

Automated Timing Characterization for the Digital AMI 05 Library

Sanjeev Jindal
December 2005
ECE 793

1. Introduction	1
2. Cadence Simulation	2
Creating a symbol	2
Creating a Verilog-AMS file	2
Designing a Schematic.....	2
Running the Simulation	3
3. The Stimulus	4
NAND Gates.....	4
NOR Gates.....	4
Inverters and Buffers	5
XOR and XNOR Gates.....	5
Multiplexer	6
D-Type Flip-Flop.....	6
Latches.....	7
AND-OR Combination Gates	7
4. The Results Extractor	9
Basic Logic Gates	9
D Flip-Flops and Latches	9
Multiplexer	10
Tri-state Buffer and Tri-state Inverter	10
5. Conclusion	11
6. Appendix	12
Stimulus file for the NAND2x1 gate: NAND2x1stim.vams	12
Stimulus file for the NAND3x1 gate: NAND3x1stim.vams	13
Stimulus file for the NAND4x1 gate: NAND4x1stim.vams	14
Stimulus file for the NOR2x1 gate: nor2x1stim.vams	15
Stimulus file for the NOR3x1 gate: nor3x1stim.vams	16
Stimulus file for the NOR4x1 gate: nor4x1stim.vams	17
Stimulus file for the BUFx1, INVx1, BUFx4, and INVx4 gates: bufx1stim.vams	18
Stimulus file for the BUFZx1 and INVZx1 gates: bufzx1.vams	19
Stimulus file for the XOR2x1 and XNOR2x1 Gates: xor2x1stim.vams	20
Stimulus file for the MUX21x1 gate: mux21xstim.vams.....	21
Stimulus File for the LAT Gate: latstim.vams	22
Stimulus File for the LATPC gate: latpcstim.vams.....	23
Stimulus for the DFF gate: dffstim.vams	25

Stimulus File for the DFFPC gate: dffpcstim.vams.....	26
Stimulus File for AO22x1 and AOI22x1 gates: ao22x1stim.vams	28
Results Extractor for 1-input gates: oneinput.vams.....	29
Results Extractor for 2-input gates: twoinputre.vams	30
Results Extractor for 3-input gates: threeinputre.vams.....	32
Results Extractor for 4-input gates: fourinputre.vams.....	34

1. Introduction

The purpose of this project was to create a test bench that provides the timing characterization for all of the cells in the digital_lib_ami05 library designed by James Copus. The test benches should be able to provide the timing information automatically. The motivation for this project is to make reuse of the library easier. Instead of having to find the timing information by hand, the test benches would provide the necessary information automatically.

The test benches were designed using Cadence Virtuoso and they were written using Verilog-AMS. There are three main parts in designing these test benches: the device under test, the stimulus, and the results extractor. The device under test is the various cells in the digital_lib_ami05 library. The stimulus provides the various inputs for the cells, and the results extractor examines the inputs and outputs of the cell and calculates the timing information. The general diagram for this is shown in Figure 1.

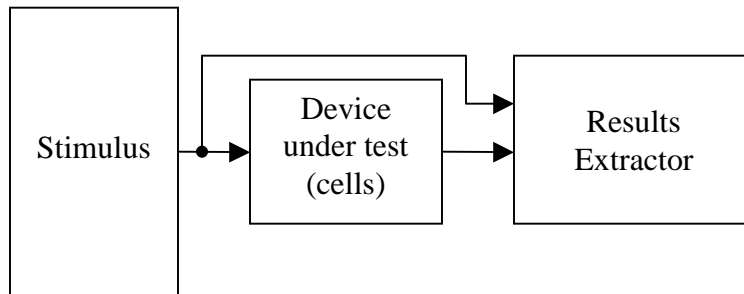


Figure 1: General Diagram for the Test Benches

2. Cadence Simulation

Cadence Virtuoso was used to design and simulate the test benches. There are four main steps in simulating these test benches: creating a symbol for both the stimulus and results extractor, creating Verilog-AMS files for the symbols, designing a schematic that instantiates the symbols made and the gate to be tested, and creating a configuration file that sets up and runs the simulation. However, first it is necessary to create a library in which all of these files are going to be saved. This is done by selecting File->New->Library in Library Manager.

Creating a symbol

When creating the symbol for either the stimulus block or the results extractor, a new cell view must be created. This is done by selecting File->New->CellView in Library Manager. Then, the window "Create new file" will pop up. Name the symbol by entering it in the space for "Cell Name" and select "Composer-Symbol" under "Tool." Once this is done, the first step in creating the actual symbol is to draw a rectangular box. This is done by selecting Add->Shape->Rectangle. The next step is to create I/O pins. This is done by selecting Add->Pin. Add the pin names and whether the pin will be an input or an output in the new pop-up window. Finally, check and save the symbol and exit.

Creating a Verilog-AMS file

When creating a Verilog-AMS file a new cell view needs to be created. This is done by selecting File->New->CellView in Library Manager. Then, a new window will pop up labeled "Create new file." Name the Verilog-AMS code the same as the symbol by entering it in the space for "Cell Name" and select "Verilog-AMS Editor" under "Tool." This should bring up an E-MACs window. Once the file has been edited, the file needs to be saved by selecting "Save Buffer" under the "File" menu. Once you close the E-MACs window, Cadence Virtuoso should bring up a window showing the errors if there are errors. The creations of the stimulus and result extractor files are explained in detail in Section 3 and Section 4.

Designing a Schematic

The first step is to create a new cell view. This is done by selecting File->New->CellView in Library Manager. Then, a new window will pop up labeled "Create new file." Name the schematic by entering it in the space for "Cell Name" and select "Composer-Schematic" under "Tool." In order to instantiate the cells from the digital_lib_ami05 library, it was first necessary to copy the files from that library into the current library. This is done by right clicking on the cell to be copied and selecting copy in Library Manager. Once this is done, the cell, the

stimulus block, and the results extractor blocks were instantiated in the design and the necessary connections were made. Instantiation of each block was done by selecting Add->Instance and then the needed part is found and added to the schematic. Then, the schematic was checked for errors and saved.

