SECTION OF ELECTRICAL ENGINEERING

EPFL STI – SEL
ELG
Station nº 11
CH-1015 Lausanne

Téléphone : +4121 693 13 46
Fax :
E-mail : alexandre.levisse@epfl.ch
Site web : https://sti.epfl.ch/fr/sel/

EE429 2024/2024: Full custom labs 1 : Testbenches, Simulation and Corners

SEL September 2025

PRACTICAL LABORATORY SESSION No. 2
Testbenches, Simulations and Corners

1 OBJECTIVES

In this lab session you will perform circuit simulations using a tool called ADE Assembler (for Analog Design Environment). ADE is a part of the Cadence Virtuoso tool suite. You will first create a testbench, run simulations, and then learn how to extract results from these simulations. Finally, you will learn how to run simulations which take into account CMOS variability (also called Monte-Carlo simulations).

> *In this project, you will use a spice simulator called **Spectre** and developed by **Cadence**. There are many spice simulators in the IC design market (the two most known ones being **Hspice** from **Synopsys** and **Eldo** from **Mentor-Graphics/Siemens**). They are all based on the spice syntax with some variations.*

First, you will design your own test-bench for an 8-bit 4-to-1 MUX which was designed in the previous session. Then you will determine transistor sizes and, with the insight provided by simulation results, verify and optimize the design.

Finally, you will practice your skills with a little exercise. As always there will be quiz questions on the content of the lab.

There are QUESTIONS along the lab, these are NOT GRADED, but will be useful for the quiz.

PREREQUISITES

- ✓ Start the Virtuoso Design Environment ***under your existing project directory*** – **EE429_FULLCUSTOM** (as described in laboratory session no.1, section 3.2).
- ✓ If the Library Manager does not appear automatically, activate it by selecting ***Tools→ Library Manager,*** from the **CIW** window.
- ✓ If a window appears asking for the available licences, simply click **Yes** and proceed.
- ✓ Use Library Manager and make sure that when creating cell views, the view type is set to **Schematic**. The default view name should be **schematic**. Also note that some windows may appear below other windows.

2 CREATING THE MUX TEST-BENCH SCHEMATIC

- ✓ Create a new schematic view in your library (EDATP) named: **MUX_4_1_8bit_TB**
- ✓ Instantiate the symbol of the **MUX_4_1_8bit** that you have already generated in the previous session into the new schematic editor window. Then complete the schematic as shown in Figure 1.

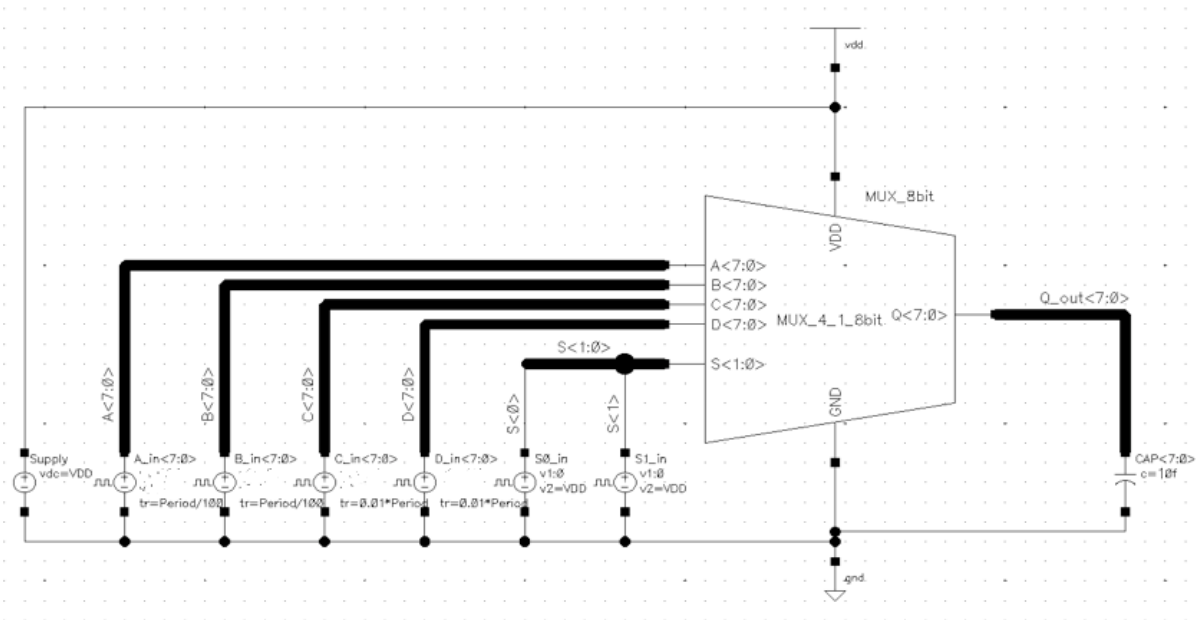


Figure 1 - Schematic View of the Testbench for the 8-bit MUX.

- ✓ For the ***DC source*** use the independent source **vdc** component from **analogLib** library. Name the instance as ***Supply***. Enter the properties (***q***) or use the ***Property Editor***. In the ***DC voltage*** field type: ***VDD***.
- ✓ For ***pulse generators*** (***A_in<7:0>, B_in<7:0>, C_in<7:0> and D_in<7:0>, S0_in*** and ***S1_in***, as indicated in Figure 1) use **vpulse** component from **analogLib**. Check that names of the instances correspond to Figure 1, and that the properties correspond to the Table I.
- ✓ For the load capacitors shown in the schematic, use the **cap** element from **analogLib**. Name the instance ***CAP<7:0>,*** and set the capacitance value to 10fF (10f).
- ✓ For the ***Vdd*** and ***ground*** symbols, use the global sources **vdd** and **gnd** cell symbol views from **analogLib** library. Ground symbol will provide the zero reference voltage level for your simulation.
- ✓ Make sure you are using buses for 8-bit signals and wires for single-bit signals.
- ✓ Use ***Add→ Wire Name*** or press ***"l"*** to place labels accordingly to the input/output signals as shown in Figure 1.

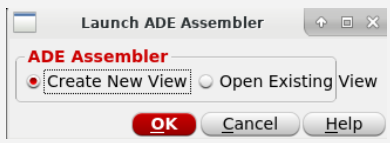
- *Note from the Table I, that **VDD, Vsweep** and **Period** are defined as variables. Numeric values can be set directly here, as we did for **Voltage 1** of the pulse generators which is always equal to zero. We will set the actual numeric values of these variables later, when we launch the circuit simulations.*
- *Variable **Vsweep** will be used to gradually vary the voltage of the input S1, and observe the corresponding output voltage of the inverter within the MUX. This type of simulation is usually called **voltage sweep**.*

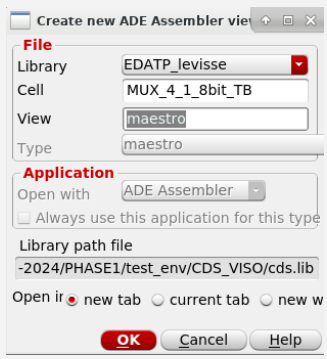
✓ **Check and Save** your design.

