© 2012 Si T. Win

# SIMULATION OF IBIS MODELS USING HSPICE AND ADS

BY

SI T. WIN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Professor Jose Schutt-Aine

# ABSTRACT

Due to timing issues created by high speed applications, simulation of input/output (I/O) buffers with an emphasis on signal integrity has been an important aspect in the design of integrated circuits (IC). Simulation of I/O buffers however proved to be quite difficult. For vendors it would mean releasing proprietary information about their I/O buffers which would not only include a full transistor-level schematic but also full process information such as transistor oxide thickness. In addition, for designers to perform a full transistor-level simulation it would take hours using traditional circuit simulators such as SPICE.

This led to the development of input/output buffer information specification (IBIS) which was developed at Intel to overcome these obstacles. IBIS uses a behavioral modeling method that is based on I-V and V-t curves obtained from either direct measurements or circuit simulations. This provided designers with a model that contains nonproprietary information about I/O buffers and provided a fast alternative method to full transistor-level simulations.

In this thesis we will show how to simulate IBIS models using traditional circuit simulators: specifically Synopsys HSpice and Agilent Advanced Design System (ADS). IBIS has become widely accepted in industry due to the fact that it protects the intellectual property of vendors and because of its standardized format which allows it to be integrated with many circuit simulators.

*To my parents, for their love and support.*

# TABLE OF CONTENTS

LIST OF FIGURES . . . . . v

CHAPTER 1 INTRODUCTION . . . . . 1
1.1 Overview . . . . . 1
1.2 Purpose . . . . . 2
1.3 Content . . . . . 2

CHAPTER 2 INTRODUCTION TO IBIS . . . . . 3
2.1 Basis for IBIS . . . . . 3
2.2 IBIS File Overview . . . . . 4
2.3 Steps to Creating an IBIS file . . . . . 5
2.4 Obtaining I-V data . . . . . 6
2.5 Obtaining V-t data . . . . . 6

CHAPTER 3 USING IBIS MODELS IN HSPICE . . . . . 8
3.1 IBIS Element General Form . . . . . 8
3.2 IBIS Element Data Fields for I/O Buffers . . . . . 10
3.3 Connecting the Circuit . . . . . 12
3.4 Analyzing the Results . . . . . 14

CHAPTER 4 USING IBIS MODELS IN ADS . . . . . 17
4.1 Adding a IBIS Circuit Element . . . . . 17
4.2 Adding the Channel and Input Source . . . . . 19
4.3 Connecting the Circuit . . . . . 19
4.4 Analyzing Simulation Results . . . . . 20

CHAPTER 5 GENERATING EYE DIAGRAMS IN HSPICE AND ADS . . . . . 24
5.1 Eye Diagrams . . . . . 24
5.2 Generating Eye Diagrams in HSpice . . . . . 24
5.3 Generating Eye Diagrams in ADS . . . . . 26

CHAPTER 6 CONCLUSION . . . . . 30

REFERENCES . . . . . 31

# LIST OF FIGURES


2.1 I-V Data Measurement Setup [3] . . . . . 6
2.2 V-t Data Measurement Setup [3] . . . . . 7

3.1 Circuit Block Diagram . . . . . 8
3.2 IBIS General Syntax [5] . . . . . 9
3.3 I/O Buffer Syntax [5] . . . . . 9
3.4 I/O Buffer Diagram [5] . . . . . 10
3.5 Buffer Type and Model Location . . . . . 11
3.6 HSpice Net List . . . . . 13
3.7 Transmitter Input/Output . . . . . 14
3.8 Channel Input/Output . . . . . 15
3.9 Receiver Input/Output . . . . . 16
3.10 Transceiver Input/Output . . . . . 16

4.1 Location of IBIS Elements . . . . . 18
4.2 I/O Buffer Element in ADS . . . . . 18
4.3 Selecting I/O Buffer Model . . . . . 19
4.4 Location of Voltage Range . . . . . 20
4.5 ADS Circuit Schematic . . . . . 20
4.6 ADS Transmitter Input/Output . . . . . 21
4.7 ADS Channel Input/Output . . . . . 22
4.8 ADS Receiver Input/Output . . . . . 22
4.9 ADS Transceiver Input/Output . . . . . 23

5.1 HSpice General Form for a PSRB Source [6] . . . . . 25
5.2 Modified Net List . . . . . 26
5.3 Measurement Setup . . . . . 27
5.4 Eye Diagram using Cosmos Scope . . . . . 27
5.5 ADS PSRB Source . . . . . 28
5.6 Eye Diagram using ADS . . . . . 29
5.7 Eye Diagram using ADS . . . . . 29

# CHAPTER 1

# INTRODUCTION

## 1.1 Overview

As signal speeds increase due to the ongoing effort to achieve faster electronic circuits, hardware designers must now take into account the effects of signal integrity in their design. This includes effects such as cross talk, attenuation, ground bounce, and simultaneous switching noise (SSN). Simulation for the purpose of signal integrity has become an integral step in the design process. However, simulation of I/O buffers proved to be difficult because device models are not readily available from vendors.

The difficulty lies in the fact that device models contain proprietary information which vendors are reluctant to release. In addition, simulation using full transistor-level models would take hours to simulate using modern day simulators. These obstacles led to the development of IBIS models. IBIS is a behavioral method of modeling I/O buffers based on I-V and V-t curve data obtained either from measurements or circuit simulations. It provides a table-based buffer model for semiconductor devices. IBIS allowed vendors to provide information about their product while protecting the intellectual property (IP) of their design.

This alternative approach became widely accepted in industry. IBIS not only provided an alternative approach to full transistor-level models but also a standardized format which allows it to be integrated with many different circuit simulators. In addition, the behavioral model significantly increased the runtime of simulations, which in turn speed up the design process.

## 1.2 Purpose

This thesis details the use of IBIS models in circuit simulation. Its objective is to provide an introduction to IBIS models and explain the basics behind the behavioral model of IBIS. It will also serve as a manual for simulating IBIS models in ADS and HSpice. This thesis documents a step-by-step procedure for simulating I/O buffers in both HSpice and ADS through an easy to follow example.

## 1.3 Content

Because the general purpose of this thesis is to serve as a manual on how to use IBIS models in HSpice and ADS, it has been organized in the following fashion. Chapter 2 begins with an introduction to IBIS where we will discuss the basic theory behind IBIS models. This includes the basis for IBIS models and explains how the models are generated. Chapter 3 will provide an example of how IBIS models are used in HSpice by simulating a simple transceiver circuit. In the example we will use IBIS models to represent the transmitter and receiver. The section will cover the general syntax for an IBIS model and cover optional keywords for IBIS models in HSpice. In Chapter 4 we will repeat the exercise we have done in Chapter 3 but with ADS. This section will include information on how to set up the IBIS element in ADS and will show how to properly connect the circuit. Chapter 5 will illustrate how to generate eye diagrams for the purpose of signal integrity using IBIS models. The chapter will contain information on how to identify jitter in an eye diagram. Finally, Chapter 6, the concluding chapter, will compare the results of the simulation in HSpice and ADS.

# CHAPTER 2

# INTRODUCTION TO IBIS

Before we proceed into the discussion of using IBIS models in computer-aided design, it would be beneficial to first review the content of an IBIS file. Understanding how an IBIS file is generated and why the behavioral model can be used to model I/O buffers is a critical step in the use of IBIS models. This section by no means is intended to replace the IBIS Modeling Cookbook prepared by the IBIS Open Forum. It is strongly recommended that the reader reference the cookbook for further detail in addition to the latest IBIS specification.

## 2.1 Basis for IBIS

From basic transmission line theory we know that a signal can be determined from the voltage carried by the initial wave $V_{initial}$, the voltage reflection coefficient at the load $\Gamma_R$, and the voltage reflection coefficient at the source $\Gamma_S$ where $V_{initial}$, $\Gamma_R$, and $\Gamma_S$ can be expressed in the following manner.

$$V_{initial} = V_s \cdot \frac{Z_L}{Z_L + Z_0}$$

$$\Gamma_L = \frac{Z_L - Z_0}{Z_L + Z_0}$$

$$\Gamma_S = \frac{Z_S - Z_0}{Z_S + Z_0}$$

We often assume that $V_{initial}$, $\Gamma_R$, and $\Gamma_S$ are linear and constant in time. But in the case of buffers, due to its non-linear behavior, we can no longer make these assumptions. Therefore we must replace the linear impedances with non-linear I-V curve elements.

$$V_{initial} = V_s \cdot \frac{Z_L(V,I)}{Z_L(V,I) + Z_0}$$

$$\Gamma_L = \frac{Z_L(V,I) - Z_0}{Z_L(V,I) + Z_0}$$

$$\Gamma_S = \frac{Z_S(V,I) - Z_0}{Z_S(V,I) + Z_0}$$

We must also consider the fact that there is a time dependence on the voltages making $V_{initial}$, $\Gamma_R$, and $\Gamma_S$ all a function of time. Therefore the voltage and current responses for all nodes in the network can be determined by replacing the buffer with the appropriate I-V impedance functions. By using this method, a full-transistor level model is no longer required to characterize a buffer. This was the basis for IBIS developed at Intel [1].