Running the Simulation

Once the schematic has been created, the next step is to simulate the schematic. However, before simulations are run it is necessary to add two lines to the cds.lib file. The two lines are:

```
SOFTINCLUDE $AMSHOME/tools/inca/files/cds.lib  
DEFINE basic $CDSHOME/tools/dfl/etc/cdslib/basic
```

For the simulations, Brian Dupaix's report for EE894z was followed. The steps for simulation are outlined below.

First, a hierarchal cell view has to be created. This is done by selecting File->Create New->Cellview in library manager and by selecting "Hierarchy-Editor" as the "Tool" when the "Create New File" window pops up. Once this is done, the "New Configuration" window will pop up. Select the view you will be using by clicking "Browse" button next to the current view and choose the schematic you just created. Also, click on "Use Template" and select "AMS."

Once this is done, the "Cadence Hierarchy Editor" window will pop-up. Select Plug-ins->AMS to be able to simulate AMS files. For the stimulus file and result extractor file make sure verilogams is chosen as the "View to Use." For the logic gates from the digital_lib_ami05 library, choose "Schematic" as the "View to Use." After this, it is necessary to update the view by selecting View->Update. Views for the transistors should show now. Choose spectre as the "View to Use" for the transistors. Again, update the view.

Then, select AMS->Run Directory and click run after choosing a directory. Next, set the stop time by selecting AMS->Option->Simulator. Find "Tran Analysis," and change "Stop time" to a non-zero value. Next, Design Prep needs to be run by selecting AMS->Design Prep. Once Design Prep is run, simulations can be performed if there are no errors. Next, choose AMS->Run Simulation. In the "AMS Run Simulation" window, select Analog Models Setup->Browse. Then, browse to:

```
/opt/local/cadence/design_kits/NCSU/ncsu_cdk.1.3VT/local/models/spectre/nom/allModels.scs
```

Then, click "Run." Cadence SimVision will start up after a few seconds. Once this is done, choose the signals to view on the left by clicking on them. Then, click on the waveform button on the top. In the "Waveform" window go to Simulation->Run to run the simulation.

3. The Stimulus

As mentioned before, the stimulus provides the input values for the cells. In order to test the cells for rise and fall times, it is necessary to provide inputs that will change the output of the cells every time that the input changes. The stimulus files are shown in the Appendix. The creation of the stimulus files for the cells are shown in detail in the sections below.

NAND Gates

In order to have the output of the NAND gate continuously change, it is necessary to hold all the inputs high in one state, and in the next state it is necessary to make one or more of the inputs low. This cycle must continue until all the values of the inputs have been tested. When running a digital simulation of this for the NAND2x1 gate, the values should basically follow the values shown in Table 1.

Table 1: Values for the NAND2x1 Gate

Input Value (A)	Input Value (B)	Output Value (Y)
1	1	0
0	0	1
1	1	0
0	1	1
1	1	0
1	0	1
1	1	0

In order to this using Verilog-AMS, I created two register values A and B and initialized the values listed in the table in the “initial” section of the stimulus file. For example, to run the first value, the statement “#100 a=1; b=1;” was used. This statement waits 100ns and then forces ‘a’ and ‘b’ to a value of 1.

Then, to convert this to an analog voltage, the “analog” section of the stimulus file was used. The statement, “V(vouta) <+ VoltageRail*a;” creates an analog voltage at the output of the stimulus block equal to the positive voltage rail when ‘a’ is a one, or an analog voltage of zero when ‘a’ is a zero. The stimulus files for all of the other gates follow the same procedure.

NOR Gates

The NOR gates stimulus files were very similar to the NAND gate stimulus file. However, in order to change the output of the NOR gate continuously, it is necessary to hold the inputs low for one state, and then hold one or more inputs high for the next state. The stimulus values for the NOR2x1 gate are shown below.

Table 2: Values for the NOR2x1 Gate

Input Value (A)	Input Value (B)	Output Value (Y)
0	0	1
0	1	0
0	0	1
1	0	0
0	0	1
1	1	0
0	0	1

Inverters and Buffers

The stimulus files for both the inverters and buffers were exactly the same. To change the output of the inverter or buffer the inputs need to be changed from low to high and then back to low.

However, in the digital_lib_ami05 library there is a tri-state buffer and a tri-state inverter. For this, it is necessary to test the condition for when the tri-state pin is asserted, and the input changes. Also, it is necessary to test the conditions from when the tri-state pin is not asserted and the output is at a high-impedance state to when the tri-state pin is asserted and the output is either high or low depending on the input. The values for the bufzx1 stimulus are shown in Table 3 below.

Table 3: Values for the BUFZX1 Gate

Input Value (A)	Input Value (Tri-state)	Output Value (Y)
0	0	0
1	0	1
0	0	0
X	1	High-Z
1	0	1
X	1	High-Z
0	0	0

XOR and XNOR Gates

The stimulus files for the XOR and the XNOR gates were exactly the same. For the 2-input XOR gate, the output is low when both inputs are equal, and is high otherwise. It is necessary to test the conditions where both the inputs are low against the two input values that make the output high. After this is done, it is necessary to test the conditions for when both the inputs are high against the

other two input values that make the output high. The stimulus values for the XOR gate are shown below.

Table 4: Values for the XOR2x1 Gate

Input (A)	Input (B)	Output (Y)
0	0	0
0	1	1
0	0	0
1	0	1
0	0	0
1	1	0
0	1	1
1	1	0
1	0	1
1	1	0

Multiplexer

The multiplexer uses the select input to select which input gets translated to the output. When the select line is low, input 'A' gets translated to the output, and when the select line is a high, input 'B' gets translated to the output. So, when making the stimulus file for the multiplexer it is necessary to make the select line high when testing the 'A' input, and low when testing the 'B' input. The values for the stimulus are shown in the table below.