Parameter	Input A	Input B	Input C	Input D	Input S0	Input S1
DC voltage	0.0 V	0.0 V	0.0 V	0.0 V	0.0 V	Vsweep V
Voltage 1	0.0 V	0.0 V	0.0 V	0.0 V	0.0 V	0.0 V
Voltage 2	VDD V	VDD V	VDD V	VDD V	VDD V	VDD V
Period	Period s	2*Period s	4*Period s	8*Period s	16*Period s	32*Period s
Rise Time	0.01*Period s	0.01*Period s	0.01*Period s	0.01*Period s	0.01*Period s	0.01*Period s
Fall Time	0.01*Period s	0.01*Period s	0.01*Period s	0.01*Period s	0.01*Period s	0.01*Period s

Table I - Parameters for the pulse generators.

3 Creating Different Tests and Global Variables





✓ Open the simulation environment by clicking on ***Launch→ ADE Assembler*** on the very left upper corner of the schematic editor window. A small window will pop-up. Choose ***Create New View***, and press OK. If ***Create new ADE Assembler view*** window pops-up, press OK. Verify that the view being created is of type "maestro".

✓ The tool may inform you that it will get some license instead of another. This is normal, as Cadence is regularly rebranding their licenses. Just click "Always".

✓ The Virtuoso Analog Design Environment Window will appear, as shown in Figure 2.

✓ Note from Figure 2, that a tab is added for your schematic. This new tab is called Maestro. This is the tab from which you will be able to define and setup your outputs and simulation results.

✓ ***Run Summary*** sub-window will allow you to see the overview of the tests that you have run and the status of the current and previous simulations.

✓ Explore the ***Data View*** sub-window. Check the available options and expand the available menus.

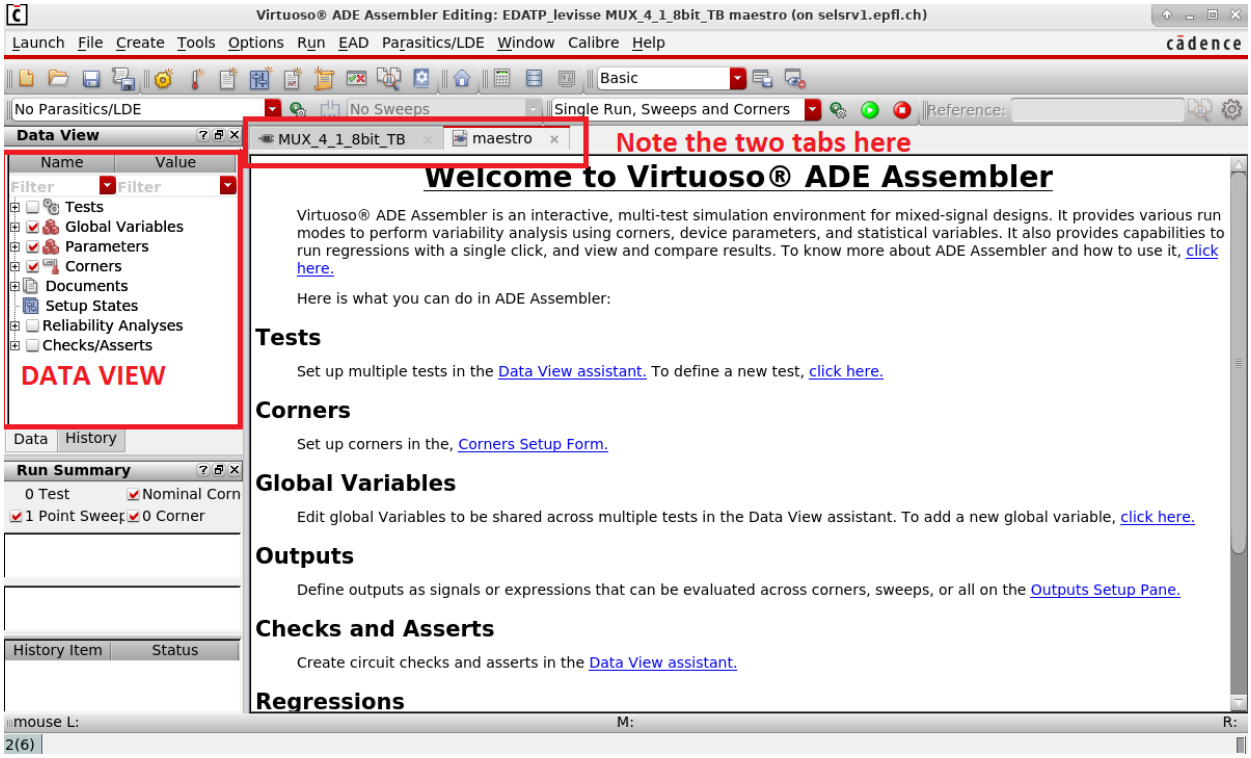
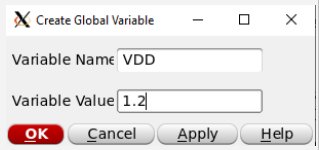
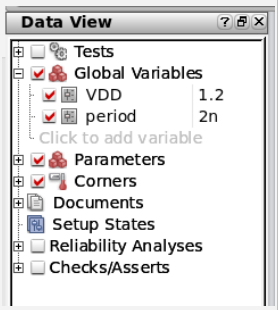


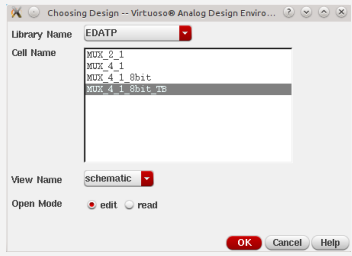
Figure 2 - The Analog Design Environment Maestro Window