## 2.2 IBIS File Overview

An IBIS file contains all the data necessary to construct a behavioral model used to perform signal integrity simulations and timing analysis of printed circuit boards (PCB). The fundamental information needed to perform these simulations are the buffer's I-V characteristics and switching characteristics (V-t).

One thing to note is that IBIS is only a standard for describing the analog behavior of the buffers of digital devices. It does not specify what an analog simulation application does with the data. In a sense IBIS files are not really models; they only contain the data that will be used to create the model. Simulation tools interpret this behavioral specification in the file to implement their own models and algorithms [2].

In general, an IBIS file characterizes an I/O buffer's behavior using the following information.

- The buffer's output I-V characteristics when the output is in the logic low and logic high state: pullup and pulldown data tables.
- The buffer's output I-V characteristics when the output is forced below ground and above the power supply rails: power clamp and ground clamp data tables.

- The time it takes a buffer's output to switch logic stages (V-t): rising and falling waveforms.
- The buffer's capacitances.

This however will change depending on the type of buffer we are trying to model. For example, an input buffer will only require the following information.

- The buffer's output I-V characteristics.
- The buffer's capacitances.

An IBIS file allows one to model an entire component, not just a particular buffer. Therefore, in addition to the electrical characteristics of a component's buffer, the electrical parameters of the component's package must also be included [2].

## 2.3 Steps to Creating an IBIS file

There are five fundamental steps that go into the process of creating an IBIS model.

- The first step is to perform the pre-modeling activities. This include determining the temperature and process limits over which the IC operates: typ, min, and max.
- Next is to obtain the I-V and V-t data through either measurement or simulation. Usually data is obtained from simulation.
- Once all the data is obtained, it must then be formatted into an IBIS file.
- Once the file is complete, it must be validate by comparing the results of the behavioral model to the transistor model.
- Finally the IBIS model data is then compared to the measured data.

These steps are discussed in further detail in the IBIS cookbook. The fundamental data that is contained in an IBIS file is the I-V and V-t tables. Let us now briefly discuss how these tables are obtained.

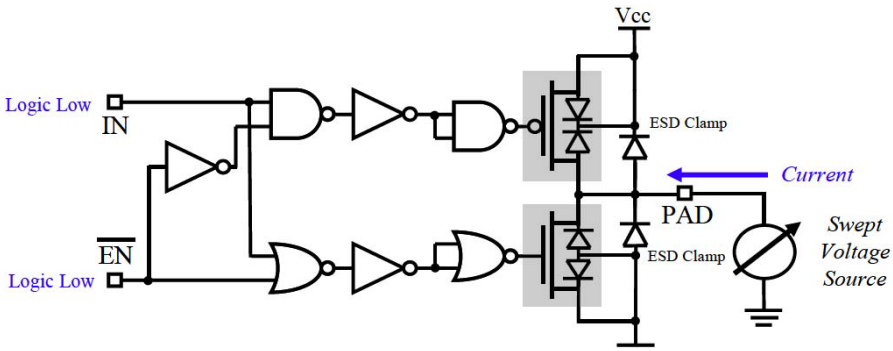



Figure 2.1: I-V Data Measurement Setup [3]

## 2.4 Obtaining I-V data

The first step in extracting the I-V tables is to understand the buffer's operation. In order to obtain the data we must be able to put the buffer's output into its different states: logic low, logic high, and, if applicable, a high impedance state. Next, any electrostatic discharge (ESD) protection diodes should be included in the schematic. Finally, we must understand the buffer's supply voltage reference requirements and connections. Figure 2.1 shows a single stage 3-state buffer with ESD protection. The diagram will help illustrate how the I-V data is measured [4].

All measurements are made at the output node (PAD). The buffer's input is then set to either logic low or logic high depending on if we are constructing the pulldown or pullup data tables, respectively. The voltage source is then swept over $-\mathrm{V}_{CC}$ to $2\mathrm{V}_{CC}$ while recording the current into the buffer. We sweep through $-\mathrm{V}_{CC}$ to $2\mathrm{V}_{CC}$ because when the reflection at the load is at its minimum, $\Gamma = -1$, the voltage at the output can be $-\mathrm{V}_{CC}$, and when the reflection is at his maximum, $\Gamma = 1$, the voltage at the output will be $2\mathrm{V}_{CC}$. This way we cover the full voltage range at the output [2,4].

## 2.5 Obtaining V-t data

The V-t data tables describe the shape of the rising and falling edges of the driver. To obtain the V-t data set we record the voltage at the pad as we switch the input from low to high and from high to low. When switching from

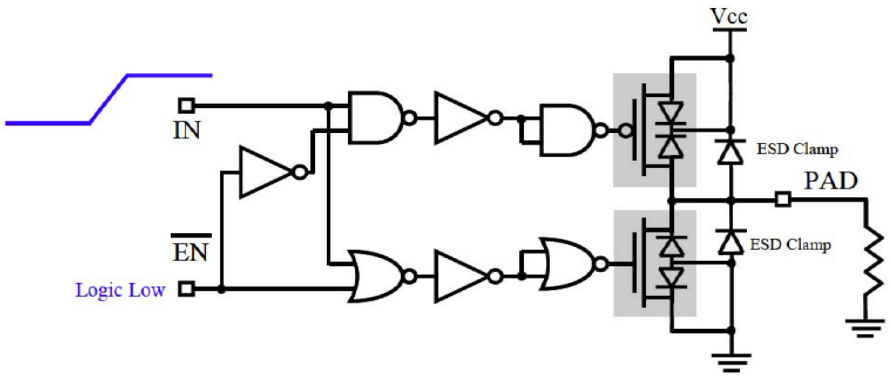



Figure 2.2: V-t Data Measurement Setup [3]

low to high or high to low the V-t table must include the entire waveform. This means that the first entry of the table must be the DC voltage of the output before switching and the last last entry must be the final DC voltage after switching [4]. Figure 2.2 shows the measurement setup for the V-t data. This data is then used to construct the Rising Waveform and Falling Waveform tables in the IBIS file. The number of waveforms required will depend on the type of buffer.

# CHAPTER 3

# USING IBIS MODELS IN HSPICE

In this chapter we will walk through a circuit simulation on a simple transceiver circuit. We will use IBIS models to represent the transmitter and receiver: specifically, the Actel-ttl3f1 I/O buffers. For the channel we will use a lossless transmission line. A simple channel allows us to easily predict the output behavior of our circuit. This way we can verify that our I/O buffers are behaving the way we expect them to and that our simulation produces accurate results. We will be assuming that the reader has some background knowledge in using HSpice.

Figure 3.1 shows a block diagram of the circuit to be simulated. The first step would be to write the corresponding net list for this circuit. Let us begin with the transmitter.

## 3.1 IBIS Element General Form

For the transmitter we will be using the Actel-tt3f1 I/O buffer model. To simulate IBIS models in HSpice, we use the element B. Figure 3.2 shows the general syntax for an IBIS element.

The data fields we must input are heavily dependent on the buffer type. For example, if we were to use an output buffer for our transmitter instead of




Figure 3.1: Circuit Block Diagram

```
bxxx node_1 node_2 ... node_N
+ file='filename' model='model_name'
+ keyword_1=value_1 ... [keyword_M=value_M]
```

Figure 3.2: IBIS General Syntax [5]

```
B_IO nd_pu nd_pd nd_out nd_in nd_en V_out_of_in [nd_pc nd_gc]
+ file='file_name' model='model_name'
+ [typ={typ|min|max|fast|slow}] [power={on|off}]
+ [buffer={3|input_output}]
+ [xv_pu=state_pu] [xv_pd=state_pd]
+ [interpol={1|2}]
+ [ramp_fwf={2|1|0}] [ramp_rwf={2|1|0}]
+ [fwf_tune=fwf_tune_value] [rwf_tune=rwf_tune_value]
+ [nowarn]
+ [c_com_pu=c_com_pu_value]
+ [c_com_pd=c_com_pd_value]
+ [c_com_pc=c_com_pc_value]
+ [c_com_gc=c_com_gc_value]
+ [pu_scal=pu_scal_value]
+ [pd_scal=pd_scal_value]
+ [pc_scal=pc_scal_value]
+ [gc_scal=gc_scal_value]
+ [rwf_scal=rwf_scal_value]
+ [fwf_scal=fwf_scal_value]
+ [spu_scal=spu_scal_value]
+ [spd_scal=spd_scal_value]
```

Figure 3.3: I/O Buffer Syntax [5]

an I/O buffer, we would only need to specify a total of 4 to 6 nodes instead of 6 to 8 nodes. Not only will the number of nodes change but how we connect those nodes will also depend on the buffer type.