Table 5: Values for the MUX2x1

Input (A)	Input (B)	Input (Sel)	Output (Y)
0	X	1	0
1	X	1	1
0	X	1	0
X	0	0	0
X	1	0	1
X	0	0	0

D-Type Flip-Flop

The flip-flop in the digital_lib_ami05 library is a positive edge triggered flip-flop. That means every time the clock goes from a low to a high state the input gets translated to the output. Like the latch, there are two types of D-type flip flops. One has just a clock and a 'D' input, and the other one has a clock, a 'D' input, a preset, and a clear. In order to generate the clock the statement "always #20 clk=~clk" was used. Three main timing values are need for the D flip-flop without the clear and preset lines. These timing values are the setup time, the hold time, and the propagation delay time. The setup time is the time 'D' needs to be set

before the rising edge of the clock, the hold time is the time 'D' must remain stable after the rising edge of the clock, and the propagation delay is the time that the output changes after the rising edge of the clock.

In the stimulus file, the setup time is first found by decreasing the time that 'D' changes before the rising edge of the clock. This will run 300 times to get an accuracy of 10ps. The hold time is then found by decreasing the time that 'D' changes after the rising edge of the clock. This will also run 300 times to get an accuracy of 10ps. After these two values are found, the propagation delay is found by seeing how long the output changes after the rising edge of the clock. When running this, it is needed to make sure the setup, hold, and propagation delay are tested independently of each other.

For the D-type flip-flop that has a preset input which sets the output and has a clear input which clears the output, the time the preset and clear must be asserted before the rising edge of the clock must be found. When testing the clear pin, 'D' was set high and the time that clear was asserted before the rising edge of the clock was decreased. Therefore, when the clear time is violated the output will be high. The preset time was done the same way except 'D' was set low.

Latches

Like the D flip-flop, the digital_lib_ami05 library contains one latch with a 'D' input pin and an enable pin and one latch with a 'D' input pin, an enable pin, a preset pin, and a clear pin. For this, the setup and hold times are based on the falling edge of the enable pin. The setup time is the time 'D' has to be set to the value wanted at the output before the enable is negated, and the hold time is time 'D' has to remain at its value after the enable is negated. The propagation delay, however, is the delay time from the rising edge of the enable pin. The clear and preset was also done like the D flip-flop except it was tested based on the falling edge of the enable pin.

AND-OR Combination Gates

The AO22x1 combination gate has two AND gates feeding an OR gate. The AOI22x1 combination gate is the inverted version of the AO22x1 gate. The stimulus files for both of these gates are exactly the same. To test this gate, the values of when one of the NAND output is asserted must be tested against all of the other values the cause the same NAND output to not be asserted. Then, the other NAND gate is tested the same way. Lastly, the value which cause both NAND outputs are asserted is tested against all the values that cause neither NAND outputs are asserted. This is shown in Table 6 below.

Table 6: Values for the AO22x1 Gate

Input (A)	Input (B)	Input (C)	Input (D)	Output (Y)
0	0	1	1	1
0	0	0	0	0
0	0	1	1	1
0	0	0	1	0
0	0	1	1	1
0	0	1	0	0
0	0	1	1	1
1	1	0	0	1
0	0	0	0	0
1	1	0	0	1
0	1	0	0	0
1	1	0	0	1
1	0	0	0	0
1	1	0	0	1
1	1	1	1	1
0	0	0	1	0
1	1	1	1	1
0	0	1	0	0
1	1	1	1	1
0	1	0	0	0
1	1	1	1	1
0	1	0	1	0
1	1	1	1	1
0	1	1	0	0
1	1	1	1	1
1	0	0	0	0
1	1	1	1	1
1	0	0	1	0
1	1	1	1	1
1	0	1	0	0
1	1	1	1	1

4. The Results Extractor

As mentioned before, the results extractor looks at the output for a change and then calculates either the rise time or the fall time compared to an input change. The results extractor should be independent of the stimulus file. This allows the user to change the stimulus file if needed without having to change the results extractor. The Verilog-AMS result extractor files are shown in the Appendix and they are explained in detail below. However, due to the lack of time, the results extractors for the D flip-flops, the Latches, the Multiplexer, the Tri-state Buffer, and the Tri-state Inverter were not completed.

Basic Logic Gates

The results extractor files were similar for the basic logic gates in the digital_lib_ami05 library. These gates include the NAND gates, the NOR gates, the XOR gates, the XNOR gates, the INV gates, and the BUF gates. The differences in the results extractor depend on the number of inputs of the gate. For these gates, four different results extractors were made. These were called: oneinputre, twoinputre, threeinputre, and fourinputre.

The rise and fall times were determined from the midpoint of the signal. This midpoint was inputted from the stimulus file. This midpoint equals $(vdd+vss)/2$ and was called vmid. To see if the signal crossed this midpoint the statement, "@(cross(V(signal,vmid))", was used. This statement executes every time the signal crossed the midpoint.

The results extractors wait until an output change. When the output changes, the results extractor looks to see what input have changed. When the input changes, the time it changed is saved. Also, when the output has changed, the time it changed is saved. Then, these two times are subtracted and outputted in the simulation window. To save the current time of change, "\$abstime" was used. To output the time, "\$strobe" was used.

D Flip-Flops and Latches

For the result extractor files for both of the D flip-flops, special care needs to be taken to be able to find the various timing information which includes the setup time, the hold time, and the propagation delay. When testing the setup time and the hold time, the outputs need to be looked at to see when the first time the output does not change. Then, the time when 'D' changes and the time the rising edge of the clock occurs need to be subtracted from each other. Timing information may need to be passed from the stimulus block to the results extractor for this to be possible. When testing the propagation delay, the time the output changes after the rising edge of the clock needs to be found. Also, there

are two outputs for the D flip-flop and they must be taken into account when testing the setup time, the hold time, and the propagation delay time. In addition, for the D flip-flop with the preset and clear, the preset and clear needs to be checked to see if they are negated when testing the 'D' input. The results extractors for the D flip-flop and the Latch should be the same.

