfm_r2 = 0.0;
    dVwfm_f1 = 0.0;
    dVwfm_f2 = 0.0;
end else begin
    dVwfm_r1 = ((Vr1_common[2] - Vr1_common[1]) / (Tcm[i] - Tcm[i-1]) +
                (Vr1_common[3] - Vr1_common[2]) / (Tcm[i+1] - Tcm[i])) / 2.0;
    dVwfm_r2 = ((Vr2_common[2] - Vr2_common[1]) / (Tcm[i] - Tcm[i-1]) +
                (Vr2_common[3] - Vr2_common[2]) / (Tcm[i+1] - Tcm[i])) / 2.0;
    dVwfm_f1 = ((Vf1_common[2] - Vf1_common[1]) / (Tcm[i] - Tcm[i-1]) +
                (Vf1_common[3] - Vf1_common[2]) / (Tcm[i+1] - Tcm[i])) / 2.0;
    dVwfm_f2 = ((Vf2_common[2] - Vf2_common[1]) / (Tcm[i] - Tcm[i-1]) +
                (Vf2_common[3] - Vf2_common[2]) / (Tcm[i+1] - Tcm[i])) / 2.0;
end
end
```

Tr1\_data corresponds to Vr1\_data  
The voltage at time point Tcm[i+1] is  
calculated using Linear interpolation  
and the above data

The 'for' loop, that begins above, is used to create the K\* curves. The K\* curves will have 'Common\_length' number of points (default 3000).

The dVwfm\_\* slope calculations above represent the following:



$$\text{Avg Slope} = \frac{1}{2} \left( \frac{Y2-Y1}{X2-X1} + \frac{Y3-Y2}{X3-X2} \right)$$

Where **Y** represents **V\*\_common[]** and **X** represents **Tcm[]**

These calculations are used to calculate the slopes, dVwfm\_\* which will be used to approximate the compensation capacitor currents (shown in the next page); since the current across a capacitor is proportional to the differential (slope) of the voltage across it.

**V\*\_common[2]** represents the current voltage point while **V\*\_common[1]** represents the voltage at the previous point and **V\*\_common[3]** represents the voltage at the next point.

Loops through each voltage in Vr1\_data interpolated to steps for Max\_dt

Currents corresponding to Vt tables & C\_comp

```

//-----
// Calculate intermediate variables and the Kr1 Kr2 coefficients
//-----
Iout1 = ((Vr1_common[2] - Vfx_r1) / Rfx_r1) + C_comp * dVwfm_r1;
Iout2 = ((Vr2_common[2] - Vfx_r2) / Rfx_r2) + C_comp * dVwfm_r2;

Ipc1 = -1.0 * $table_model(Vpc_ref - Vr1_common[2], Vpc_data, Ipc_data, "LL");
Ipc2 = -1.0 * $table_model(Vpc_ref - Vr2_common[2], Vpc_data, Ipc_data, "LL");
Ipu1 = -1.0 * $table_model(Vpu_ref - Vr1_common[2], Vpu_data, Ipu_data, "LL");
Ipu2 = -1.0 * $table_model(Vpu_ref - Vr2_common[2], Vpu_data, Ipu_data, "LL");
Ipd1 = $table_model(Vr1_common[2] - Vpd_ref, Vpd_data, Ipd_data, "LL");
Ipd2 = $table_model(Vr2_common[2] - Vpd_ref, Vpd_data, Ipd_data, "LL");
Igc1 = $table_model(Vr1_common[2] - Vgc_ref, Vgc_data, Igc_data, "LL");
Igc2 = $table_model(Vr2_common[2] - Vgc_ref, Vgc_data, Igc_data, "LL");

```

Currents in PC, GC, PU, PD for both (1 & 2) termination cases

```

Term1 = Iout1 + Igc1 - Ipc1;
Term2 = Ipc2 - Igc2 - Iout2;
Den = (Ipu1 * Ipd2) - (Ipd1 * Ipu2);

// Since these are rising waveforms
// Kr1 == Kpu_on and Kr2 == Kpd_off
Kr1[i] = (Ipd2*Term1 + Ipd1*Term2) / Den;
Kr2[i] = (Ipu2*Term1 + Ipu1*Term2) / Den;

```

Termination #1 (default 0V)

Termination #2 (default 5V)

Kr\* (rising) Tables defined

for PU\_on

for PD\_off

The code shown above shows the algorithm used to calculate the K\* tables. See Section 1.4 for more information on the algorithm.

$$Kr1 = \frac{Ipd2 (Iout1 + Igc1 - Ipc1) + Ipd1 (Ipc2 - Igc2 - Iout2)}{(Ipu1 * Ipd2) - (Ipd1 * Ipu2)}$$

Iout corresponds to Rising waveforms

$$Kr2 = \frac{Ipu2 (Iout1 + Igc1 - Ipc1) + Ipu1 (Ipc2 - Igc2 - Iout2)}{(Ipu1 * Ipd2) - (Ipd1 * Ipu2)}$$

---


$$Kr1 = \frac{Ipu2 (Iout1 + Igc1 - Ipc1) + Ipu1 (Ipc2 - Igc2 - Iout2)}{(Ipu1 * Ipd2) - (Ipd1 * Ipu2)}$$

Iout corresponds to Falling waveforms

$$Kr2 = \frac{Ipd2 (Iout1 + Igc1 - Ipc1) + Ipd1 (Ipc2 - Igc2 - Iout2)}{(Ipu1 * Ipd2) - (Ipd1 * Ipu2)}$$

Similarly for Kf1 & Kf2:

```

//-----
// Calculate intermediate variables and the Kf1 Kf2 coefficients
//-----
Iout1 = ((Vf1_common[2] - Vfx_f1) / Rfx_f1) + C_comp * dVwfm_f1;
Iout2 = ((Vf2_common[2] - Vfx_f2) / Rfx_f2) + C_comp * dVwfm_f2;