### 3.1.1 I/O Buffer General Form

Since we are dealing with an I/O buffer for our transmitter, let us look at the general syntax for an I/O buffer type. The general syntax for an I/O buffer type is shown in Figure 3.3.

Now that we have the general form for our I/O buffer, we are now ready to begin filling out the data fields.

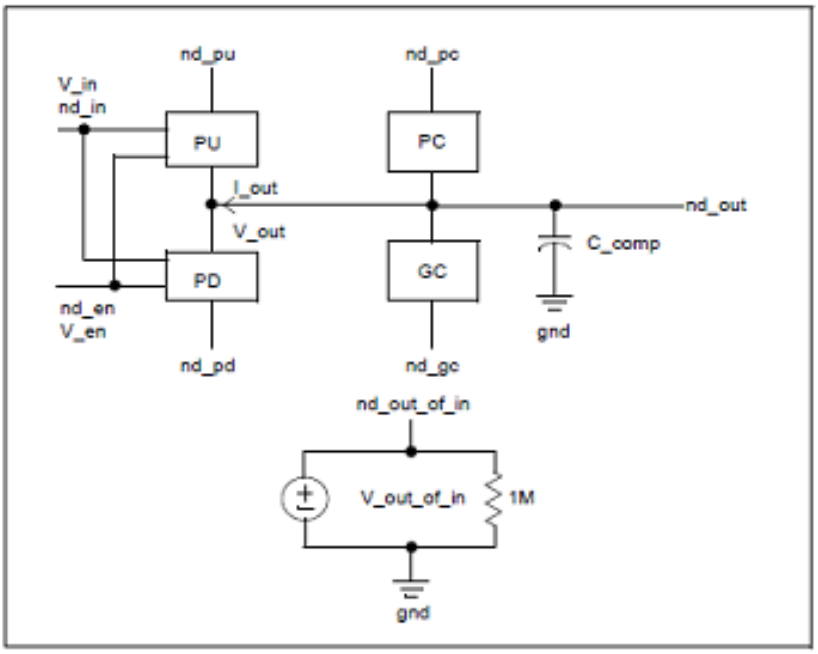



Figure 3.4: I/O Buffer Diagram [5]

## 3.2 IBIS Element Data Fields for I/O Buffers

First we will name our element. Here I have chosen the name bTrans. However, any name is valid as long as it starts with the letter b and is followed by 1023 alphanumeric characters.

Next we will name our nodes. We can see from Figure 3.4 that we have a total of 8 nodes for an I/O buffer excluding the ground node: nd_in, nd_en, nd_pu, nd_pd, nd_pc, nd_gc, nd_out, and nd_out_of_in. Every single node is required accept for the nd_pc and nd_gc nodes, which are optional. That is why the total number of external nodes is either 6 or 8 for an I/O buffer. Those two nodes are in case our buffer model has ESD protection. How we connect these nodes will be explained in Section 3.3.

Now we specify the IBIS file we wish to use with the keyword *file*. The format will be +file = 'file_name.ibis'. If the file that we wish to use is in a different directory, we can specify the path to the file as the file name. Since many IBIS files have multiple models, we specify which model we wish to simulate with the keyword *model*. In our case we are using the model tt3f1 so we type +model = tt3f1. These are the only required data fields. However,

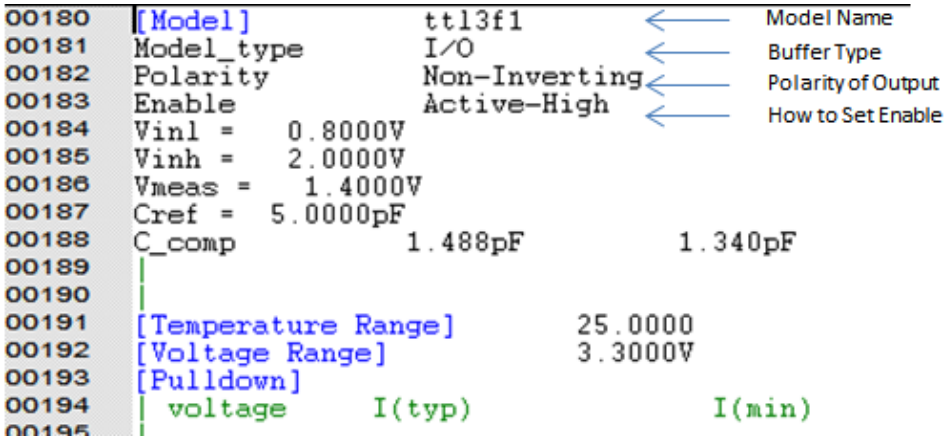



Figure 3.5: Buffer Type and Model Location

there are some optional fields that are helpful. One of these is the buffer type. Since we are using an I/O buffer, our buffer type is 3. We specify this by typing +buffer = 3. By doing this, HSpice will do a check to make sure the buffer type we specified matches that of the model buffer type. Figure 3.5 shows where to find the model name and the buffer type in an IBIS file.

### 3.2.1 Data Fields: Optional Keywords

The option keywords, if not specified, will be set to their default values. These data fields allow us to further specify how the circuit simulator should interpret and process our IBIS model. Such as specifying which ramp data to use, what type of interpolation algorithm to perform, and which processor corner to select. The HSpice Signal Integrity User Guide provided by Synopsis contains detailed information on these optional keywords and their default values.

Now that we are finished with adding the transmitter, we will repeat the process to add another I/O buffer for the receiver. The data fields will be identical to those of the transmitter. However, the nodes of the buffer will change slightly. This will be shown in Section 3.3.

Now in our net list we should have two IBIS elements. For our channel, we will just use a lossless transmission line with a characteristic impedance of 50 ohms and a normalized electrical length of 0.05 $\lambda$ at 250 MHz.

## 3.3 Connecting the Circuit

Now that we have the core elements to our circuit, we can now begin to connect the nodes. Let us start with the transmitter.

First we will connect our digital input signal into node nd_in. For this simulation we will use a simple pulse that toggles between 0 and 1 volt, a rise and fall time of 100 ps, a delay of 100 ps, a 2 ns high time, and a period of 4 ns. Next we will connect our clock to node nd_en. From Figure 3.5 we can see for this particular model, it is active-high enabled. This means that the buffer will enter the enabled state if the voltage applied to node nd_en is greater than 0.5 V. Using this information we can set our clock accordingly. For simplicity we will just add a DC voltage source of 1 V to our enable node.

HSpice by default will connect nodes nd_pu and nd_pd using the voltage values specified in the model. It will also connect nd_pc and nd_gc if applicable. If we wish to supply our own external sources, such as the case where we want to account for ground bounce, we would then want to use the keyword power and set +power = off. Otherwise we never connect any external sources to these four nodes. Also, we will leave node nd_out_of_in hanging. This node is used if we are using our I/O buffer as a input buffer instead of a output buffer. The last thing to do is to connect the output of our buffer, nd_out, to our channel. We do not have to worry about adding the package RLC values. By default HSpice will use these values if they are provided by the IBIS model. Once again we can use the keyword package if we do not wish to include package RLC values.

Finally we must connect the receiver to our circuit. Since we are using our I/O buffer as an input buffer now, the connections will be slightly different from the transmitter. The output of our channel, which is an analog signal, will be connected to the node nd_out instead of nd_in. This may seem confusing, but it is best to think of nd_out as an analog port and nd_in as a digital port. Once again we will leave nodes nd_pu, nd_pd, nd_gc, and nd_pc hanging, as it will be taken care of by HSpice. This time we will leave nd_in hanging instead of nd_out_of_in, since we are now using the buffer as an input buffer. Node nd_out_of_in is the digital output of our receiver. Now we have completed our transceiver circuit. The resulting HSpice net list is shown in Figure 3.6.