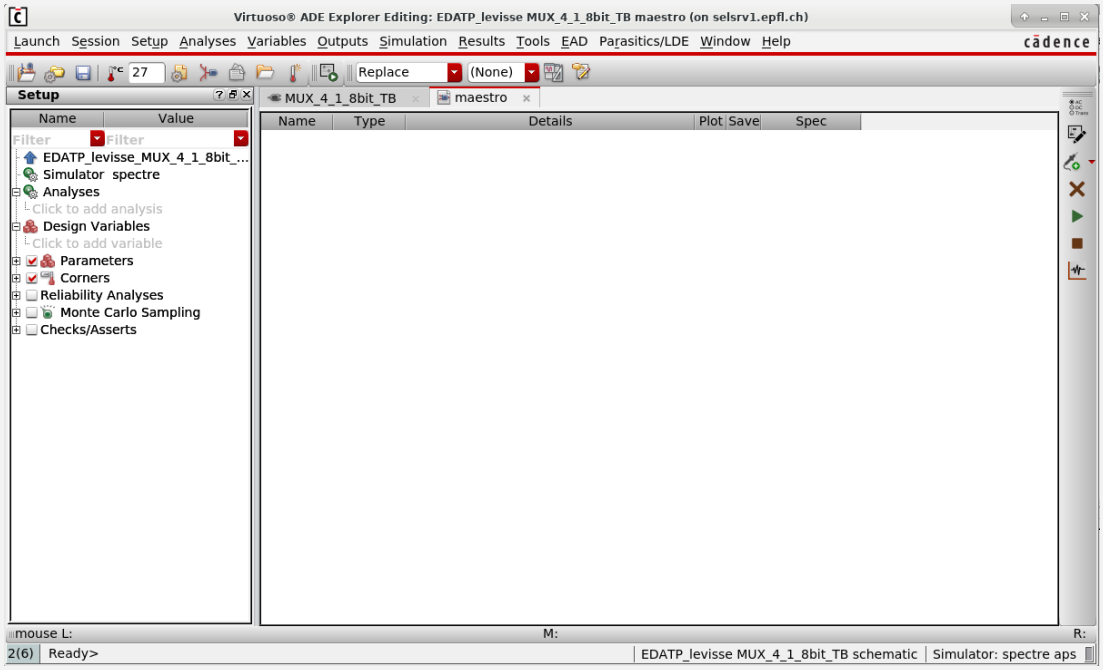
In case you used ADE in the past, ADE L and XL are soon discontinued and now replaced by ADE Explorer and Assembler respectively. Generally, ADE Explorer can be used for simple simulations and test. More complex exploration such as parametric simulations should be done through ADE Assembler.





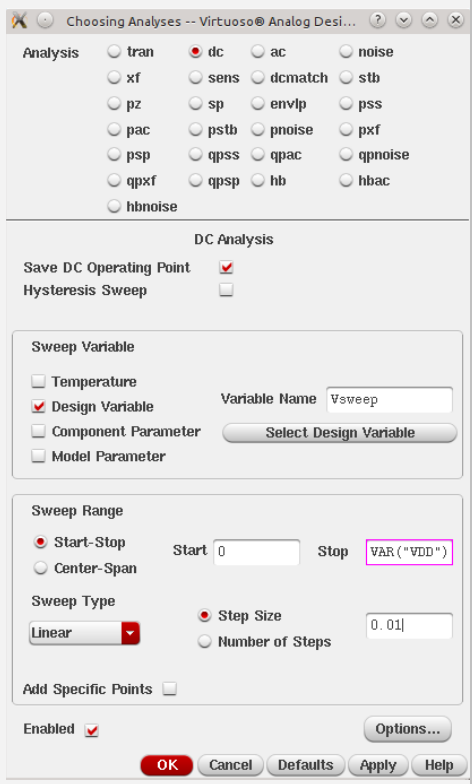
- ✓ We will first set our global variables. Expand the ***Global Variables*** menu in the ***Data View*** window. Use ***Click to add variable*** button to add the following variables: ***VDD***=1.2 (since we used ***VDD*** as a supply voltage variable, this defines your supply as 1.2V), ***Period***=2n (this sets the minimum period of the A input to 2ns).
- ✓ Your global variables should appear in the ***Data View*** window. The ***Period*** and ***VDD*** from Table I are now defined.
- ✓ Now we will define our tests. Expand the Tests menu, and use ***Click to add test*** button to define a test.
- ✓ A window will appear prompting you to choose a cell name for which you wish to define a test. Choose **MUX_4_1_8bit_TB** and click OK.
- ✓ A Test Editor window now appears (experienced designers would notice that this window is a ADE Explorer view). Note the little ⬆ next to the name of the test, on the left hand side of the window. With this arrow you can come back to the ADE Assembler view. Click on it.
- ✓ You are now back in the ADE Assembler view. Click back on the ⬇ in the newly created test to come back in the Explorer view.





- ✓ Once we are back in the ADE Explorer window, let's now create a test. First, let's recover all the defined design variables. Right Click on ***Variables → Copy From Cellview.*** Observe that ***VDD*** and ***Period*** variables will appear automatically in the ***Test Editor*** window, as well as the defined ***Vsweep*** variable (***Vsweep*** also appears in ***Global Variables***). **Do not** set any values here, since we will use the global definitions.

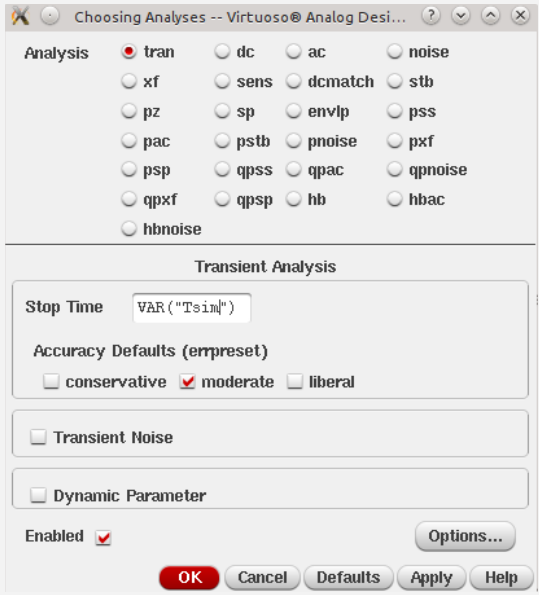
  - *When you copy the variables from the design (cellview) both global and design (local) variables will be created. However, the defined design variable values are valid only for the specific test (simulation), while global values are valid for all the defined tests.*
  - *If for a specific test we want to define a value different from the global one, we can simply define the local variable. Global value will not be used for that test.*



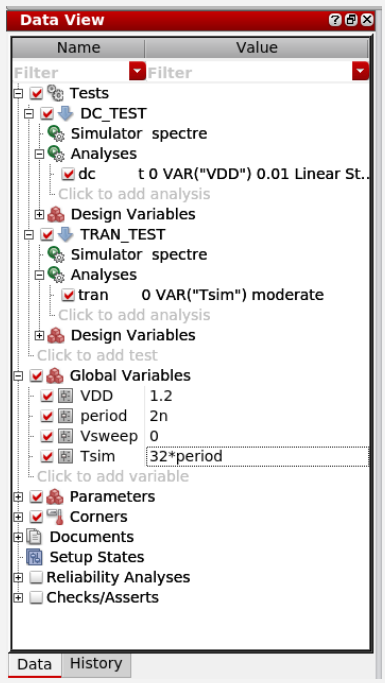
- ✓ Let's now choose the Analysis. Go to ***Analysis → Choose*** *(or in the Analysis pane, "click to add analysis")*, and select DC analysis. Set the remaining values as presented in the figure.
- ✓ Make sure that in the ***Stop*** field the defined limit is: ***VAR("VDD")***. This will define the DC Voltage sweep from zero to the variable ***VDD*** (rail-to-rail sweep), varying the input ***Vsweep***.