Ipc1 = -1.0 * $table_model(Vpc_ref - Vf1_common[2], Vpc_data, Ipc_data, "LL");
Ipc2 = -1.0 * $table_model(Vpc_ref - Vf2_common[2], Vpc_data, Ipc_data, "LL");
Ipu1 = -1.0 * $table_model(Vpu_ref - Vf1_common[2], Vpu_data, Ipu_data, "LL");
Ipu2 = -1.0 * $table_model(Vpu_ref - Vf2_common[2], Vpu_data, Ipu_data, "LL");
Ipd1 = $table_model(Vf1_common[2] - Vpd_ref, Vpd_data, Ipd_data, "LL");
Ipd2 = $table_model(Vf2_common[2] - Vpd_ref, Vpd_data, Ipd_data, "LL");
Igc1 = $table_model(Vf1_common[2] - Vgc_ref, Vgc_data, Igc_data, "LL");
Igc2 = $table_model(Vf2_common[2] - Vgc_ref, Vgc_data, Igc_data, "LL");

Term1 = Iout1 + Igc1 - Ipc1;
Term2 = Ipc2 - Igc2 - Iout2;
Den = (Ipu1 * Ipd2) - (Ipd1 * Ipu2);

// Since these are falling waveforms
// Kf1 == Kpd on and Kf2 == Kpu off
Kf1[i] = (Ipu2*Term1 + Ipu1*Term2) / Den;
Kf2[i] = (Ipd2*Term1 + Ipd1*Term2) / Den;

//$strobe("\nTcom = %e\tKr1 = %e\tKr2 = %e\tKf1 = %e\tKf2 = %e", Tcm[i], Kr1[i], K
//-----
// Shift points by one index for next iteration (for C_comp compensation)
//-----
Vr1_common[1] = Vr1_common[2];
Vr2_common[1] = Vr2_common[2];
Vf1_common[1] = Vf1_common[2];
Vf2_common[1] = Vf2_common[2];

Vr1_common[2] = Vr1_common[3];
Vr2_common[2] = Vr2_common[3];
Vf1_common[2] = Vf1_common[3];
Vf2_common[2] = Vf2_common[3];

end // for loop

```

Same as Kr1 & Kr2 calculations

Equations switched compared to Rising calculations

Points shifted for slope calculation in next iteration

At the end of this 'for' loop all the K\* tables (i.e. Kr1, Kr2, Kf1, Kf2) are initialized.

```

//-----
// Fill the unused portion of the arrays with reasonable values
// to prevent any misbehaving later with $table_model
//-----
for (i = Common_length + 1; i <= Max_length; i = i + 1) begin
    Tcm[i] = Tcm[i-1] + Max_dt;
    Kr1[i] = Kr1[Common_length];
    Kr2[i] = Kr2[Common_length];
    Kf1[i] = Kf1[Common_length];
    Kf2[i] = Kf2[Common_length];
end // for loop
//-----
// End of Common_time / Common_wfm / Kcoef "function"
//-----

//-----
// Initialize logic state variables and kpu, kpd scaling coefficients
//-----
if ((V(En_D) > Vth_R) && (V(In_D) > Vth_R)) begin // Find initial logic state
    pu_on = 1;
    pd_off = 1;
    pd_on = 0;
    pu_off = 0;
end else if ((V(En_D) > Vth_R) && (V(In_D) < Vth_F)) begin
    pu_on = 0;
    pd_off = 0;
    pd_on = 1;
    pu_off = 1;
end else if ((V(En_D) < Vth_F)) begin
    pu_on = 0;
    pd_off = 1;
    pd_on = 0;
    pu_off = 1;
end else begin
    pu_on = 1;
    pd_off = 0;
    pd_on = 1;
    pu_off = 0;
end

if (pu_on == 1) begin
    kpu = Kr1[Common_length];
end else if (pu_off == 1) begin
    kpu = Kf2[Common_length];
end else begin
    kpu = 0.0;
end

if (pd_on == 1) begin
    kpd = Kf1[Common_length];
end else if (pd_off == 1) begin
    kpd = Kr2[Common_length];
end else begin
    kpd = 0.0;
end
//-----
end // @ (initial_step)

```

**Fills in unused spaces with the last value defined**

**En\_D=1 & In\_D=1**

**Vth\_R = 0.8  
Vth\_F = 0.2**

**En\_D=1 & In\_D=0**


**En\_D=0**

**Indeterminate region  
both PU & PD are active**

**// Initialization of kpu, kpd  
// Start with the end of the  
// Vt curves for those which  
// are fully on initially**

**Kr1 terminates to (default) 0V  
Kr2 terminates to (default) 5V  
Kf1 terminates to (default) 5V  
Kf2 terminates to (default) 0V**

**End of initial\_step**



```

graph LR
    In_D --> BUFFER
    En_D --> BUFFER
    BUFFER --> IO

```

```

//=====
// Catch: process to catch threshold crossings in In_D, En_D
//-----
@ (cross(V(In_D)-Vth_R, 1, 100p, 100m)) begin // Rising edge on In_D
  if (V(En_D) > Vth_R) begin
    pu_on = 1;
    pd_off = 1;
    pd_on = 0;
    pu_off = 0;
  end
end

@ (cross(V(In_D)-Vth_F, -1, 100p, 100m)) begin // Falling edge on In_D
  if (V(En_D) > Vth_R) begin
    pu_on = 0;
    pd_off = 0;
    pd_on = 1;
    pu_off = 1;
  end
end

@ (cross(V(En_D)-Vth_R, 1, 100p, 100m)) begin // Rising edge on En_D
  if (V(In_D) > Vth_R) begin
    pu_on = 1;
    pd_off = 1;
    pd_on = 0;
    pu_off = 0;
  end else if (V(In_D) < Vth_F) begin
    pu_on = 0;
    pd_off = 0;
    pd_on = 1;
    pu_off = 1;
  end
end

@ (cross(V(En_D)-Vth_F, -1, 100p, 100m)) begin // Falling edge on En_D
  pu_on = 0;
  pd_off = 1;
  pd_on = 0;
  pu_off = 1;
end

```

Detects In\_D changing from 0 to 1 with time tolerance 100ps and expression tolerance 100mV

Detects In\_D changing from 1 to 0 with time tolerance 100ps and expression tolerance 100mV

Note: All these @ events are run whenever their arguments are validated. Unlike the initial\_step, they are evaluated every cycle

The **cross** function, `cross()`, is used for generating a monitored analog event. It is used to detect threshold crossings in analog signals when the expression crosses zero in the direction specified. The `cross()` function can control the timestep to accurately resolve the crossing. The format is:

$$\text{cross}(\text{expr}[\text{dir}[\text{time\_tol}[\text{expr\_tol}]]]);$$

where

*expr* is a required argument

*dir* is an optional integer expression (-1: negative cross, +1: positive cross)

*time\_tol* and *expr\_tol* are optional arguments that are real and specify the time tolerance and expression tolerance respectively.