To run the simulation, first place both the net list and the IBIS model file

```
actel2.cir TransRec simulation using actel-ttl3f1.ibs

Btrans pullup pulldown outp in enable Voi1 powerclamp groundclamp
+file = 'actel-ttl3f1.ibs'
+model = 'ttl3f1'
+typ = typ
+power = on
+buffer = 3


T1 outp 0 in2 0 Z0 = 50 F = 250meg NL = .05

Brecev pullup2 pulldown2 in2 open2 enable2 Voi2 powerclamp2 groundclamp2
+file = 'actel-ttl3f1.ibs'
+model = 'ttl3f1'
+typ = typ
+power = on
+buffer = 3

R1 Voi2 0 50

vVDD enable 0 1
VIN in 0 pulse( 0 1 100ps 100ps 100ps 2ns 4ns )

.tran 1ps 1000ns

.option post
.end
```

Figure 3.6: HSpice Net List

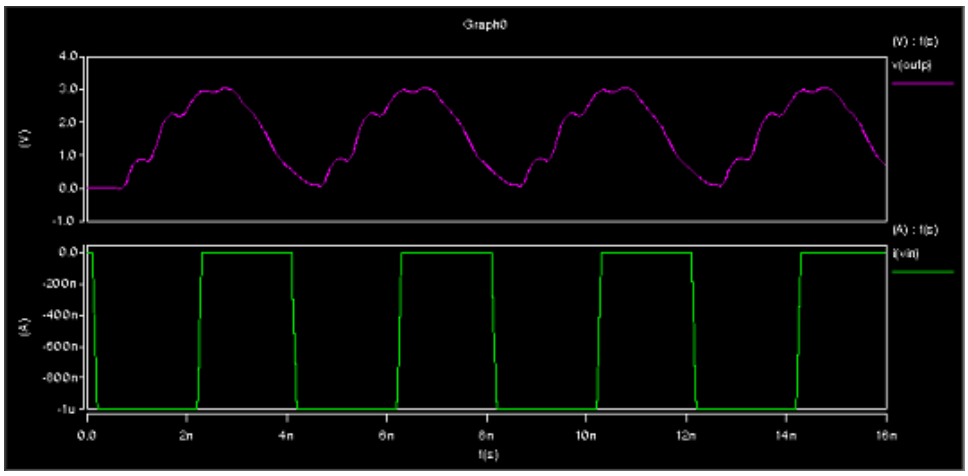



Figure 3.7: Transmitter Input/Output

in the same directory. Then change the current directory to the one that contains the net list and the IBIS model file. Next run HSpice by typing the following command: *bash -c "module load synopsys/hspice-E-2010.12-SP2 ;hspice ACT.cir>ACT.out"*. In this case ACT.cir represents the name of net list file. ACT.out is the simulation output file which will provide a summary of the simulation. ACT.out will indicate any problems that HSpice encountered during the simulation process. This file can be used for debugging if any errors should arise. One such error would be that it was unable to parse one of the devices in the net list.

If no errors occurred during simulation, then the file ACT.tr0 will be created which contains the results of the transient simulation.

## 3.4 Analyzing the Results

In this section we will analyze the results of the simulation. The plots were obtained using CosmosScope using the following command in the Linux terminal: *bash -c "module load synopsys/CosmosScope-E-2011.03 ;cscope"*.

### 3.4.1 Transmitter Input/Output

Let us first look at the output of our transmitter and compare it to our digital input signal. This is shown in Figure 3.7.

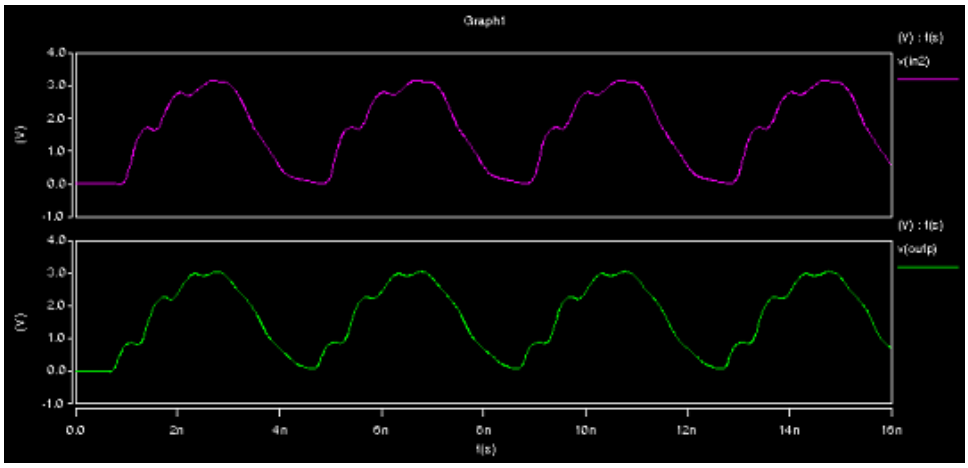



Figure 3.8: Channel Input/Output

We can see our transmitter is behaving the way we expect it to. It is converting our digital signal into an analog signal with a slight delay. In addition, we can see our package RLC values are distorting our output analog signal.

### 3.4.2 Channel Input/Output

Let us now look at the signal at the end of our channel and compare it to the output of our transmitter.

From Figure 3.8 we can see that our signal is being delayed at the output by a slight amount. This is what we expect from a lossless transmission line. A lossless transmission does nothing but delay the signal. The delay amount is determined by how long it takes the signal to travel down the channel.

### 3.4.3 Receiver Input/Output

Next let us compare the signal going into the input of our receiver to the signal at the output of the receiver. This is shown in Figure 3.9.

Our receiver is also behaving the way we expect it. It is converting our analog signal back to a digital signal. We can see how our threshold is being set in HSpice. From Figure 3.5 we can see that our Vinl is 0.8 V and Vinh is 2.0 V. When the output voltage is less than Vinl it is 0 V, and when it is greater than Vinh it is 1 V. For the cases where it is not defined, when the

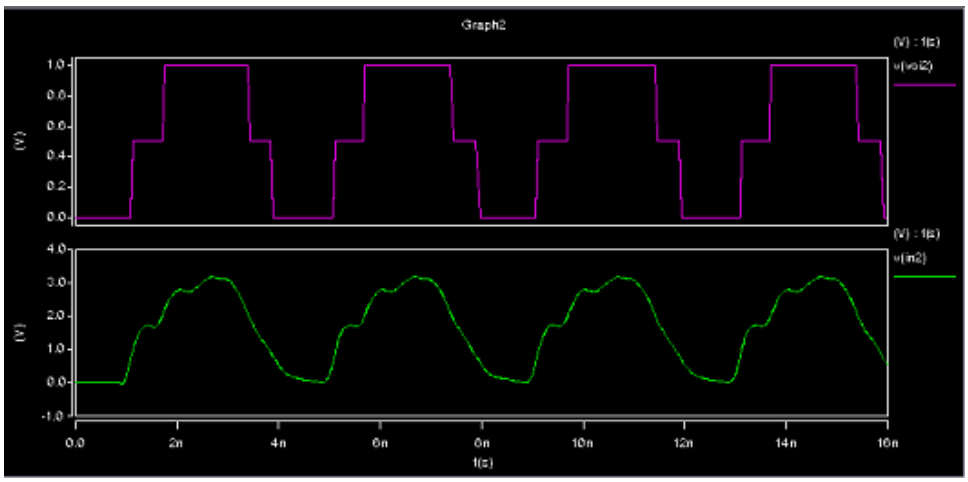



Figure 3.9: Receiver Input/Output

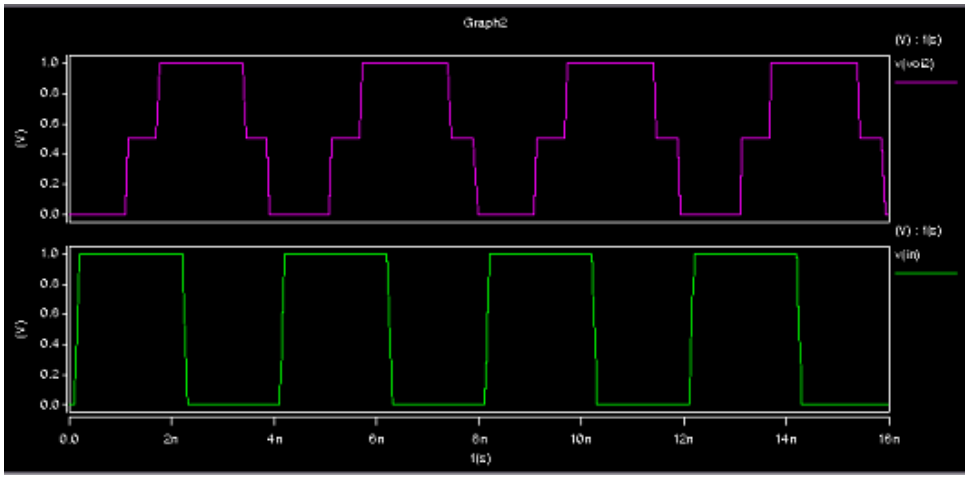



Figure 3.10: Transceiver Input/Output

output voltage is between Vinh and Vinl, the output is set to 0.5 V. This is to preserve the logic levels 0 and 1 for the low and high states.