  - *For every field that expects a real (numeric) value, if a variable is used, the syntax (VAR("")) is needed. The function VAR returns the real value of the variable.*

- ✓ Click OK and the test is defined.
- ✓ Note that by double clicking on the name of the test you can rename it. Let's call it "**DC_TEST**"



- ✓ Click on the to go back to ADE Assembler.
  - ℹ ***Good Practice*** *: Every now and then, have a click on the* *to save your view.*
- ✓ Define a new global variable ***Vsweep*** (make it zero).
  - ℹ *The default value (zero) will be used by the simulator for calculating the DC operating point.*
- ✓ Next, let us define a Transient Analysis. We define it as a new test, so we must repeat the whole procedure by using ***Click to add test...***
- ✓ Choose **MUX_4_1_8bit_TB** again, and subsequently choose transient analysis by selecting ***tran***.
- ✓ Set up all the fields as in the figure. Set the simulator accuracy to moderate. For the ***Stop Time*** field, use the variable ***Tsim.*** Click OK.



- ✓ This operation creates the **Tsim** design variable, and adds a new test.
- ✓ **Do not forget** to copy the variables from the cellview as before. Rename the test **TRAN_TEST**. And click on to go back to the Assembler window.
- ✓ Finally, define the ***Tsim*** global variable that determines the simulation time. Define it to be 32 periods or: ***Tsim=32*Period*** (see figure).
  - ℹ *Note : if you click on the little "+" next to the design variables of one of the tests, you will note that these variables are ~~crossed~~. This means that the test will consider the global variables values. You could totally specify some special values you'd like to use for some tests without impacting the whole design.*
- ✓ <u>DC test</u> represents the analysis of the operating point (DC - level). We defined a DC sweep, which represents a special kind of DC analysis. We simply vary the value of the input from ***0*** to ***VDD,*** and observe the corresponding output values.
- ✓ <u>Transient test</u> represents the time domain analysis. We apply voltage pulses to our inputs and simulate the circuit for the specified simulation time (***Tsim***). Simulation time is defined as 32 periods of the fastest pulse (see Table I).
  - ℹ *Note that it is also possible to define different types of analysis within the same test, by using:* ***Click to add analysis****. For the different analysis that observe the same or similar outputs (such as for example DC and AC analysis in some cases), this option can also be used.*

4 DEFINING OUTPUTS AND RUNNING SIMULATIONS

So far, we have defined design variables and the type of simulations that we want to perform. Before we launch the simulations, we must define the outputs that we want to observe. Output could be defined after the simulations are performed, if the good options are selected, though, it means that every single net of the circuit will be saved – this will induce a large memory footprint and can be a problem for large circuits. Therefore, it is generally a better idea to define outputs before we perform simulations.

Note that you will generally not be the only user of a server, thereby having a mindful utilization of resources is always a good practice.

✓ In your ADE Assembler window, select the **maestro** tab. In the ***Output Setup*** section, click on the arrow next to the ***Add new output*** button. A falling menu will appear. From there, choose the desired test for which you want to define the output, as well as the desired type of the output. Define an expression output for DC_Test and a signal output for Tran_Test.

✓ If you followed everything correctly, you should have the following list of outputs (you can change the name by clicking on the name field):

2 rows

Test	Name	Type	Details	EvalType
Filter	Filter	Filter	Filter	Filter
DC_TEST		expr		point
TRAN_TEST	MUX_out	signal		point

Defining names can be very useful once the signals/expressions are plotted. It can allow us to easily distinguish the expressions in the plot/graph.

✓ When double-clicking on the Details line of the DC_test expression, note that a little appears. If you click on this, it pops a window in which you can write or paste expressions. This is extremely useful when you know what expression to write, or you copied it from somewhere else.

✓ If you do not know the syntax of the expression you want to write (which is your case now), you must use the calculator. Click on or Tools>Calculator

Virtuoso Calculator *is a very powerful tool. It can help you define waveforms and expressions, plot circuit time or frequency response, perform useful transforms, signal post-processing and/or analysis. It has many pre-defined mathematical and processing functions, but it also allows you to define your own functions and transforms.*

Calculator *can be opened faster by using* ***Tools→ Calculator*** *or by using a calculator icon from the toolbar. Once you open the calculator, there is no need to close it and open it again, until you are completely finished with using it.*

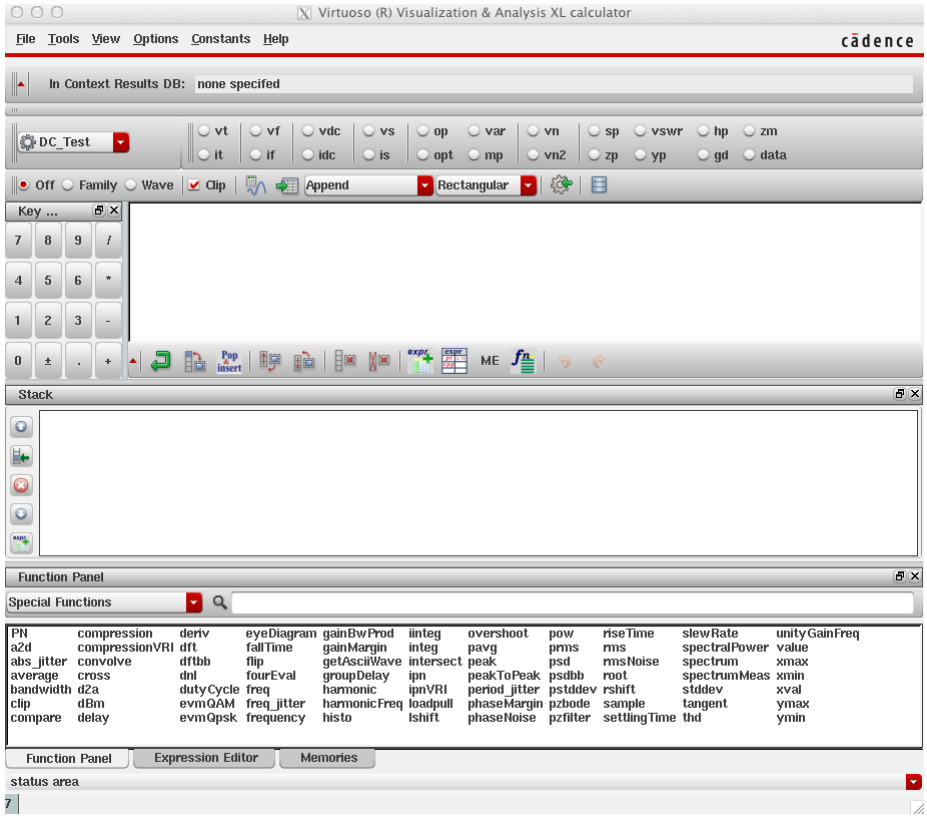


Figure 3 – Virtuoso Calculator

- ✓ Now, we propose to plot the expression of the signal which is deep within our design's hierarchy. To be able to do so, we first need to enter the corresponding cellview. **Do not** close the ***Calculator*** and go back to the ADE Assembler window.
- ✓ Switch from the ***maestro*** tab to the tab with the testbench schematic. Descend inside your 8_bit MUX as explained before (***Shift+e***). Once you enter the lower level, descend inside the **MUX_4_1**. Since there are 8 instances of **MUX_4_1**, you will be prompted to select which iteration to use. Choose zero and proceed.