Multiplexer

The multiplexer has two inputs 'A' and 'B' that will get translated to the output depending on the value of the select input. Therefore, when designing the results extractor, it is necessary to make sure select is asserted when testing the 'A' and negated when testing the 'B' input.

Tri-state Buffer and Tri-state Inverter

The tri-state buffer and the tri-state inverters should have the same results extractors. Like the multiplexer, it is necessary to make sure the tri-state pin is asserted when testing the input 'A.' However, when the tri-state pin is not asserted the output is at a High-Z level. The High-Z level can be lie anywhere between the voltage rails so special care needs to be taken when testing the tri-state buffer and inverter.

5. Conclusion

The test benches created during this project provides the timing information of most of the gates in the digital_lib_ami05 library designed by James Copus. This allows someone to be able to reuse the cell easier without having to find the timing information by hand.

Some problems encountered while running the simulations is that the D-flip flop and the Latch schematics from the digital_lib_ami05 library used an inverter symbol from the digital_lib_ami05. It was necessary to add the inverters copied to my library to the schematics for the D flip-flop and the Latch. However, when checking and saving the schematic there were some warnings, and when running the simulations, the D flip-flop and Latch did not work correctly.

In the future, the results extractors for both of the D flip-flops, both of the latches, the multiplexer, the tri-state buffer, and the tri-state inverter need to be designed and placed in the schematics already made. Also, the stimulus files for all of the gates could be better designed by using a loop to change the values of the inputs.

Also, the Environmental Control needs to be designed to automate the simulation process. This is a script that runs the tests automatically, saves the waveforms, and creates a log file for the simulation results for a given configuration.

6. Appendix

Stimulus file for the NAND2x1 gate: NAND2x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module nand2x1stim(vouta,voutb,vdd,vss,vmid);
output vouta,voutb,vdd,vss,vmid;
electrical vouta,voutb,vdd,vss,vmid;
reg a,b;
parameter real VoltageRail=3.3;

initial begin
a=1; b=1;

#100 a=1; b=1;
#100 a=0; b=0;

#100 a=1; b=1;
#100 a=0; b=1;

#100 a=1; b=1;
#100 a=1; b=0;

#100 a=1; b=1;

end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta,vss) <+ a*VoltageRail;
V(voutb,vss) <+ b*VoltageRail;
end

endmodule
```

Stimulus file for the NAND3x1 gate: NAND3x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module nand3x1stim(vouta,voutb,voutc,vdd,vss,vmid);
output vouta,voutb,voutc,vdd,vss,vmid;
electrical vouta,voutb,voutc,vdd,vss,vmid;
reg a,b,c;
parameter real VoltageRail=3.3;

initial begin
a=1; b=1; c=1;
#100 a=1; b=1; c=1;
#100 a=0; b=0; c=0;

#100 a=1; b=1; c=1;
#100 a=0; b=0; c=1;

#100 a=1; b=1; c=1;
#100 a=0; b=1; c=0;

#100 a=1; b=1; c=1;
#100 a=0; b=1; c=1;

#100 a=1; b=1; c=1;
#100 a=1; b=0; c=0;

#100 a=1; b=1; c=1;
#100 a=1; b=0; c=1;

#100 a=1; b=1; c=1;
#100 a=1; b=1; c=0;

#100 a=1; b=1; c=1;

end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
V(voutc) <+ c*VoltageRail;
end

endmodule
```

Stimulus file for the NAND4x1 gate: NAND4x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module nand4x1stim(vouta,voutb,voutc,voutd,vdd,vss,vmid);
output vouta,voutb,voutc,voutd,vdd,vss,vmid;
electrical vouta,voutb,voutc,voutd,vdd,vss,vmid;
reg a,b,c,d;
parameter real VoltageRail=3.3;
initial begin
a=1;b=1;c=1;d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=0; c=0; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=0; c=1; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=0; c=1; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=1; c=0; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=1; c=0; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=1; c=1; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=1; c=1; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=0; c=0; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=0; c=0; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=0; c=1; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=0; c=1; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=1; c=0; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=1; c=0; d=1;
#100 a=1; b=1; c=1; d=1;
#100 a=1; b=1; c=1; d=0;
#100 a=1; b=1; c=1; d=1;
end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
V(voutc) <+ c*VoltageRail;
V(voutd) <+ d*VoltageRail;
end

endmodule
```

Stimulus file for the NOR2x1 gate: nor2x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module nor2x1stim(vouta,voutb,vdd,vss,vmid);
output vouta,voutb,vdd,vss,vmid;
electrical vouta,voutb,vdd,vss,vmid;
reg a,b;
parameter real VoltageRail=3.3;

initial begin
a=0; b=0;

#100 a=0; b=0;
#100 a=0; b=1;

#100 a=0; b=0;
#100 a=1; b=0;

#100 a=0; b=0;
#100 a=1; b=1;
#100 a=0; b=0;

end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
end

endmodule
```

Stimulus file for the NOR3x1 gate: nor3x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module nor3x1stim(vouta,voutb,voutc,vdd,vss,vmid);
output vouta,voutb,voutc,vdd,vss,vmid;
electrical vouta,voutb,voutc,vdd,vss,vmid;
reg a,b,c;
parameter real VoltageRail=3.3;

initial begin
a=0; b=0; c=0;
#100 a=0; b=0; c=0;
#100 a=0; b=0; c=1;

#100 a=0; b=0; c=0;
#100 a=0; b=1; c=0;

#100 a=0; b=0; c=0;
#100 a=0; b=1; c=1;

#100 a=0; b=0; c=0;
#100 a=1; b=0; c=0;

#100 a=0; b=0; c=0;
#100 a=1; b=0; c=1;

#100 a=0; b=0; c=0;
#100 a=1; b=1; c=0;