### 3.4.4 Transceiver Input/Output

From these results we can conclude that we have successfully simulated our transceiver circuit using IBIS models. This was confirmed by looking at every stage of our circuit and making sure that our actual data matched what we expected.

The final graph, Figure 3.10, shows the input and output of our transceiver.

# CHAPTER 4

# USING IBIS MODELS IN ADS

In this chapter we will repeat the exercise we have done in Chapter 3 but this time perform the simulation in ADS. By using the same circuit, we can compare our results to that of the HSpice simulation results. This way we can further verify our simulations. Also, we will be able to see the slight difference in the way HSpice and ADS process the data in the IBIS file. Let us once again begin with the transmitter. Once again we are assuming that the reader has prior experience with the ADS design tool

## 4.1 Adding a IBIS Circuit Element

In ADS the IBIS circuit elements are located under the section *Signal-Integrity IBIS*. This is shown in Figure 4.1. Here we will find a list of all the different types of buffers that ADS supports. Since our Actel-tt3f1 is a I/O buffer we must select the IBIS IO component. Figure 4.2 shows what the I/O buffer element looks like in ADS.

To configure the I/O buffer, double click on the component. Here the only two required fields are the name and the file path to the IBIS file. Once the file path to the IBIS file is set, we can select the model we wish to simulate by clicking on the *pin* tab and selecting the model we want. This is shown in Figure 4.3. Once we select our model we can view information about that model by clicking on the model tab. Here we can find key information such as the model type, the polarity of the buffer, and if it is active-high or active low.

Repeat this process so we have two I/O buffers. One for the transmitter and one for the receiver.

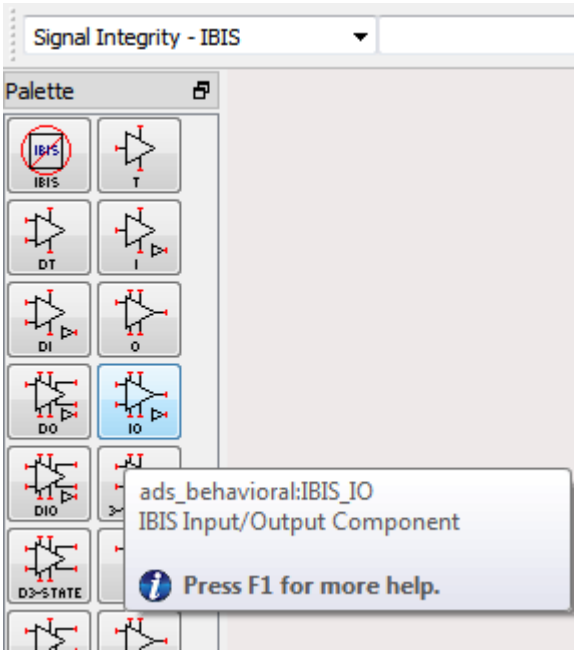



Figure 4.1: Location of IBIS Elements

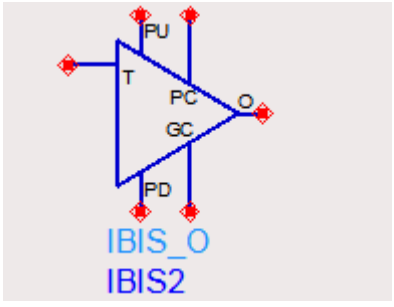



Figure 4.2: I/O Buffer Element in ADS

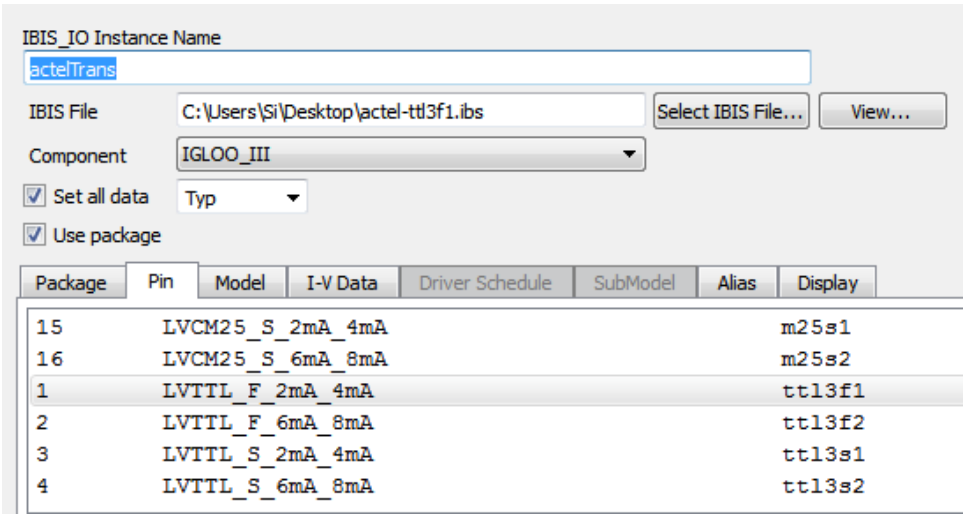



Figure 4.3: Selecting I/O Buffer Model

## 4.2 Adding the Channel and Input Source

Next we want to add our channel and digital Input. For our channel we want to use the same lossless transmission line we used before. The lossless transmission line component, TLIN, can be found under *TLines-Ideal*. Once selected, configure the transmission line so that it has a characteristic impedance of 50 Ω and a normalized electrical length of 0.05 $\lambda$ at 250 MHz.

For the digital input we want a voltage pulse. The component we are looking for is called VtPulse and it can be found under *Sources-Time Domain*. Once selected set Vlow to 0 V, Vhigh to 1 V, delay to 100 ps, edge to be linear, rise and fall to 100 ps, width to 2ns, and period to 4 ns. Now we are ready to connect our circuit.

## 4.3 Connecting the Circuit

First connect the digital source to the transmitter port T. Next connect the ports PU and PC to a DC voltage source. How much voltage to supply can be found under *Voltage Range* under the *I-V Data* tab. This is shown in Figure 4.4. Since we are running a typ simulation, supply a DC voltage of 3.3 V. Next connect PD and GC to ground. Since our I/O buffer is active-high enabled, connect the port E to a 1 V DC voltage supply. Finally connect the

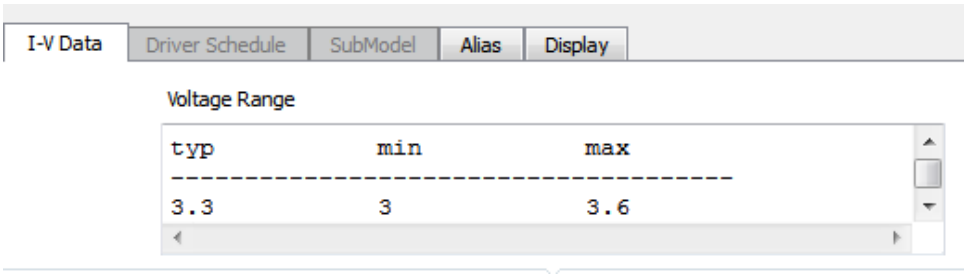



Figure 4.4: Location of Voltage Range

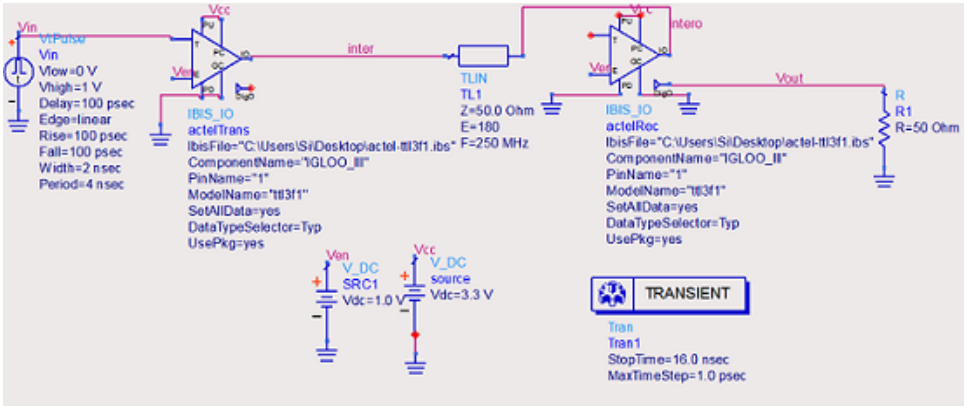



Figure 4.5: ADS Circuit Schematic

port I/O to the input of our channel. For the transmitter leave the DigO port hanging.