```

//-----
// Save time of event when any of the state variables changed
//-----
@(cross(pu_on-0.5, 0)) begin
  Tpu_on_event = $abstime;
  // $strobe("\nPU_ON = %d\tTpu_on = %e\tTpu_off = %e\tTpd_on = %e\tTpd_off = %e", pu_on, Tpu_o:
end

@(cross(pu_off-0.5, 0)) begin
  Tpu_off_event = $abstime;
  // $strobe("\nPU_OFF = %d\tTpu_on = %e\tTpu_off = %e\tTpd_on = %e\tTpd_off = %e", pu_off, Tpu_
end

@(cross(pd_on-0.5, 0)) begin
  Tpd_on_event = $abstime;
  // $strobe("\nPD_ON = %d\tTpu_on = %e\tTpu_off = %e\tTpd_on = %e\tTpd_off = %e", pd_on, Tpu_o:
end

@(cross(pd_off-0.5, 0)) begin
  Tpd_off_event = $abstime;
  // $strobe("\nPD_OFF = %d\tTpu_on = %e\tTpu_off = %e\tTpd_on = %e\tTpd_off = %e", pd_off, Tpu_
end
//-----
// End of process Catch;
//=====

//-----
// This section contains the simultaneous analog equations to find the
// appropriate scaling coefficients according to the state the buffer.
//-----
if (Tpu_on_event > 0.0 || Tpu_off_event > 0.0 ||
    Tpd_on_event > 0.0 || Tpd_off_event > 0.0) begin
  // Look up coefficients in normal operation

  if (pu_on == 1) kpu = $table_model($abstime - Tpu_on_event, Tcm, Kr1, "LL");
  else if (pu_off == 1) kpu = $table_model($abstime - Tpu_off_event, Tcm, Kf2, "LL");
  else
    kpu = Kr1[1];

  if (pd_on == 1) kpd = $table_model($abstime - Tpd_on_event, Tcm, Kf1, "LL");
  else if (pd_off == 1) kpd = $table_model($abstime - Tpd_off_event, Tcm, Kr2, "LL");
  else
    kpd = Kf1[1];
end

//=====
// This section contains the analog expressions which calculate
// the output currents of the IV curves and C_comp capacitors.
//-----
I(pc) <+ -$table_model(V(pc), Vpc_data, Ipc_data, "LL") + kC_comp_pc*C_comp*ddt(V(pc));
I(pu) <+ kpu * -$table_model(V(pu), Vpu_data, Ipu_data, "LL") + kC_comp_pu*C_comp*ddt(V(pu));
I(pd) <+ kpd * $table_model(V(pd), Vpd_data, Ipd_data, "LL") + kC_comp_pd*C_comp*ddt(V(pd));
I(gc) <+ $table_model(V(gc), Vgc_data, Igc_data, "LL") + kC_comp_gc*C_comp*ddt(V(gc));