#100 a=0; b=0; c=0;
#100 a=1; b=1; c=1;

#100 a=0; b=0; c=0;

end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
V(voutc,vss) <+ c*VoltageRail;
end

endmodule
```

Stimulus file for the NOR4x1 gate: nor4x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module nor4x1stim(vouta,voutb,voutc,voutd,vdd,vss,vmid);
output vouta,voutb,voutc,voutd,vdd,vss,vmid;
electrical vouta,voutb,voutc,voutd,vdd,vss,vmid;
reg a,b,c,d;
parameter real VoltageRail=3.3;

initial begin
a=0;b=0;c=0;d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=0; c=0; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=0; c=1; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=0; c=1; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=1; c=0; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=1; c=0; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=1; c=1; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=0; b=1; c=1; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=0; c=0; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=0; c=0; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=0; c=1; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=0; c=1; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=1; c=0; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=1; c=0; d=1;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=1; c=1; d=0;
#100 a=0; b=0; c=0; d=0;
#100 a=1; b=1; c=1; d=1;
#100 a=0; b=0; c=0; d=0;

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
V(voutc) <+ c*VoltageRail;
V(voutd) <+ d*VoltageRail;
end
endmodule
```

**Stimulus file for the BUFx1, INVx1, BUFx4, and INVx4 gates:
bufx1stim.vams**

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module bufx1stim(vouta,vdd,vss,vmid);
output vouta,vdd,vss,vmid;
electrical vouta,vdd,vss,vmid;
reg a;
parameter real VoltageRail=3.3;

initial begin
a=0;
#100 a=0;
#100 a=1;
#100 a=0;

end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta,vss) <+ VoltageRail*a;
end

endmodule
```

Stimulus file for the BUFZx1 and INVZx1 gates: bufzx1.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module bufzx1stim(vouta,vdd,voutts,vss,vmid);
output vouta,vdd,vss,voutts,vmid;
electrical vouta,vdd,vss,vmid,voutts;
reg a,ts;
parameter real VoltageRail=3.3;

initial begin
a=0; ts=0;

#100 a=0; ts=0;
#100 a=1; ts=0;
#100 a=0; ts=0;

#100 ts=1;
#100 a=1; ts=0;
#100 ts=1;
#100 a=0; ts=0;
end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutts) <+ ts*VoltageRail;
end

endmodule
```

Stimulus file for the XOR2x1 and XNOR2x1 Gates: xor2x1stim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module xor2x1stim(vouta,voutb,vdd,vss,vmid);
output vouta,voutb,vdd,vmid;
electrical vouta,voutb,vdd,vss,vmid;
reg a,b;
parameter real VoltageRail=3.3;

initial begin

a=0; b=0;
#100 a=0; b=0;
#100 a=0; b=1;

#100 a=0; b=0;
#100 a=1; b=0;
#100 a=0; b=0;

#100 a=1; b=1;
#100 a=0; b=1;

#100 a=1; b=1;
#100 a=1; b=0;
#100 a=1 ; b=1;
end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
end

endmodule
```

Stimulus file for the MUX21x1 gate: mux21xstim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module mux21x1stim(voutsel,vdd,vouta,voutb,vss,vmid);
output voutsel,vdd,vss,vouta,voutb,vmid;
electrical voutsel,vdd,vss,vmid,voutb,vouta;
reg sel,a,b;
parameter real VoltageRail=3.3;

initial begin
a=0; b=0; sel=1;
#100 sel=1; a=0;
#100 sel=1; a=1;
#100 sel=1; a=0;
#100 sel=0; b=0;
#100 sel=0; b=1;
#100 sel=0; b=0;
end

analog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(vouta) <+ a*VoltageRail;
V(voutb) <+ b*VoltageRail;
V(voutsel) <+sel*VoltageRail;
end

endmodule
```

Stimulus File for the LAT Gate: latstim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module latstim(voutd,vdd,vouten,vss,vmid);
output voutd,vdd,vss,vouten,vmid;
electrical voutd,vdd,vss,vmid,vouten;
reg vout,clk;
parameter real VoltageRail=3.3;
real risedelay;
real falldelay;

initial begin
clk=1;
vout=1;
risedelay=17;
falldelay=3;
end

always #20 clk=~clk;

always @(posedge clk) begin
if (falldelay==3) begin
if (risedelay<20) begin
#risedelay vout=~vout;
risedelay=risedelay+0.01;
end
end
end

always @(negedge clk) begin
if (risedelay==20 && falldelay>0) begin
#falldelay vout=~vout;
falldelay=falldelay-0.01;
end
end

always @(negedge clk) begin
if (risedelay==20 && falldelay==0) begin
#10 vout=~vout;
end
end

endanalog begin
V(vdd,vmid) <+ VoltageRail/2;
V(vmid,vss) <+ VoltageRail/2;
V(voutd) <+ VoltageRail*vout;
V(vouten) <+ VoltageRail*clk;
end

endmodule
```

Stimulus File for the LATPC gate: latpcstim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module latpcstim(voutd,ven,vclr,vpre,vdd,vss,vmid);

output voutd,ven,vdd,vss,vclr,vpre,vmid;
electrical voutd,ven,vdd,vss,vclr,vpre,vmid;

parameter real VoltageRail=3.3;
real risedelay,falldelay, prerise,clrrise;
reg clk, vout;

initial begin
    pre=0;
    clr=0;
    clk=1;
    vout=0;
    risedelay=17;
    falldelay=3;
    prerise=17;
    clrrise=17;
end

always #20 clk=~clk;

always @(posedge clk) begin
    if (prerise<20 && clrrise==17) begin
        #clrrise vout=1; clr=1;
        #25 clr=0;
        clrrise=clrrise+0.01;
    end
    if (prerise==20 && clrrise<20) begin
        #prerise vout=0; pre=1;
        #25 pre=0;
    end
end

always @(posedge clk) begin
    if (falldelay=3 && prerise==20 && clrrise==20) begin
        if (risedelay<20) begin
            #risedelay vout=~vout;
            risedelay=risedelay+0.01;
        end
    end
end

always @(negedge clk) begin
    if (risedelay==20 && prerise==20 && clrrise==20) begin
        if (falldelay>0) begin
            #falldelay vout=~vout;
            falldelay=falldelay-0.01;
        end
    end
end
```

```
        end
    end

    end

always @(negedge clk) begin
    if (risedelay==20 && falldelay==0) begin
        #10 vout=~vout;
    end

analog begin
    V(vdd,vmid) <+ VoltageRail/2;
    V(vmid,vss) <+ VoltageRail/2;
    V(voutd) <+ vout*VoltageRail;
    V(ven) <+ clk*VoltageRail;
    V(vclr) <+ clr*VoltageRail;
    V(vpre) <+ pre*VoltageRail;
end

endmodule
```