➔ *Note: The "new tab" option allows you to open the next hierarchy level without closing the current one in the same window. If you do not plan to edit anything, use the "read" option. You can always make a read-only view editable later through "file>Make Editable"*

- ✓ Once you enter the **MUX_4_1** schematic, descent into MUX3 (check again the **MUX_4_1** schematic from the previous exercise).
- ✓ Once you entered in the transistor-level schematic of MUX3, switch back to the ***Calculator***.
- ✓ Click on ***vs*** (voltage sweep) circular box on the ***Calculator.*** You will be directly prompted back to the schematic. Now click on the **S_INV** label (wire name) or the wire itself. An expression will appear in the calculator buffer. Click on to send the expression to the ADE output panel.

- ✓ Note that a new line has been created in the output setup panel. You can now delete the firest expression you created by selecting it and clicking on the button. Rename the new DC_test expression **INV_out**.

Test	Name	Type	Details
Filter	Filter	Filter	Filter
DC_TEST		expr	
DC_TEST		expr	VS("/I0/MUX41<0>/I4/S_INV")
TRAN_TEST	MUX_out	signal	

- ✓ Now we need to define the output signal for the transient test. Go back to the schematic, and return to the top level (**MUX_4_1_8bit_TB**) (use ***Ctrl+E***). In the ***maestro*** tab, double-click on the ***Details*** field, and click on (…) symbol.
- ✓ Click on ***Q_out<7:0>*** bus. You will be asked to select the exact bit. Choose ***Q_out<0>,*** and press OK (since each bit of the MUX is the same, we can choose any of the 8 bits). In the same manner, define signals for the transient test for ***A<0>, B<0>, C<0>, D<0>, S<0>*** and ***S<1>***. Once you finish, if everything is correct, you should have a list of outputs as follows:

Test	Name	Type	Expression/Signal/File	EvalType	Plot	Save
DC_Test	INV_out	expr	VS("/MUX_8bit/MUX41<0>/...	point	✓	
Tran_Test	MUX_out	signal	/Q_out<0>	point	✓	
Tran_Test	A_in	signal	/A<0>	point	✓	
Tran_Test	B_in	signal	/B<0>	point	✓	
Tran_Test	C_in	signal	/C<0>	point	✓	
Tran_Test	D_in	signal	/D<0>	point	✓	
Tran_Test	S0_in	signal	/S<0>	point	✓	
Tran_Test	S1_in	signal	/S<1>	point	✓	

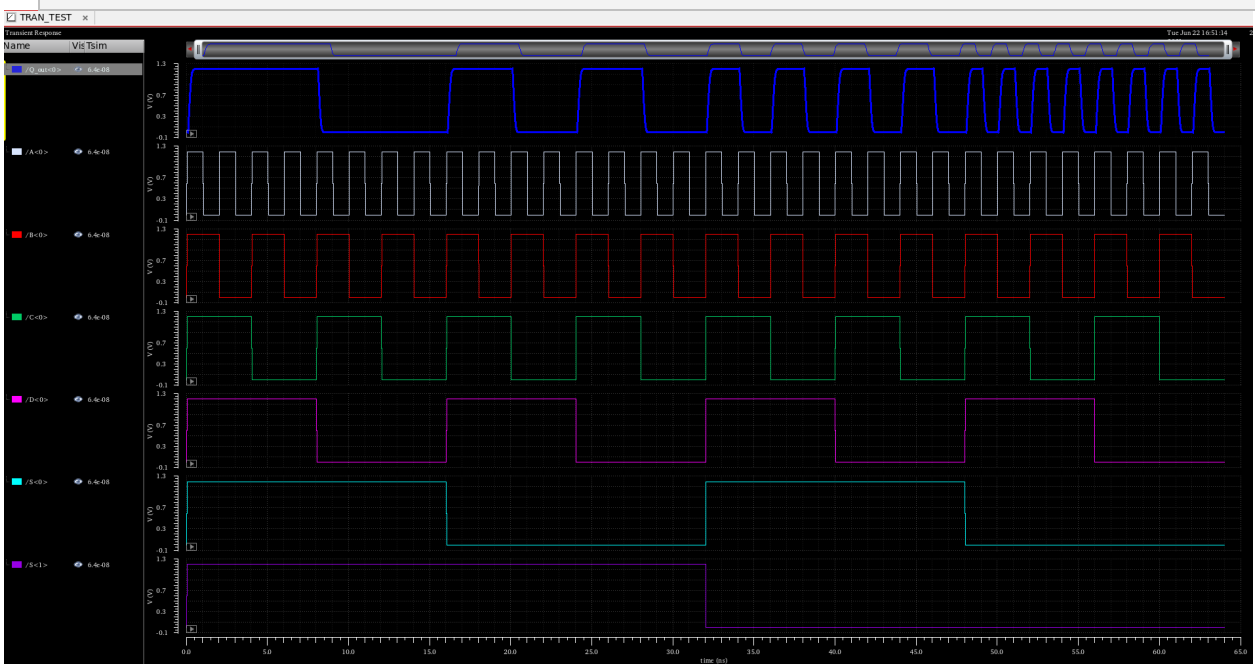


Figure 4 – Transient Test Output Waveforms

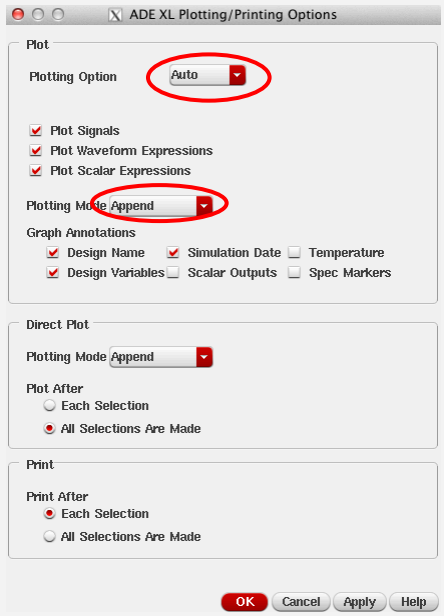
- ✓ All the outputs are now defined. The plotting options can be set by using the ***Set up plotting options*** button or from the main menu ***Options → Plotting/Printing …*** . To plot the selected outputs automatically, we need to set-up as shown in the figure (right).