//-----
// A simple receiver logic
//-----
if (V(pd) > Vinh) V(Rcv_D) <+ 1.0;
else if (V(pd) < Vinl) V(Rcv_D) <+ 0.0;
else V(Rcv_D) <+ 0.5;
//-----
end
endmodule // IBIS_IO

```

Time between event start and current time  
The corresponding Kr1 for this time becomes kpu

C\_comp compensation

Branches defined at the start of this module

This is an IO buffer. This code is used when the buffer is in Input mode

This ends the declaration of the IBIS\_IO module. The other components of the AMS library are similar. The corresponding VHDL-AMS library contains the same functions in VHDL-AMS.

# Appendix D

## Simulating the “IBIS\_IO” model in Dolphin Smash

As mentioned previously, the AMS library was created in both Verilog & VHDL separately, the reason was that most EDA vendors lean towards one or the other. The problem is that not many tool vendors support all the required AMS functions for either one and the ones that do support them have many bugs that haven't been resolved. For example Hspice 2005.09 has bugs in implementing the Verilog keyword 'parameter'- external tables with varying size cannot be passed. A tool that has adequate functionality is Dolphin Smash. Although it can't support Verilog-AMS completely it does support VHDL-AMS. Smash is a mixed signal simulator which supports SPICE, ABCD, VHDL, Verilog-HDL and VHDL-AMS (Verilog-AMS pending). It's capable of mixing all these flavors as well.

The IBIS\_IO model implemented in Appendix E (the VHDL-AMS version) will be stimulated by a pulse source and the outputs will be monitored. The lab\_1\_io50\_t.txt model created to explain the ibis2ams flow in Section 3.2 will be supplying the IBIS data tables.

As shown in the following figure, the pulse source (Vpls) has rise/fall times of 0.1ns and a 20ns period. The output is loaded with a parallel RC structure and terminated to Vcc/2.

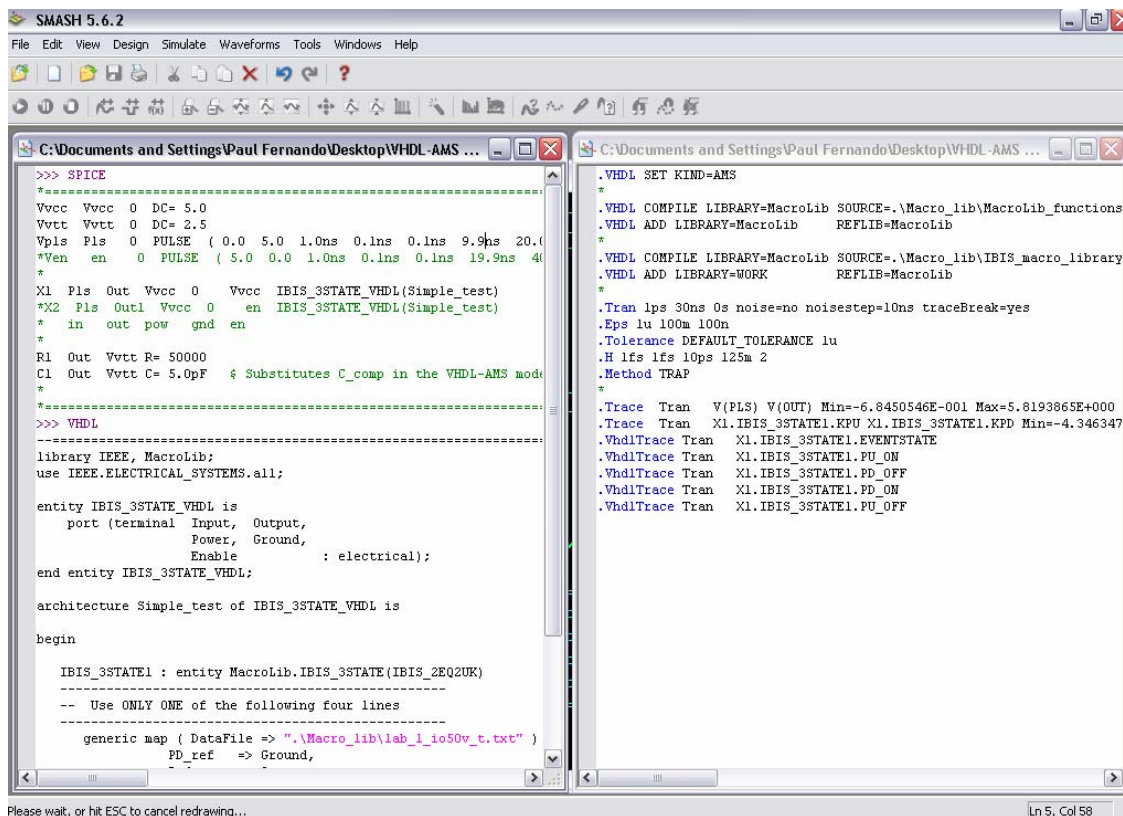


Figure D.1: SMASH GUI

The following plot shows the output ( $V_{out}$ ) and relevant internal variables. Note the KPU & KPD variables; they vary from 0 to 1. They plot the  $K^*$ 's given in section 1.4.



Figure D.2: SMASH output for Risetime = 0.1ns

The same simulation was redone with rise/fall times set to 0.5ns (0.1ns previously). The purpose was to show that the rise/fall times of the input have no effect on the output curve; the only things that matter are the threshold points. Thus, other than a slight shift on the time axis, the outputs look identical to the above. The reason for this right shift is because it takes more time to reach the threshold (switching) points since the slew rates are less.



Figure D.3: SMASH output for Risetime = 0.5ns

# Appendix E

## Case Study I: Output Buffer simulation files

This appendix contains the SMASH simulation files used in section 3.1

### IBIS\_3STATE\_test.nsx

```
>>> SPICE
*=====
===
Vvcc Vvcc 0 DC= 5.0
VGND GND 0 DC= 0
Vvtt Vvtt 0 DC= 0
Vpls Pls 0 PULSE ( 0.0 5.0 1.0ns 1E-10 1E-10 9.9ns 20.0ns )
*
X1 Pls Out Vvcc GND Vvcc IBIS_3STATE_VHDL(Simple_test)
*
R1 Out Vvtt R= 50
C1 Out Vvtt C= 5.0pF $ Substitutes C_comp in the VHDL-AMS model
*=====
===
>>> VHDL
--
=====
=
library IEEE, MacroLib;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity IBIS_3STATE_VHDL is
    port (terminal Input, Output, Power, Ground,
          Enable          : electrical);
end entity IBIS_3STATE_VHDL;

architecture Simple_test of IBIS_3STATE_VHDL is
begin
    IBIS_3STATE1 : entity MacroLib.IBIS_3STATE(IBIS_2EQ2UK)
    -----
    -- Use ONLY ONE of the following four lines
    -----
        generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt" )
        port map ( PU_ref => Power, PD_ref => Ground,
                  Pad => Output, In_D => Input,
                  EN_D => Enable, PC_ref => Power,
                  GC_ref => Ground );
end architecture Simple_test;
--
=====
=
```

## IBIS\_3STATE\_test.pat

```
.VHDL SET KIND=AMS
*
.VHDL COMPILE LIBRARY=MacroLib SOURCE=.\Macro_lib\MacroLib_functions.vhd
.VHDL ADD LIBRARY=MacroLib REFLIB=MacroLib
*
.VHDL COMPILE LIBRARY=MacroLib SOURCE=.\Macro_lib\IBIS_macro_library.vhd
.VHDL ADD LIBRARY=WORK REFLIB=MacroLib
*
.Tran lps 16ns 0s noise=no noisestep=10ns traceBreak=yes
.Eps 1u 100m 100n
.Tolerance DEFAULT_TOLERANCE 1u
.H 1fs 1fs 10ps 125m 2
.Method TRAP
*
.Trace Tran V(PLS) V(OUT) Min=-5.5399397E-001 Max=5.5049085E+000
.Trace Tran I(VGND) I(VVCC) I(VVTT) Min=-6.0209083E-002 Max=6.0099738E-
002
.Trace Tran X1.IBIS_3STATE1.KPU X1.IBIS_3STATE1.KPD Min=-5.0794491E-001
Max=1.1565698E+000
.VhdlTrace Tran X1.IBIS_3STATE1.EVENTSTATE
.VhdlTrace Tran X1.IBIS_3STATE1.PU_ON
.VhdlTrace Tran X1.IBIS_3STATE1.PD_OFF
.VhdlTrace Tran X1.IBIS_3STATE1.PD_ON
.VhdlTrace Tran X1.IBIS_3STATE1.PU_OFF
```

# Appendix F

## Case Study II: Pre-emphasis Buffer simulation files

This appendix contains the SMASH simulation files used in section 3.2

### PreEm.nsx