Stimulus for the DFF gate: dffstim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module dffstim(voutd,clock,vdd,vss,vmid);

output voutd,clock,vdd,vss,vmid;
electrical voutd,vdd,vss,vmid,clock;

parameter real VoltageRail=3.3;
real risedelay,falldelay;
reg clk, vout;

initial begin
    clk=1;
    vout=0;
    risedelay=17;
    falldelay=3;
end

always #20 clk=~clk;

always @(negedge clk) begin
    if (falldelay==3) begin
        if (risedelay<20) begin
            #risedelay vout=~vout;
            risedelay=risedelay+0.01;
        end

        if (risedelay==20) begin
            #10 vout=~vout;
        end
    end
end

always @(posedge clk) begin
    if (risedelay==20 && falldelay>0) begin
        #falldelay vout=~vout;
        falldelay=falldelay-0.01;
    end
end

analog begin
    V(vdd,vmid) <+ VoltageRail/2;
    V(vmid,vss) <+ VoltageRail/2;
    V(clock) <+ clk*VoltageRail;
    V(voutd) <+ vout*VoltageRail;
end

endmodule
```

Stimulus File for the DFFPC gate: dffpcstim.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module dffpcstim(voutd,clock,vclr,vpre,vdd,vss,vmid);
output voutd,clock,vdd,vss,vclr,vpre,vmid;
electrical voutd,vdd,vss,vclr,vpre,vmid,clock;
parameter real VoltageRail=3.3;
real risedelay,falldelay, prerise,clrrise;
reg clk, vout,pre,clr;

initial begin
    pre=0;
    clr=0;
    clk=1;
    vout=0;
    risedelay=17;
    falldelay=3;
    prerise=17;
    clrrise=17;
end

always #20 clk=~clk;

always @(negedge clk) begin
    if (prerise<20 && clrrise==17) begin
        #clrrise vout=1; clr=1;
        #25 clr=0;
        clrrise=clrrise+.01;
    end
    if (prerise==20 && clrrise<20) begin
        #prerise vout=0; pre=1;
        #25 pre=0;
    end
end

always @(negedge clk) begin
    if (falldelay==3 && prerise==20 && clrrise==20) begin
        if (risedelay<20) begin
            #risedelay vout=~vout;
            risedelay=risedelay+0.01;
        end

        if (risedelay==20) begin
            #10 vout=~vout;
        end
    end
end

always @(posedge clk) begin
    if (risedelay==20 && prerise==20 && clrrise==20) begin
        if (falldelay>0) begin
            #falldelay vout=~vout;
            falldelay=falldelay-0.01;
        end
    end
end

analog begin
    V(vdd,vmid) <+ VoltageRail/2;
```

```
V(vmid,vss) <+ VoltageRail/2;  
V(clock) <+ clk*VoltageRail;  
V(voutd) <+ vout*VoltageRail;  
V(vclr) <+ clr*VoltageRail;  
V(vpre) <+ pre*VoltageRail;  
end
```

```
endmodule
```


Results Extractor for 1-input gates: oneinput.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module oneinputre(vmid,vina,vouty);
input vmid,vina,vouty;
electrical vmid,vina,vouty;

real startttimea,startttimey,risettimea,falltimea, cs_a, ns_a, ns_y, cs_y;

analog begin

@(initial_step) begin
    startttimea=0;
    startttimey=0;
    cs_a=V(vina);
    cs_y=V(vouty);
    ns_a=V(vina);
    ns_y=V(vouty);
end

@(cross(V(vina,vmid))) begin
    startttimea=$abstime;
    risettimea=0;
    falltimea=0;
    ns_a=V(vina);
end

@(cross(V(vouty,vmid))) begin
    startttimey=$abstime;
    ns_y=V(vouty);

    if (ns_a > cs_a && ns_y > cs_y) begin
        risettimea=startttimey-startttimea;
        $strobe("Y rise time to A rise time is %f ns", risettimea/ln);
    end
    if (ns_a > cs_a && ns_y < cs_y) begin
        risettimea=startttimey-startttimea;
        $strobe("Y fall time to A rise time is %f ns", risettimea/ln);
    end
    if (ns_a < cs_a && ns_y > cs_y) begin
        falltimea=startttimey-startttimea;
        $strobe("Y rise time to A fall time is %f ns", falltimea/ln);
    end
    if (ns_a < cs_a && ns_y < cs_y) begin
        falltimea=startttimey-startttimea;
        $strobe("Y fall time to A fall time is %f ns", falltimea/ln);
    end
    cs_a=V(vina);
    cs_y=V(vouty);
end
end