Let us now connect the receiver. Connect the output of the channel to the I/O port of the receiver. Connect PU, PC, PD, E, and GC in the same manner as the transmitter. Now leave port T hanging and connect the DigO port to a 50 Ohm load termination. Once all the connections have been made/ it should look like Figure 4.5.

Once everything is set, we are now ready to simulate. In this example the transient simulation was run for 16 ns with a step of 1 ps.

## 4.4 Analyzing Simulation Results

The results of the ADS simulation are shown below. Let us analyze each component individually.

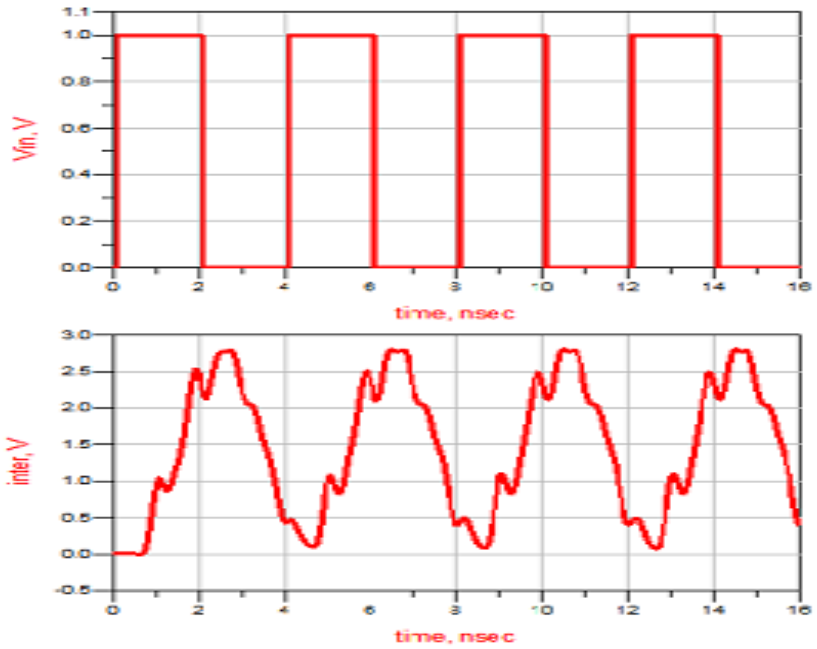



Figure 4.6: ADS Transmitter Input/Output

### 4.4.1 ADS Transmitter Input/Output

Let us first look at the output of our transmitter and compare it to our digital input signal. This is shown in Figure 4.6.

This is identical to that of the HSpice simulation. As we expect it is converting our digital signal into an analog signal with a slight delay. Once again we can see the distortion cause by the package's parasitics.

### 4.4.2 ADS Channel Input/Output

Let us now look at the channel input and output. This is shown in Figure 4.7.

Once again we are seeing exactly what we expect. We are seeing a signal that is delayed slightly at the output.

### 4.4.3 ADS Receiver Input/Output

Next let us compare the input of our receiver to the receiver output. This can be seen in Figure 4.8.

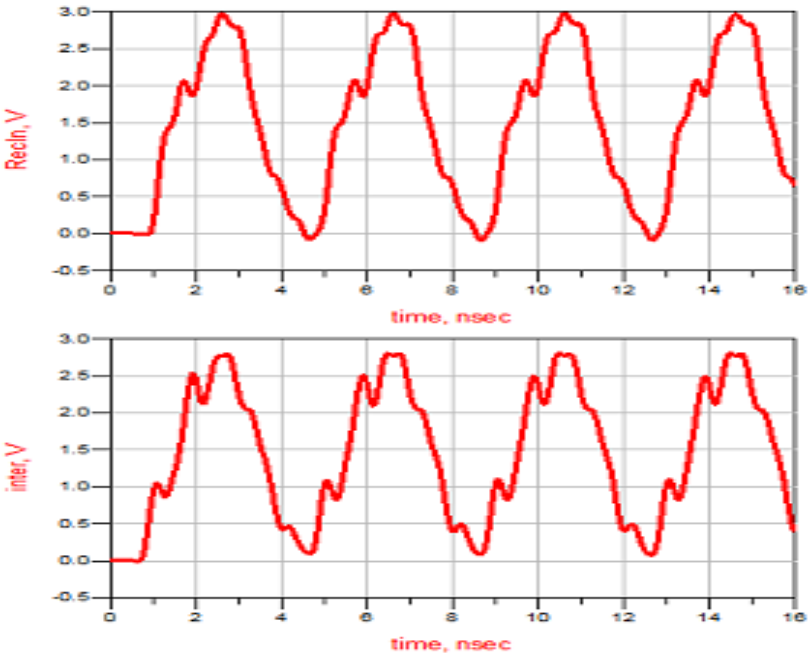



Figure 4.7: ADS Channel Input/Output

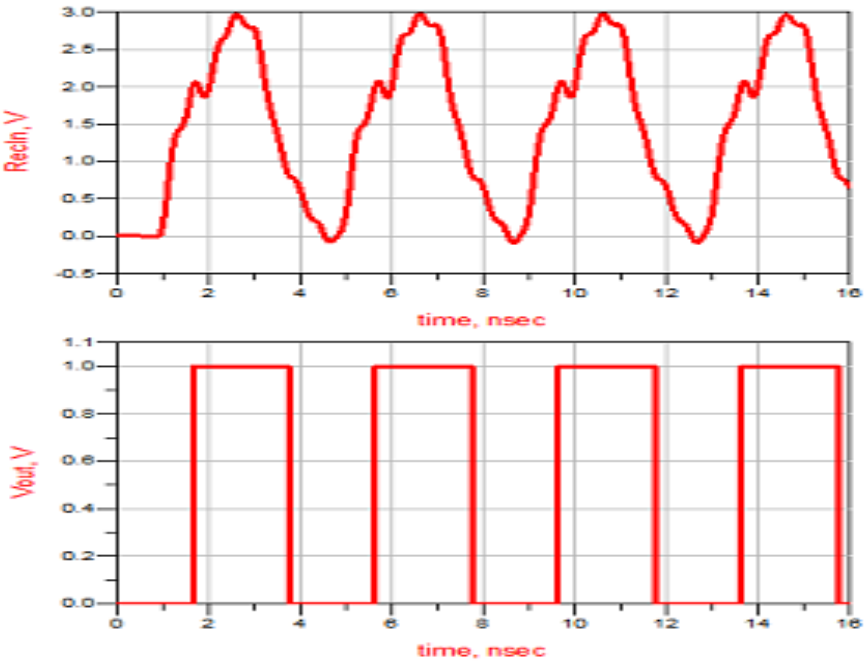



Figure 4.8: ADS Receiver Input/Output

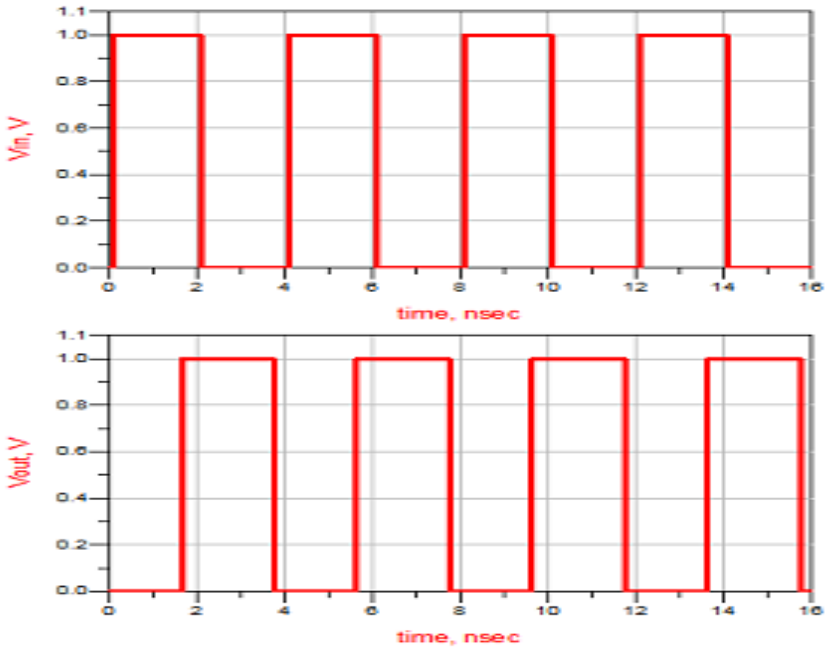



Figure 4.9: ADS Transceiver Input/Output

The receiver is also behaving the way we expect it to. It is converting the analog input back to a digital output. Unlike HSpice, ADS seems to threshold the voltage in a slightly different mannter. This is a great example of how different circuit simulators utilize the IBIS data differently.