```
>>> SPICE
*=====
===
* Paul Fernando, NCSU 3/3/2006
*=====
===
*
Vvcc Vvcc 0 DC= 5.0
VGND GND 0 DC= 0
Vvtt Vvtt 0 DC= 0
*Vpls Pls 0 PULSE ( 0.0 5.0 1.0ns 1E-10 1E-10 9.9ns 20.0ns )
.include 'InP.sp'
.param inperiod=10n
.param edge=100p
*
X1 Pls OutP OutN Vvcc Vvcc GND GND Vvcc
IBIS_PreDeMacro_VHDL(Simple_test)
* InD, IOp, IOon, PCref, PUref, PDref, GCref, EnD
*
RP1 OutP Vvtt R= 50
CP1 OutP Vvtt C= 5.0pF $ Substitutes C_comp in the VHDL-AMS model
RN1 OutN Vvtt R= 50
CN1 OutN Vvtt C= 5.0pF $ Substitutes C_comp in the VHDL-AMS model

*****
* T-line model - Provided as spec
*****
*W1 N=2 OutP OutN 0 TL_OUT_POS TL_OUT_NEG 0 RLGCMODEL=ECE733 l=1.

*=====
===
>>> VHDL
--
=====
=
library IEEE, MacroLib;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity IBIS_PreDeMacro_VHDL is
    generic (ScaleBoost: real := -0.9;
             TapDelay: real := 10.0e-9);
    port (terminal InD, IOp, IOon, PCref, PUref, PDref, GCref, En :
          electrical);
```

```

end entity IBIS_PreDeMacro_VHDL;

architecture Simple_test of IBIS_PreDeMacro_VHDL is

terminal Dref, InNM, InPB, InNB : electrical;
terminal PUrefPB, PDrefPB, PCrefPB, GCrefPB : electrical;
terminal PUrefNB, PDrefNB, PCrefNB, GCrefNB : electrical;

begin
  Dig1 : entity MacroLib.IBIS_V
    generic map ( Vdc => 1.0 )
    port map ( p => Dref, n => PDref);

  Inv1 : entity MacroLib.IBIS_VCVS
    port map ( p => Dref, n => InNM, ps => InD, ns => PDref);

  PosM : entity MacroLib.IBIS_IO(IBIS_2EQ2UK)
    generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt" )
    port map ( PU_ref  => PUref,
              PD_ref  => PDref,   Pad      => IOp,
              In_D    => InD,     EN_D    => En,
              PC_ref  => PCref,   GC_ref  => GCref );

  NegM : entity MacroLib.IBIS_IO(IBIS_2EQ2UK)
    generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt" )
    port map ( PU_ref  => PUref,
              PD_ref  => PDref,   Pad      => IOp,
              In_D    => InNM,    EN_D    => En,
              PC_ref  => PCref,   GC_ref  => GCref );

-----TAP

  Dly1 : entity MacroLib.IBIS_VCVS_DELAY
    generic map ( TD => TapDelay )
    port map ( p => InNB, n => PDref, ps => InD, ns => PDref);

  Dly2 : entity MacroLib.IBIS_VCVS_DELAY
    generic map ( TD => TapDelay )
    port map ( p => InPB, n => PDref, ps => InNM, ns => PDref);

  PosB : entity MacroLib.IBIS_IO(IBIS_2EQ2UK)
    generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt"
--      ,kI_pc => ScaleBoost, kI_pu => ScaleBoost,
--      kI_pd => ScaleBoost, kI_gc => ScaleBoost
    )
    port map ( PU_ref  => PUrefPB,
              PD_ref  => PDrefPB,   Pad      => IOp,
              In_D    => InPB,     EN_D    => En,
              PC_ref  => PCrefPB,   GC_ref  => GCrefPB );

  NegB : entity MacroLib.IBIS_IO(IBIS_2EQ2UK)
    generic map ( DataFile => ".\Macro_lib\buffer_driver_t.txt"
--      ,kI_pc => ScaleBoost, kI_pu => ScaleBoost,
--      kI_pd => ScaleBoost, kI_gc => ScaleBoost
    )

```

```

port map ( PU_ref    => PUnrefNB,
           PD_ref    => PDrefNB,   Pad      => IOn,
           In_D      => InNB,      EN_D     => En,
           PC_ref    => PCrefNB,   GC_ref   => GCrefNB );

IpcP : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => PCref, n => IOp, ps => PCref, ns => PCrefPB);
IpuP : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => PUnref, n => IOp, ps => PUnref, ns => PUnrefPB);
IpdP : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => PDref, n => IOp, ps => PDref, ns => PDrefPB);
IgcP : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => GCref, n => IOp, ps => GCref, ns => GCrefPB);

IpcN : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => PCref, n => IOn, ps => PCref, ns => PCrefNB);
IpuN : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => PUnref, n => IOn, ps => PUnref, ns => PUnrefNB);
IpdN : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => PDref, n => IOn, ps => PDref, ns => PDrefNB);
IgcN : entity MacroLib.IBIS_CCCS
generic map (Scale => ScaleBoost)
port map ( p => GCref, n => IOn, ps => GCref, ns => GCrefNB);

end architecture Simple_test;
--
=====
=

```

## PreEm.pat

```
.VHDL SET KIND=AMS
*
.VHDL COMPILE LIBRARY=MacroLib SOURCE=.\Macro_lib\MacroLib_functions.vhd
.VHDL ADD LIBRARY=MacroLib REFLIB=MacroLib
*
.VHDL COMPILE LIBRARY=MacroLib SOURCE=.\Macro_lib\IBIS_macro_library.vhd
.VHDL ADD LIBRARY=WORK REFLIB=MacroLib
*
.Tran lps 80ns 0s noise=no noisestep=10ns traceBreak=yes
.Eps 1u 100m 100n
.Tolerance DEFAULT_TOLERANCE 1u
.H 1fs 1fs 10ps 125m 2
.Method TRAP
*
.Trace Tran V(PLS) V(OUTP) V(OUTN) Min=-5.0260886E-001
Max=5.5002372E+000
*.Trace Tran X1.POSM.KPU X1.POSM.KPD Min=-5.0799639E-001
Max=1.1561627E+000
.Trace Tran X1.POSB.kI_pc X1.NEGB.kI_pc Min=-5.0799639E-001
Max=1.1561627E+000
*.VhdlTrace Tran X1.POSM.EVENTSTATE
*.VhdlTrace Tran X1.POSM.PU_ON
*.VhdlTrace Tran X1.POSM.PD_OFF
*.VhdlTrace Tran X1.POSM.PD_ON
*.VhdlTrace Tran X1.POSM.PU_OFF
*.Trace Tran I(VGND) I(VVCC) I(VVTT) Min=-6.0209083E-002
Max=6.0099738E-002