  ➔ *You can also use the "new win" option there.*

- ✓ Before any simulation can be run, we **always** need to ***Check and Save*** all the schematics that we have changed in the meantime (this sometimes means we have to descend in hierarchy and save every hierarchical level - not performing ***C&S*** is a very common mistake).

- ✓ Finally, the simulations can be run. This can be done by pressing the ***Run Simulation*** button in the ADE Assembler window .

- ✓ You will be able to follow the progress of the simulations in the ***Run Summary*** sub-window.



Here give it some time. You are running two different simulations, each needing some time to be configured and performed. Keep an eye on the summary window. On Options>Job Setup, enabling "show output log on error" is a good practice.

- ✓ Once the simulations are finished, ***Virtuoso Visualization and Analysis XL (ViVA)*** window will appear. By default it is embedded on the right hand side of the Assembler view. Extend it as suggested by the popup. Click on .

- ✓ Two tabs are available: DC-test tab, and the transient test tab (Figures 4 and 5).

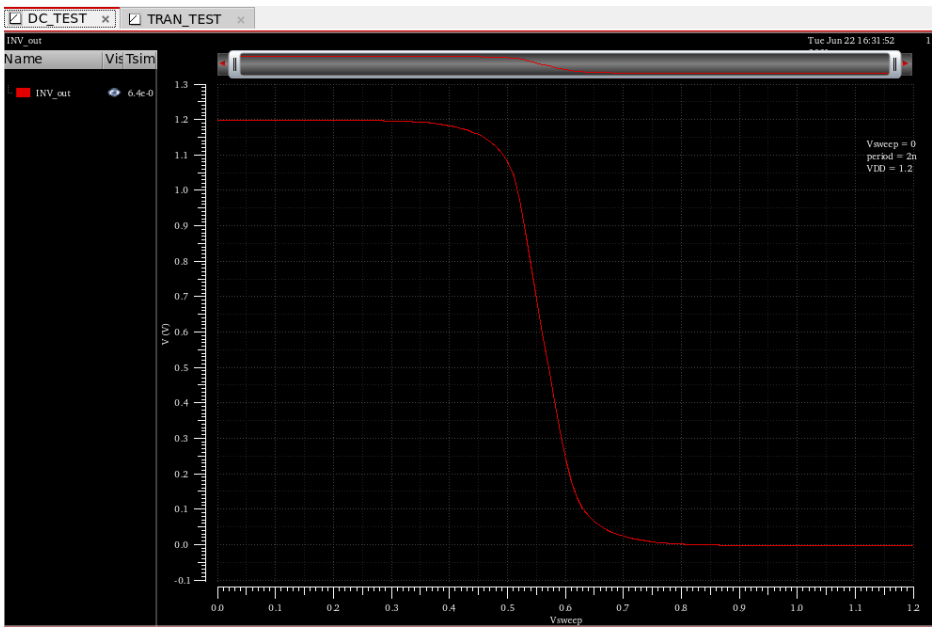
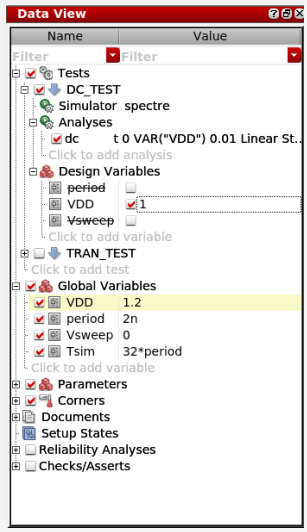


Figure 5 – DC Test Voltage Sweep

QUESTION 4-1 : Explain the waveform of the multiplexer output, what does it do ?
QUESTION 4-2 : How do you interpret the results of the DC simulation? Explain the resulting waveform. What does it represent?
QUESTION 4-3 : What is the threshold voltage of the NMOS transistor? What about PMOS transistor?

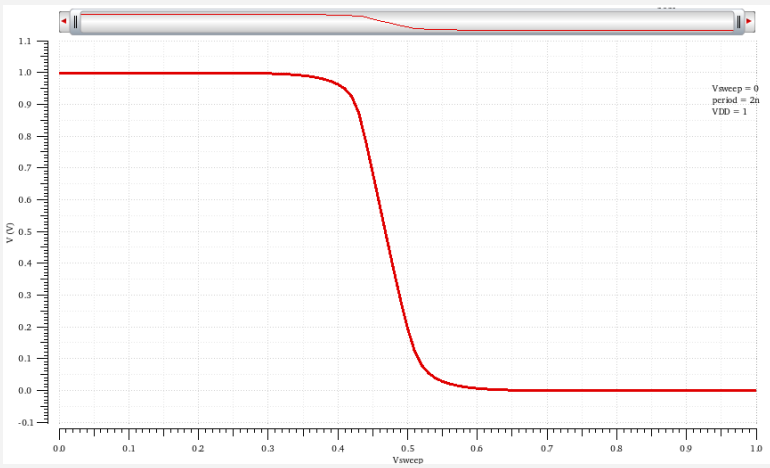
5 USING LOCAL VARIABLES

Until now, we were using global variables. It is also possible (and sometimes useful) to define variables localized to the specific test.



- ✓ Go to the ***Data View*** window and expand the design variables menu of the **DC_Test.**
- ✓ Check the box next to ***VDD*** and type 1 in the field next to box.
- ✓ The local value of the ***VDD*** for the **DC_Test** will now be 1.0V.
- ✓ To avoid waiting for the transient simulation to finish, uncheck the box next to the transient simulation. Transient simulation will not be performed.
- ✓ Run the simulation again. Comment on the results. Does local variable ***VDD*** depend on the global variable ***VDD***? What has changed in the ***Run Summary*** sub-window?