endmodule
```

Results Extractor for 2-input gates: twoinputre.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module twoinputre(vmid,vina,vinb,vouty);
input vmid,vina,vinb,vouty;
electrical vmid,vina,vinb,vouty;

real starttimea,starttimeb,starttimey,risetimea,risetimeb;
real falltimea,falltimeb, cs_a, ns_a,cs_b,ns_b, ns_y, cs_y;

analog begin

@(initial_step) begin
    starttimea=0;
    starttimeb=0;
    starttimey=0;
    cs_a=V(vina);
    cs_b=V(vinb);
    cs_y=V(vouty);
    ns_a=V(vina);
    ns_b=V(vinb);
    ns_y=V(vouty);
end

@(cross(V(vina,vmid))) begin
    starttimea=$abstime;
    risetimea=0;
    falltimea=0;
    ns_a=V(vina);
end

@(cross(V(vinb,vmid))) begin
    starttimeb=$abstime;
    risetimeb=0;
    falltimeb=0;
    ns_b=V(vinb);
end

@(cross(V(vouty,vmid))) begin
    starttimey=$abstime;
    ns_y=V(vouty);

    if (ns_a > cs_a && ns_y > cs_y) begin
        risetimea=starttimey-starttimea;
        $strobe("Y rise time to A rise time is %f ns", risetimea/1n);
    end
    if (ns_a > cs_a && ns_y < cs_y) begin
        risetimea=starttimey-starttimea;
        $strobe("Y fall time to A rise time is %f ns", risetimea/1n);
    end
    if (ns_a < cs_a && ns_y > cs_y) begin
        falltimea=starttimey-starttimea;
        $strobe("Y rise time to A fall time is %f ns", falltimea/1n);
    end
    if (ns_a < cs_a && ns_y < cs_y) begin
        falltimea=starttimey-starttimea;
        $strobe("Y fall time to A fall time is %f ns", falltimea/1n);
    end

    if (ns_b > cs_b && ns_y > cs_y) begin
        risetimeb=starttimey-starttimeb;
        $strobe("Y rise time to B rise time is %f ns", risetimeb/1n);
    end
    if (ns_b > cs_b && ns_y < cs_y) begin
        risetimeb=starttimey-starttimeb;
        $strobe("Y fall time to B rise time is %f ns", risetimeb/1n);
    end
end
end
```

```
        end
    if (ns_b < cs_b && ns_y > cs_y) begin
        falltimeb=starttimey-starttimeb;
        $strobe("Y rise time to B fall time is %f ns", falltimeb/1n);
    end
    if (ns_b < cs_b && ns_y < cs_y) begin
        falltimeb=starttimey-starttimeb;
        $strobe("Y fall time to B fall time is %f ns", falltimeb/1n);
    end
    cs_a=V(vina);
    cs_b=V(vinb);
    cs_y=V(vouty);
end
end

endmodule
```

Results Extractor for 3-input gates: threeinputre.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module threeinputre(vmid,vina,vinb,vinc,vouty);
input vmid,vina,vinb,vinc,vouty;
electrical vmid,vina,vinb,vinc,vouty;

real starttimea,starttimeb,starttimec,starttimey,risetimea,risetimeb;
real risetimec,falltimea,falltimeb,falltimec;
real cs_a, ns_a,cs_b,ns_b,cs_c,ns_c, ns_y, cs_y;

analog begin

@(initial_step) begin
    starttimea=0;
    starttimeb=0;
    starttimec=0;
    starttimey=0;
    cs_a=V(vina);
    cs_b=V(vinb);
    cs_c=V(vinc);
    cs_y=V(vouty);
    ns_a=V(vina);
    ns_b=V(vinb);
    ns_c=V(vinc);
    ns_y=V(vouty);
end

@(cross(V(vina,vmid))) begin
    starttimea=$abstime;
    risetimea=0;
    falltimea=0;
    ns_a=V(vina);
end

@(cross(V(vinb,vmid))) begin
    starttimeb=$abstime;
    risetimeb=0;
    falltimeb=0;
    ns_b=V(vinb);
end

@(cross(V(vinc,vmid))) begin
    starttimec=$abstime;
    risetimec=0;
    falltimec=0;
    ns_c=V(vinc);
end

@(cross(V(vouty,vmid))) begin
    starttimey=$abstime;
    ns_y=V(vouty);

    if (ns_a > cs_a && ns_y > cs_y) begin
        risetimea=starttimey-starttimea;
        $strobe("Y rise time to A rise time is %f ns", risetimea/1n);
    end
    if (ns_a > cs_a && ns_y < cs_y) begin
        risetimea=starttimey-starttimea;
        $strobe("Y fall time to A rise time is %f ns", risetimea/1n);
    end
    if (ns_a < cs_a && ns_y > cs_y) begin
        falltimea=starttimey-starttimea;
        $strobe("Y rise time to A fall time is %f ns", falltimea/1n);
    end
end
```

```

if (ns_a < cs_a && ns_y < cs_y) begin
    falltimea=starttimey-starttimea;
    $strobe("Y fall time to A fall time is %f ns", falltimea/1n);
end

if (ns_b > cs_b && ns_y > cs_y) begin
    risetimeb=starttimey-starttimeb;
    $strobe("Y rise time to B rise time is %f ns", risetimeb/1n);
end
if (ns_b > cs_b && ns_y < cs_y) begin
    risetimeb=starttimey-starttimeb;
    $strobe("Y fall time to B rise time is %f ns", risetimeb/1n);
end
if (ns_b < cs_b && ns_y > cs_y) begin
    falltimeb=starttimey-starttimeb;
    $strobe("Y rise time to B fall time is %f ns", falltimeb/1n);
end
if (ns_b < cs_b && ns_y < cs_y) begin
    falltimeb=starttimey-starttimeb;
    $strobe("Y fall time to B fall time is %f ns", falltimeb/1n);
end

if (ns_c > cs_c && ns_y > cs_y) begin
    risetimec=starttimey-starttimec;
    $strobe("Y rise time to C rise time is %f ns", risetimec/1n);
end
if (ns_c > cs_c && ns_y < cs_y) begin
    risetimec=starttimey-starttimec;
    $strobe("Y fall time to C rise time is %f ns", risetimec/1n);
end
if (ns_c < cs_c && ns_y > cs_y) begin
    falltimec=starttimey-starttimec;
    $strobe("Y rise time to C fall time is %f ns", falltimec/1n);
end
if (ns_c < cs_c && ns_y < cs_y) begin
    falltimec=starttimey-starttimec;
    $strobe("Y fall time to C fall time is %f ns", falltimec/1n);
end