*.MODEL ECE733 W MODELTYPE=RLGC N=2
*+Lo= 300e-9 60e-9 300e-9
*+Co= 120e-12 -24e-12 120e-12
*+Ro= 0.54 0 0.54
*+Rs= 0.000462 0 0.000462
*+Gd= 1.4e-11 -0.14e-11 1.4e-11
```

# Appendix G

## Case Study III: Equalized Receiver simulation files

This appendix contains the SMASH simulation files used in section 3.3

### EqRcvr.nsx

```
>>> SPICE
*=====
===
* Paul Fernando, NCSU 3/3/2006
*=====
===
*
Vvcc Vvcc 0 DC= 1.8
VGND GND 0 DC= 0
Vvtt Vvtt 0 DC= 0
VplsN INN 0 PULSE ( 0.0 1.8 1.0ns 2E-10 2E-10 1.9ns 4.0ns )
VplsP INP 0 PULSE ( 1.8 0.0 1.0ns 2E-10 2E-10 1.9ns 4.0ns )
*VplsP INP 0 DC= 5.0
*VplsN INN 0 DC= 0.0
*
X1 Vvcc GND INP INN OUT Eq_Rcvr
* PC_ref, GC_ref, Input_p, Input_n, Rcv_D
*
ROUT OUT Vvtt R= 5000
COUT OUT Vvtt C= 5.0pF $ Substitutes C_comp in the VHDL-AMS model

*****
* T-line model - Provided as spec
*****
*W1 N=2 OutP OutN 0 TL_OUT_POS TL_OUT_NEG 0 RLGCMODEL=ECE733 l=1.

*=====
===
>>> VHDL
library IEEE, MacroLib;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity Eq_Rcvr is
    generic (Cpad: real := 1.0e-12; Rt: real := 50.0;
             pas_on: real := 1.0; Rpas: real := 130.0;
             Cpas: real := 0.5e-12; Rt2: real := 1000.0;
             gain: real := 4.0);
    port (terminal PC_ref, GC_ref, Input_p, Input_n, Rcv_D : electrical);
end entity Eq_rcvr;

architecture Simple_test of Eq_Rcvr is
terminal EQ_p, EQ_n, EQSUM : electrical;
```

```

begin

  RT1P : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => Input_p, n => PC_ref);
  CP : entity MacroLib.IBIS_C
    generic map ( Cval => Cpad )
    port map ( p => Input_p, n => PC_ref);
  RPASP : entity MacroLib.IBIS_R
    generic map ( Rval => Rpas )
    port map ( p => Input_p, n => EQ_p);
  CPASP : entity MacroLib.IBIS_C
    generic map ( Cval => Cpas*pas_on )
    port map ( p => Input_p, n => EQ_p);
  RT2P : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => EQ_p, n => PC_ref);

  RT1N : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => Input_n, n => PC_ref);
  CN : entity MacroLib.IBIS_C
    generic map ( Cval => Cpad )
    port map ( p => Input_n, n => GC_ref);
  RPASN : entity MacroLib.IBIS_R
    generic map ( Rval => Rpas )
    port map ( p => Input_n, n => EQ_n);
  CPASN : entity MacroLib.IBIS_C
    generic map ( Cval => Cpas*pas_on )
    port map ( p => Input_n, n => EQ_n);
  RT2N : entity MacroLib.IBIS_R
    generic map ( Rval => Rt )
    port map ( p => EQ_n, n => PC_ref);

  -- This voltage source derives the differential output of the 2 passive
  EQs,
  -- and then amplifies it by the gain parameter.

  EQSUM1 : entity MacroLib.IBIS_VCVS_SUM
    port map ( p =>EQSUM,      n =>GC_ref,
              ps1 =>EQ_p,      ns1 =>GC_ref,
              ps2 =>GC_ref,    ns2 =>EQ_n);
  EQOUT : entity MacroLib.IBIS_VCVS
    generic map ( Scale => (Rpas + Rt2)*(gain/Rt2) )
    port map ( p =>Rcv_d,      n =>GC_ref,
              ps =>EQSUM,     ns =>GC_ref);