  - *Note that by selecting **Graph→Properties...** in the **Visualization** window (**General** tab), you can change the graph options such as the background. This can be very useful for including graphs from **Virtuoso** in the presentation slides or project reports.*
  - *Note that as we did setup the plotting mode as append (Options>Plotting/Formatting) if you did not close the Visualizer it will superpose the new curve on top the previous one.*



- ✓ **Disable the local variable after this experiment to have your VDD back to 1.2V.**

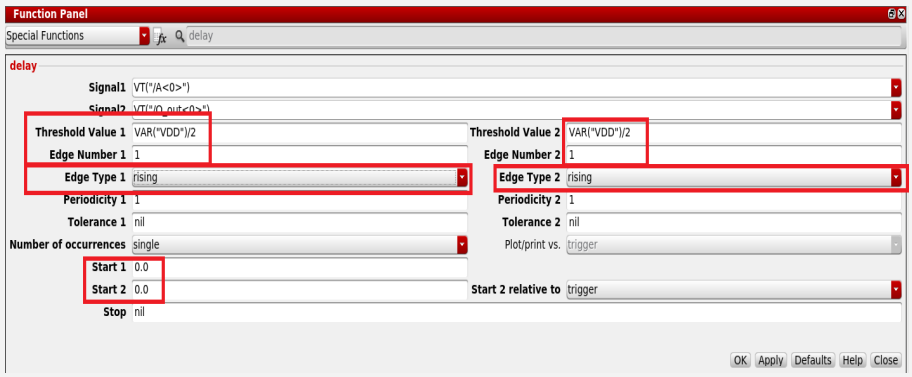
6 USING SPECIAL FUNCTIONS AND SPECIFICATIONS

As we mentioned, ***Calculator*** allows us to use many different features and functionalities. A very useful group of these features are so called: ***Special Functions***. Here, we will learn how to use special functions on a very simple example: ***delay*** function.

- ✓ In the ***Data View*** window, disable the DC_Test (untick it) and enable the TRAN_test (tick it).
- ✓ Open the ***Calculator,*** and make sure that the ***Special Functions*** are selected in the Function Panel as in Figure 3de.
- ✓ Press ***vt*** (voltage transient) button, and go back to the schematic. Click on ***Q_out<7:0>*** bus. You will be prompted to enter the exact bus bit: choose ***Q_out<0>*** *and click* ***ok.*** The value then appears in the calculator buffer. **Do not** press anything, go back to schematic and click on bus ***A<7:0>.*** Once again you will be prompted to enter the bit: choose ***A<0>***.

  ℹ *In the calculator. Note that the voltage transient (VT) of Q_out<0> is now be in the calculator* ***Stack.*** *Press* *to add VT("/A<0>") in the stack. This way you'll be able to select it from the delay function.*

- ✓ Go back to the calculator and click on ***delay*** function in the ***Function Panel***. The following dialog should appear:



- ✓ Modify the values as specified in the figure. ***Threshold Value***s should be ***0.6***, ***Edge Number***s ***1***, and ***Edge Type***s set to rising. This will measure the delay from the first rising edge of ***A<0>,*** until the first rising edge of ***Qout<0>*** that appears after the specified rising edge of ***A<0>***. Edge is defined as crossing the threshold of **VAR("VDD")/2**. Make sure that ***Start 1*** and ***Start 2*** is at ***0.0***. After you set your delay function dialog as shown, click ***Apply***, and the full expression should be available in the ***Calculator*** buffer.

  ℹ *If the delay function dialog does not appear correctly as shown, you should delete all data from the calculator buffer and stack, then try again all the previous 3 steps.*

- ✓ Make sure that ***Tran_Test*** is selected in the ***Calculator*** window and once the full expression is available, use the button (***Send buffer expression to ADE Outputs*** - button). Click on the button and go back to the ***Output Setup***.
- ✓ The corresponding output expression is now defined and available. Name it as suggested:

Tran_Test	D_Delay_R	expr	delay(?wf1 VT("/A<0>") ?value1 0.9 ?edg...	point	☑ ☐

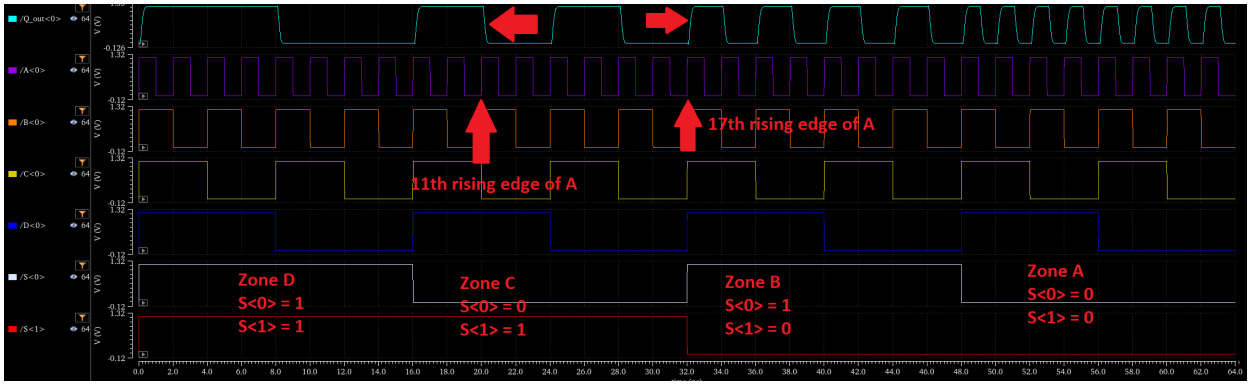


Figure 6 – Measure Different Delays

- ✓ Note that now we have defined a single delay value. By using the same principle, define 3 more expressions to represent the delay between: 11th rising edge of ***A<0>*** and the following falling edge of ***Qout<0>;*** 17th rising edge of ***A<0>*** and the next rising edge of ***Qout<0>;*** 25th falling edge A<0> and the next falling edge of ***Qout<0>*** *(e.g. edge 11 rising of A<0> to edge 1 falling of Q<0> - look below for an important tip)*.

  ℹ ***Important!*** *The edge number for the first signal is absolute. The edge number of the second signal represents the number of edges from the specified edge of the first signal.*

  ℹ *Also note that the expressions can be changed manually (directly), which can allow you to perform the definitions faster. You can achieve that by observing the expression:*

*delay(?wf1 VT("/**net1**") ?value1 **th1** ?edge1 "**type1**" ?nth1 **N1** ?td1 **ref1** ?tol1 **t1** ?wf2 VT("/**net2**") ?value2 **th2** ?edge2 "**type2**" ?nth2 **N2** ?td2 **ref2** ?tol2 **t2**)*

  and by modifying the corresponding numeric values: ***net1,2*** *- corresponding nets in the schematic,* ***th1,2*** *- thresholds,* ***type1,2*** *- type of the edge (rising or falling),* ***N1,2*** *- edge number,* ***ref1,2*** *reference points,* ***t1,2*** *tolerances.*

- ✓ Add each expression to the ***Output Setup.***

  ℹ *Here we define 4 zones to characterize the MUX. Zone D corresponds to the case where both S<0> and S<1> are equal to logic 1. Zone C corresponds to S<0> being 0 and S<1> being 1. Same for B with S<0> being 1 and S<1> being 0. Finally zone A corresponds to both S<0> and S<1> being 0. For each of them we define the names accordingly.*

- ✓ In the ***Output Setup*** list, for each delay expression, add a specification. Click on a ***Spec*** field and choose "<". By double-click on the empty field, enter the value of ***190p,*** next to the "<" sign. This defines our design specification. Our delay is required to be lower than 190ps.