cs_a=V(vina);
cs_b=V(vinb);
cs_c=V(vinc);
cs_y=V(vouty);

end
end

endmodule

```

Results Extractor for 4-input gates: fourinputre.vams

```
`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns/1ns

module fourinputre(vmid,vina,vinb,vinc,vind,vouty);
input vmid,vina,vinb,vinc,vind,vouty;
electrical vmid,vina,vinb,vinc,vind,vouty;

real starttimea,starttimeb,starttimec,starttime_d,starttimey;
real risetimea,risetimeb,risetimec,risetime_d;
real falltimea,falltimeb,falltimec,falltime_d;
real cs_a, ns_a,cs_b,ns_b,cs_c,ns_c, cs_d, ns_d, ns_y, cs_y;

analog begin

@(initial_step) begin
    starttimea=0;
    starttimeb=0;
    starttimec=0;
    starttime_d=0;
    starttimey=0;
    cs_a=V(vina);
    cs_b=V(vinb);
    cs_c=V(vinc);
    cs_d=V(vind);
    cs_y=V(vouty);
    ns_a=V(vina);
    ns_b=V(vinb);
    ns_c=V(vinc);
    ns_d=V(vind);
    ns_y=V(vouty);
end

@(cross(V(vina,vmid))) begin
    starttimea=$abstime;
    risetimea=0;
    falltimea=0;
    ns_a=V(vina);
end

@(cross(V(vinb,vmid))) begin
    starttimeb=$abstime;
    risetimeb=0;
    falltimeb=0;
    ns_b=V(vinb);
end

@(cross(V(vinc,vmid))) begin
    starttimec=$abstime;
    risetimec=0;
    falltimec=0;
    ns_c=V(vinc);
end

@(cross(V(vind,vmid))) begin
    starttime_d=$abstime;
    risetime_d=0;
    falltime_d=0;
    ns_d=V(vind);
end

@(cross(V(vouty,vmid))) begin
    starttimey=$abstime;
    ns_y=V(vouty);

    if (ns_a > cs_a && ns_y > cs_y) begin
        risetimea=starttimey-starttimea;
        $strobe("Y rise time to A rise time is %f ns", risetimea/1n);
    end

    if (ns_a > cs_a && ns_y < cs_y) begin
        risetimea=starttimey-starttimea;
    end
end
end
```

```

        $strobe("Y fall time to A rise time is %f ns", risetimea/ln);
    end
    if (ns_a < cs_a && ns_y > cs_y) begin
        falltimea=starttimey-starttimea;
        $strobe("Y rise time to A fall time is %f ns", falltimea/ln);
    end
    if (ns_a < cs_a && ns_y < cs_y) begin
        falltimea=starttimey-starttimea;
        $strobe("Y fall time to A fall time is %f ns", falltimea/ln);
    end

    if (ns_b > cs_b && ns_y > cs_y) begin
        risetimeb=starttimey-starttimeb;
        $strobe("Y rise time to B rise time is %f ns", risetimeb/ln);
    end
    if (ns_b > cs_b && ns_y < cs_y) begin
        risetimeb=starttimey-starttimeb;
        $strobe("Y fall time to B rise time is %f ns", risetimeb/ln);
    end
    if (ns_b < cs_b && ns_y > cs_y) begin
        falltimeb=starttimey-starttimeb;
        $strobe("Y rise time to B fall time is %f ns", falltimeb/ln);
    end
    if (ns_b < cs_b && ns_y < cs_y) begin
        falltimeb=starttimey-starttimeb;
        $strobe("Y fall time to B fall time is %f ns", falltimeb/ln);
    end

    if (ns_c > cs_c && ns_y > cs_y) begin
        risetimec=starttimey-starttimec;
        $strobe("Y rise time to C rise time is %f ns", risetimec/ln);
    end
    if (ns_c > cs_c && ns_y < cs_y) begin
        risetimec=starttimey-starttimec;
        $strobe("Y fall time to C rise time is %f ns", risetimec/ln);
    end
    if (ns_c < cs_c && ns_y > cs_y) begin
        falltimec=starttimey-starttimec;
        $strobe("Y rise time to C fall time is %f ns", falltimec/ln);
    end
    if (ns_c < cs_c && ns_y < cs_y) begin
        falltimec=starttimey-starttimec;
        $strobe("Y fall time to C fall time is %f ns", falltimec/ln);
    end

    if (ns_d > cs_d && ns_y > cs_y) begin
        risetimed=starttimey-starttimed;
        $strobe("Y rise time to D rise time is %f ns", risetimed/ln);
    end
    if (ns_d > cs_d && ns_y < cs_y) begin
        risetimed=starttimey-starttimed;
        $strobe("Y fall time to D rise time is %f ns", risetimed/ln);
    end
    if (ns_d < cs_d && ns_y > cs_y) begin
        falltimed=starttimey-starttimed;
        $strobe("Y rise time to D fall time is %f ns", falltimed/ln);
    end
    if (ns_d < cs_d && ns_y < cs_y) begin
        falltimed=starttimey-starttimed;
        $strobe("Y fall time to D fall time is %f ns", falltimed/ln);
    end
    cs_a=V(vina);
    cs_b=V(vinb);
    cs_c=V(vinc);
    cs_d=V(vind);
    cs_y=V(vouty);
end
end

endmodule

```

7. References

- [1] Jason Abele. *Characterization and Capture of Design Intent for Mixed-Signal Circuits and Systems*. Master's thesis, The Ohio State University, 2005.
- [2] James Robert Copus. *Open Digital HDL to Synthesized Layout Flow for Mixed IC's*. Master's thesis. The Ohio State University, 2003.
- [3] Brian Dupaix. *Report for EE894z*. The Ohio State University, 2004.