end architecture Simple_test;
--
=====
=

```

## EqRcvr.pat

```
.VHDL SET KIND=AMS
*
.VHDL COMPILE LIBRARY=MacroLib SOURCE=.\Macro_lib\MacroLib_functions.vhd
.VHDL ADD LIBRARY=MacroLib REFLIB=MacroLib
*
.VHDL COMPILE LIBRARY=MacroLib SOURCE=.\Macro_lib\IBIS_macro_library.vhd
.VHDL ADD LIBRARY=WORK REFLIB=MacroLib
*
.Tran lps 6ns 0s noise=no noisestep=10ns traceBreak=yes
.Eps 1u 100m 100n
.Tolerance DEFAULT_TOLERANCE 1u
.H 1fs 1fs 10ps 125m 2
.Method TRAP
*
.Trace Tran V(INP) V(INN) Min=-5.0260886E-001 Max=5.5002372E+000
.VhdlTrace Tran X1.EQP X1.EQN
.Trace Tran V(OUT) Min=-5.0260886E-001 Max=5.5002372E+000
.Tolerance DEFAULT_DOT 1
```

# Appendix H

## IBIS plotting utility (s2iplt)

The original version of s2iplt was a Perl script developed by Steve Lipa on 08/24/1995. It was a command line program which simply plotted all the curves in an IBIS file using the gnuplot (<http://www.gnuplot.info/>) utility. The new version of s2iplot provides an easy to use GUI interface (Fig. G.1) using Perl/Tk and also provides the option of creating PostScript files of these curves.