### 4.4.4 ADS Transceiver Input/Output

From these results we can conclude that we have successfully simulated our transceiver circuit using ADS. This was confirmed by looking at every stage of our circuit and comparing it to the data we gathered from the HSpice simulations. The final graph, Figure 4.9, shows the input and output of our transceiver using ADS.

# CHAPTER 5

# GENERATING EYE DIAGRAMS IN HSPICE AND ADS

Using eye diagrams is one of the most complete ways to analyze a signal after transmission for signal integrity purposes. Eye diagrams display visually how much jitter is present in the signal. Just by seeing if the eye is open or closed, we can see how clean a signal is at the output. That is why being able to generate eye diagrams is an important skill to have. In this section we will generate eye diagrams from the circuits we simulated in Chapters 3 and 4.

## 5.1 Eye Diagrams

An eye diagram is an oscilloscope display of a digital signal triggered on a particular cycle of the clock. Each period is then superimposed on top of each other to construct the eye diagram.

Usually eye diagrams are generated using a pseudo random bit generator so that all possible bit sequences are generated. Then the simulation is run until a large sample size is obtained. This usually means in excess of 100 million samples.

## 5.2 Generating Eye Diagrams in HSpice

To generate eye diagrams in HSpice is relatively straightforward. To illustrate, we will use the same circuit we used in Chapter 3 but with some slight modifications. One modification is to replace our source with a pseudo random bit generator source (PSRBS). The general form for the PRBS and the descriptions for its data fields are shown in Figure 5.1. One thing to keep in mind when setting the rate and the taps is to make sure that every possible bit sequence is being generated.

**Pseudo Random-Bit Generator Source (PRBS Function)**

| | |
|---|---|
| General Form | vxxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall rate + seed <[> taps <]> <rout=val> <)> |
| Or | Ixxx n+ n- LFSR <(> vlow vhigh tdelay trise tfall rate + seed <[> taps <]> <rout=val> <)> |
| LFSR | Specifies the voltage/current source as PRBS. |
| vlow | The minimum voltage/current level. |
| vhigh | The maximum voltage/current level. |
| tdelay | Specifies the initial time delay to the first transition. |
| trise | Specifies the duration of the onset ramp (in seconds), from the initial value to the pulse plateau value (reverse transit time). |
| tfall | Specifies the duration of the recovery ramp (in seconds), from the pulse plateau, back to the initial value (forward transit time). |
| rate | The bit rate. seed The initial value loaded into the shift register. |
| taps | The bits used to generate feedback. |
| rout | The output resistance. |

Figure 5.1: HSpice General Form for a PSRB Source [6]

```
Actel ttl3f1

Btrans pullup pulldown outp in enable Voi1 powerclamp groundclamp
+file = 'actel-ttl3f1.ibs'
+model = 'ttl3f1'
+typ = typ
+power = on
+buffer = 3


T1 outp 0 in2 0 Z0 = 50 F = 250meg NL = .05

Brecev pullup2 pulldown2 in2 open2 enable2 Voi2 powerclamp2 groundclamp2
+file = 'actel-ttl3f1.ibs'
+model = 'ttl3f1'
+typ = typ
+power = on
+buffer = 3

R1 Voi2 0 50

vVDD enable 0 3.3
VIN in 0 LFSR ( 0 1 100ps 100ps 100ps 500meg 'pow(2,7)-1' [7,6] rout=50)

.tran 1ps 1000ns

.option post
.end
```

Figure 5.2: Modified Net List

Once we have replaced our source, we need to now modify our transient simulation settings. We must now run the simulation long enough so that we can fully capture the behavior of the signal in the eye diagram. Here the transient simulation is run for 1000 ns and the output is printed out every 1 ps. The modified net list is shown in Figure 5.2.

Now that we have our data, we can generate the eye diagram in CosmosScope. Once CosmosScope is open, go to tools → measurement → time domain → eye diagram. From here select Ideal trigger. Next we must fill out the three parameters time base, start X value, and end X value. Time base sets the time period for the eye diagram. Start X value and End X value should be set to the start and end time of the transient simulation [6]. All of this can be seen in Figure 5.3. Once those fields are set click apply. The eye diagram should look like Figure 5.4.

## 5.3 Generating Eye Diagrams in ADS

To generate eye diagrams in ADS we must make the same changes we made with the HSpice example. We must replace our source with a pseudo random