As shown in the figure below, the left column displays the Model name and the specific curve type. Clicking on this column will display the specific curve as shown in Figure G.2. By clicking on the right column, the curves will be dumped into an output PostScript (.ps) file as shown in Figure G.3. The PS file has the default name “modelname\_[curvename].ps”

All the curves can be dumped into a PostScript file by using the file menu and clicking “Print all curves in PS file”.

As in the original program, temporary files are stored in a subdirectory, “s2iplt\_temp”.



Figure H.1: s2iplt GUI main interface

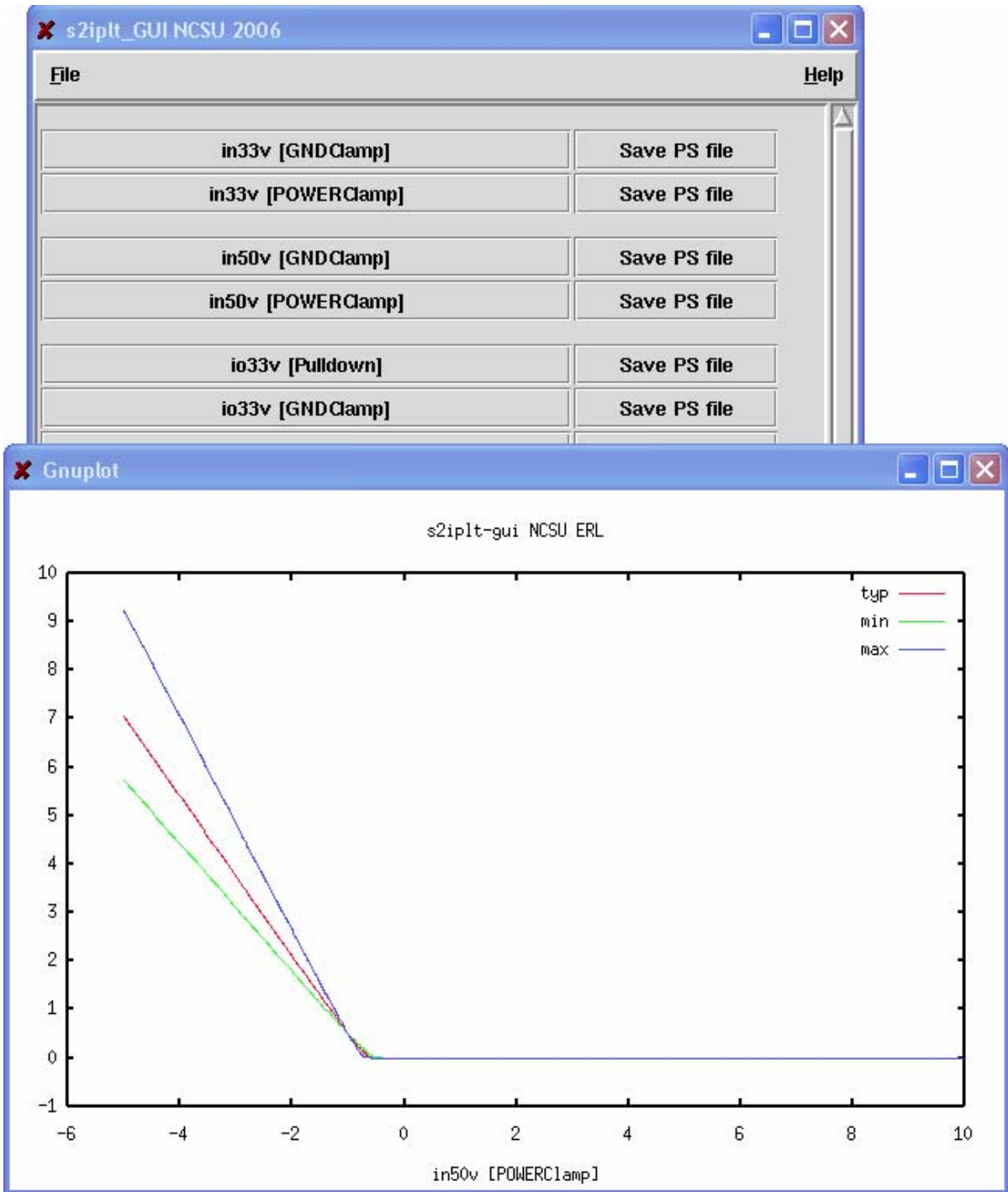


Figure H.2: s2iplt in50v Power Clamp Curve

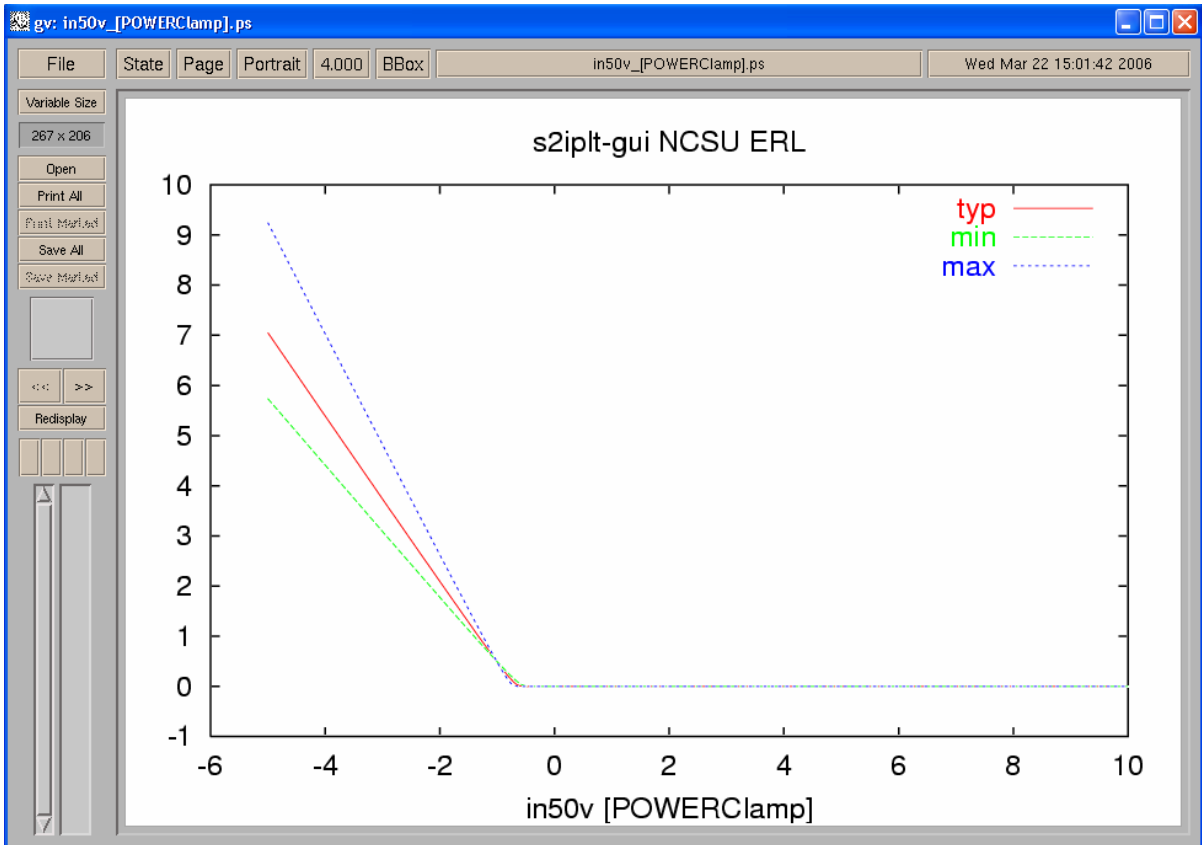


Figure H.3: s2iplt sample PostScript file

# Appendix I

## A Perl/Tk GUI for s2ibis

A GUI was created in Perl/Tk that acts as a GUI for s2ibis. Currently, the GUI can only handle Output, I/O, 3-state and Input buffers. It can be easily updated for future versions of IBIS.

The program gives the model maker the ability to do the following using a single program (window):

1. Open s2i files
2. Edit s2i files
3. Execute the s2i files using s2ibis and create the IBIS file
4. Run ibischk
5. Run s2iplt

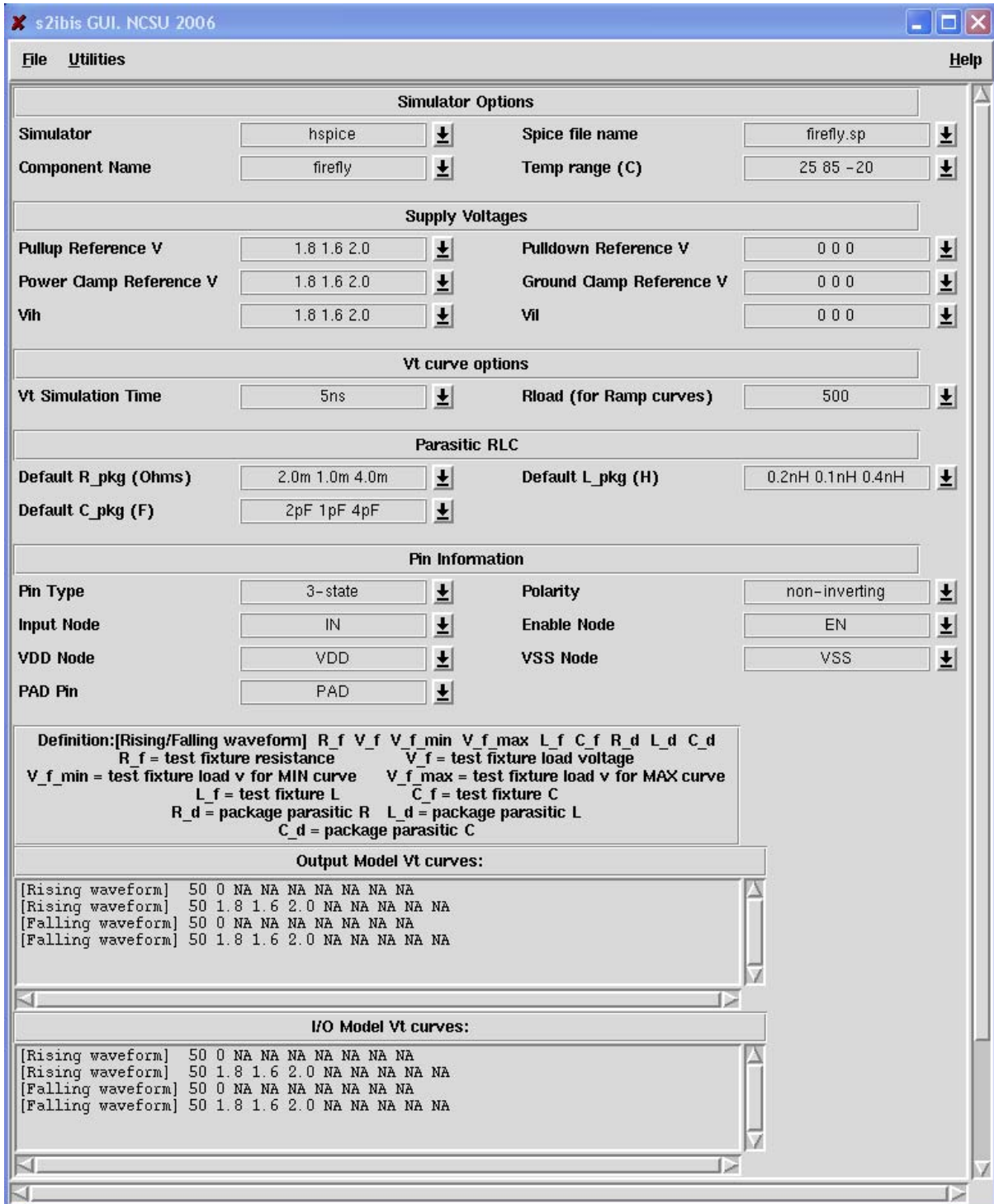


Figure I.1: s2ibis GUI interface