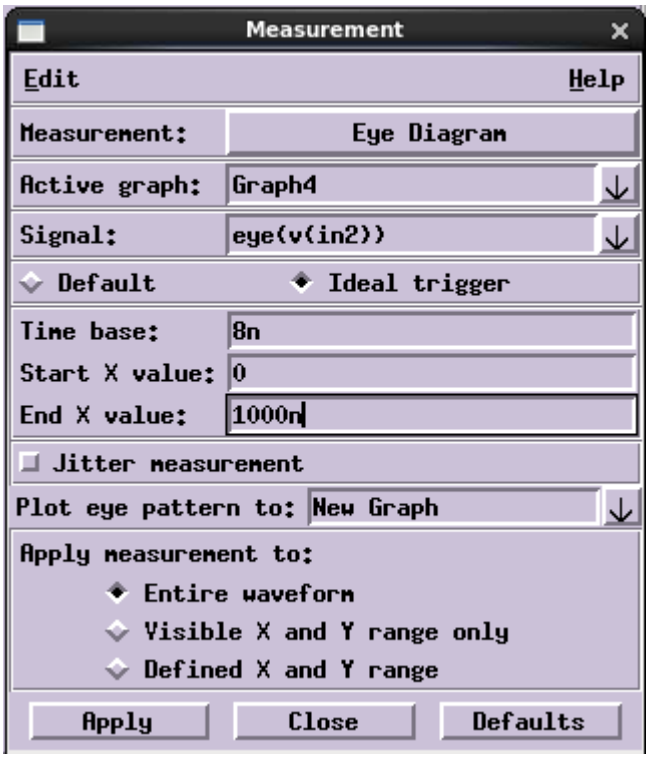



Figure 5.3: Measurement Setup

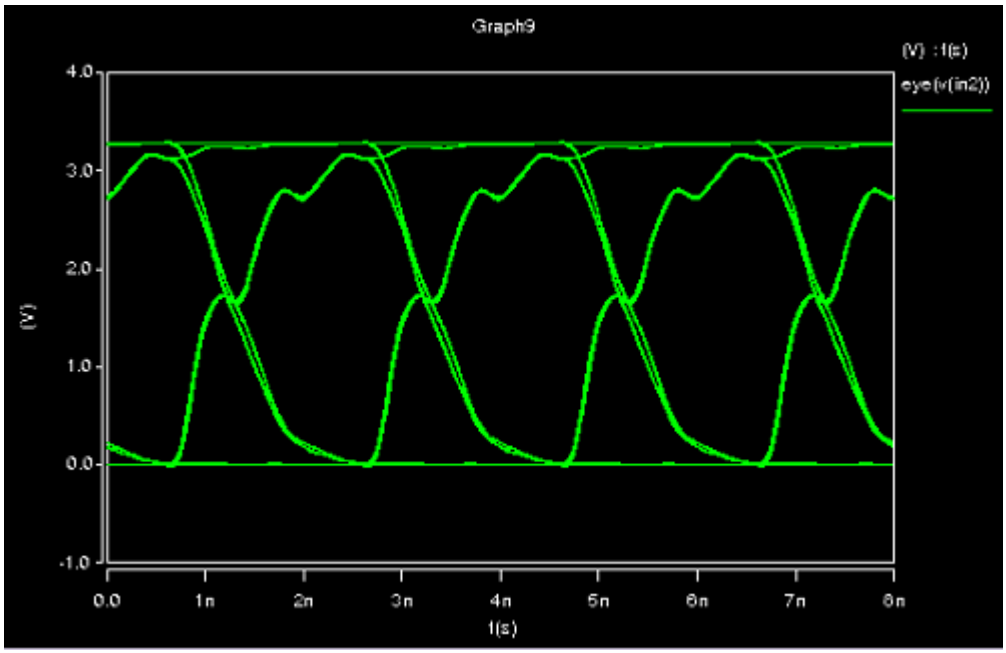



Figure 5.4: Eye Diagram using Cosmos Scope

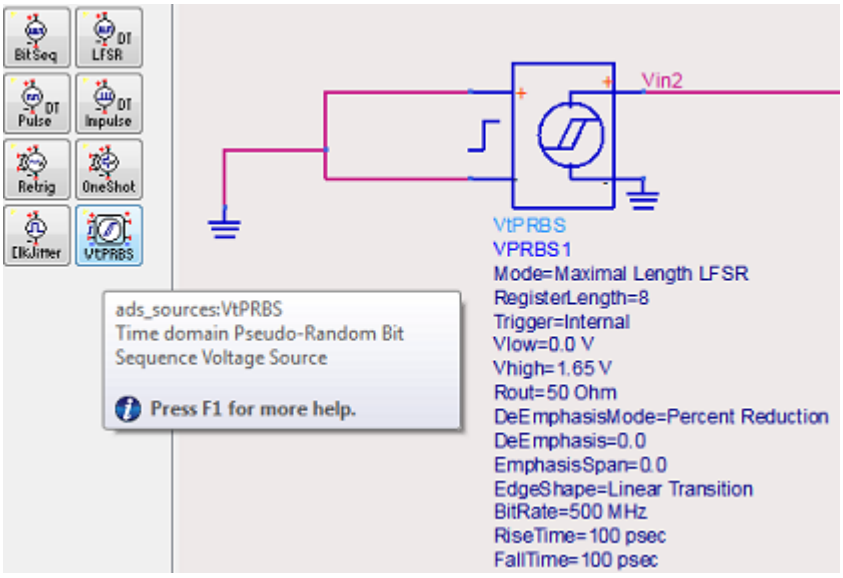



Figure 5.5: ADS PSRB Source

bit generator source. The source is located under *Sources-Time Domain* and it's called VtPRBS. The source is shown in Figure 5.5.

After replacing the source, edit the simulation runtime so it is long enough to generate a good eye. The resulting circuit is shown in Figure 5.6. To generate an eye diagram in ADS, we must write an equation using the form *eqn = eye (node_name, data_rate, number_of_cycles, delay)* [7].

As we can see, in Figure 5.7, the eye diagram is very similar to the one generated using HSpice.

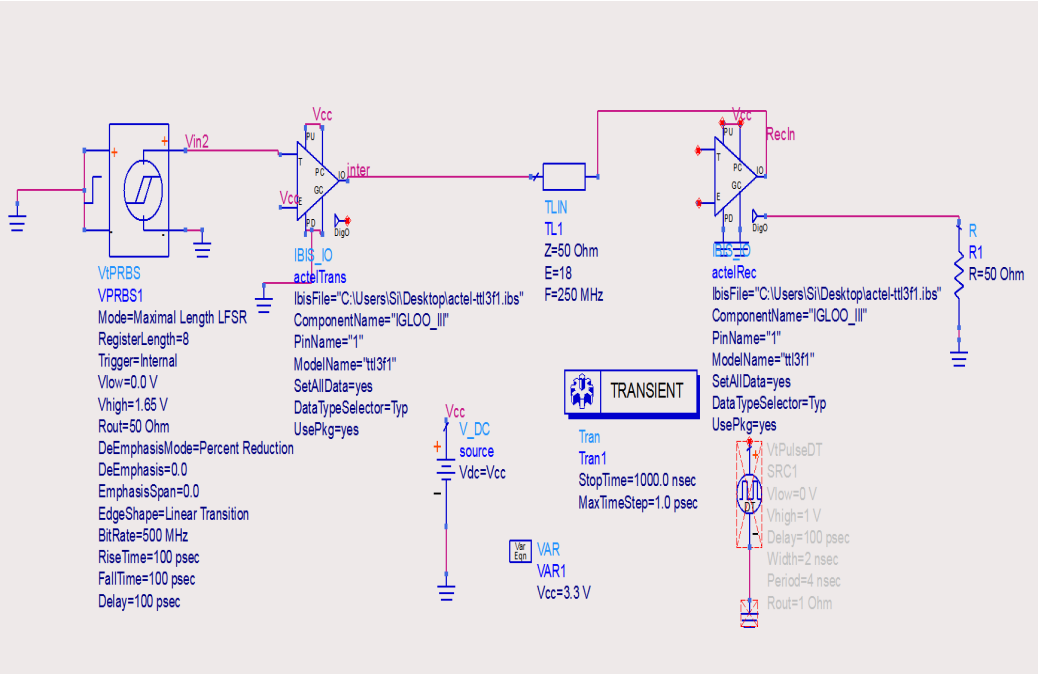



Figure 5.6: Eye Diagram using ADS

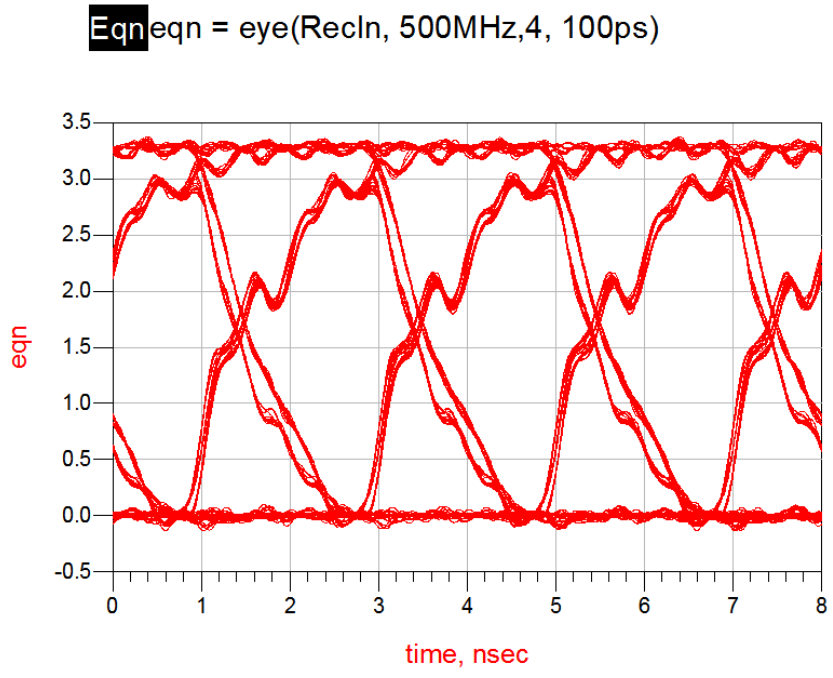



Figure 5.7: Eye Diagram using ADS

# CHAPTER 6

# CONCLUSION

In this thesis, simulation of IBIS models using HSpice and ADS was shown. A broad overview of IBIS models was presented, followed by examples of circuit simulations using IBIS models in both HSpice and ADS. This was then followed by an introduction to eye diagrams and instructions on how to generate them in both HSpice and ADS. There are some additional comments that should be made about simulating IBIS models in ADS and Hspice.

As we can see from the examples, there is no major difference between the two simulators with respect to results. However, there is a difference in runtime. It seems as though HSpice is better equipped than ADS in using IBIS models. This was apparent when we tried to simulate eye diagrams using ADS. Using the same settings, it took HSpice only a couple of seconds to simulate while it took ADS a minute or two to finish its simulation. This is only for a simple case. If we were to use a more complicated circuit, ADS would have a more difficult time than HSpice. However, they both produce similar results.

In summary, this thesis provided examples on how to simulate IBIS models in ADS and HSpice. Using the steps provided, any circuit that requires the use of IBIS models can be simulated. Although ADS is not as well equipped to simulate IBIS models as HSpice, both still produce similar results. However, simulation runtime is compromised using ADS.

# REFERENCES

[1] I/O Buffer Modeling, Intel Corporation, 2005, presentation downloaded from Intel website.

[2] The IBIS Open Forum, IBIS Modeling Cookbook for IBIS Version 4.0. [Online]. Available: http://www.eda.org/ibis/cookbook/cookbook-v4.pdf, September 15, 2005

[3] J.E. Schutt-Aine, "IO Models and Terminations," Class notes for ECE 598, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 2012.

[4] The IBIS Open Forum, IBIS (I/O Buffer Information Specification) Version 5.0, [Online]. Available: http://eda.org/pub/ibis/ver5.0/ver5_0.pdf, August 29, 2008

[5] Synopsis Technical Staff, *HSPICE Signal Integrity User Guide*, Synopsis, September 2005.

[6] *HSPICE Device Models Quick Reference Guide Version W-2005.03*, Synopsis, March 2005.

[6] *CosmosScope Reference Manual Version W-2004.12*, Synopsis, December 2004.

[7] *Agilent Eye Diagram Measurements in Advanced Design System*, Agilent Technologies, USA, May 26, 2006.