- ✓ If you performed everything correctly, you should have the following outputs in your ***Output Setup*** window (rename the expressions as shown below):

TRAN_TEST	D_Delay_R	expr	delay(?wf1 VT("/A<0>") ?value1 (VAR("VDD") / 2) ?edge1 "risi...	point	✓		< 190p
TRAN_TEST	C_Delay_F	expr	delay(?wf1 VT("/A<0>") ?value1 (VAR("VDD") / 2) ?edge1 "risi...	point	✓		< 190p
TRAN_TEST	B_Delay_R	expr	delay(?wf1 VT("/A<0>") ?value1 (VAR("VDD") / 2) ?edge1 "risi...	point	✓		< 190p
TRAN_TEST	A_Delay_F	expr	delay(?wf1 VT("/A<0>") ?value1 (VAR("VDD") / 2) ?edge1 "fall...	point	✓		< 190p

- ✓ Descend into the **MUX_2_1** schematic. For every PMOS transistor width (***Total Width***), put ***300nm*** and for every NMOS (***Total Width***) put ***80nm***.

- ✓ Run the simulation .

- ✓ Go to the ***Results*** tab in the **maestro** section. The following results should appear:

TRAN_TEST	D_Delay_R	178.7p	< 190p		pass
TRAN_TEST	C_Delay_F	199.7p	< 190p		near
TRAN_TEST	B_Delay_R	178.5p	< 190p		pass
TRAN_TEST	A_Delay_F	197.7p	< 190p		near

✓ Change the ***Spec*** to more demanding 180ps and run the simulation again. Discuss the meaning of these results with your colleagues.

TRAN_TEST	D_Delay_R	178.7p	< 180p		pass
TRAN_TEST	C_Delay_F	199.7p	< 180p		fail
TRAN_TEST	B_Delay_R	178.5p	< 180p		pass
TRAN_TEST	A_Delay_F	197.7p	< 180p		near

ℹ This tool allows you to evaluate the quality of an obtained results with regard to a metric that you can define. It is a good way to identify fast paths are being the bottleneck in a design.

QUESTION 6-1 : What does the pass/near/fail feature allows you to do ?

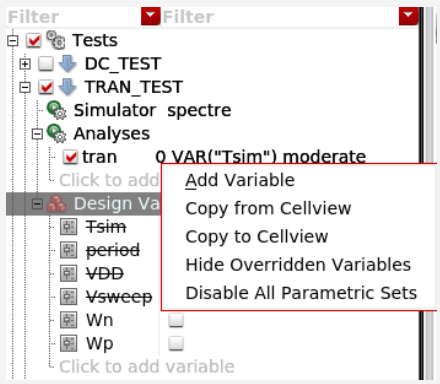
QUESTION 6-2 : These delay functions are extremely useful, and allow you in certain conditions to save memory (as you do not need to save the entire waveforms) and time. Though, as usual, they have their limits. What kind of pitfall do you see there ?

7 USING DESIGN/INSTANCE PARAMETERS

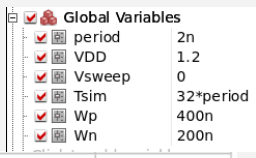
- ✓ Use the calculator to add four more delay expressions. You should have one rising edge and one falling edge delay for every pulse type (***A, B, C, D*** - see Figure 6). After you run the simulation, depending on what edges you have chosen, you should have a result very similar (or same) as the following:

TRAN_TEST	D_DELAY_R	178.7p	< 180p		pass
TRAN_TEST	C_DELAY_F	199.5p	< 180p		fail
TRAN_TEST	B_DELAY_R	178.6p	< 180p		pass
TRAN_TEST	A_DELAY_F	197.8p	< 180p		near
TRAN_TEST	D_DELAY_F	201.2p	< 180p		fail
TRAN_TEST	C_DELAY_R	188.3p	< 180p		near
TRAN_TEST	B_DELAY_F	199.5p	< 180p		fail
TRAN_TEST	A_DELAY_R	188.7p	< 180p		near

- ✓ Now, descend into the **MUX_2_1** schematic. For every PMOS transistor width (***Total Width***), instead of 300nm value, type a parameter ***Wp***. For every NMOS transistor do the same and set ***Wn***.



- ✓ ***Check and Save*** the **MUX_2_1** schematic.
- ✓ In the TRAN_TEST, right click on design variables and select Copy from Cellview. It will the design variables Wp and Wn.
- ✓ On the Global Variable panel, define the global variables ***Wn = 200n*** and ***Wp = 400n.***
- ✓ Run the simulation .
- ✓ Go to the ***Results*** tab in the ***maestro*** section. The following (or similar) results should appear:



TRAN_TEST	D_DELAY_R	147.7p	< 180p		pass
TRAN_TEST	C_DELAY_F	160.3p	< 180p		pass
TRAN_TEST	B_DELAY_R	147.3p	< 180p		pass
TRAN_TEST	A_DELAY_F	158.9p	< 180p		pass
TRAN_TEST	D_DELAY_F	161.4p	< 180p		pass
TRAN_TEST	C_DELAY_R	154.8p	< 180p		pass
TRAN_TEST	B_DELAY_F	160p	< 180p		pass
TRAN_TEST	A_DELAY_R	155.2p	< 180p		pass

QUESTION 7-1 : How do you explain the decrease in delay?

8 CURIOSITY ON DEVICE SIZING AND BEST WAYS TO HANDLE VARIOUS CELL SIZING.

A lot of ways to parametrize transistors and cell exist in virtuoso. An extremely detailed documentation exists on it in the Cadence support. Generally, learning about it once in a company is a really good practice. Still, defining parameters as we just did can be suboptimal as it fixes the size of the MUX_2_1 transistors for ALL the instantiations of this cell. Still, parametrizing cells can be good for simulation (this can be done through the CDF file – I write this here so that you have the keyword), however, when it comes to layout (next session), having hardcoded parameters is the way to go. A easy and optimal practice consists in duplicating a Cell View (right click on the cell view > copy), rename it with another name (which accounts for its size – or drive) and change the sizing accordingly in the schematic. This way, each identical circuit with a different sizing exists as a different cell view. Take a look at the UMC65LL_UMK65LSCLLMVBBR__B03PB standard cell library in your library manager. All the cell names finish by *X*RA where X is a number ranging from 0 to 40+. These are the drive strength of the standard cells. At this point you will not be able to understand the layout, but trust us, these are the same cells (when they have the same name) and just the size of the output stage changes.

Alternatively, in virtuoso, one could create something called pcells (parametrizable). Though, these take a lot of effort to design, specifically when it comes to layout.

Finally, making a design too specific to one tool is never a good idea. Better having 10 different cells with 10 different sizing and names which you can import anywhere, than a complex cell which way not behave the same in another tool or with the next version of the same tool.

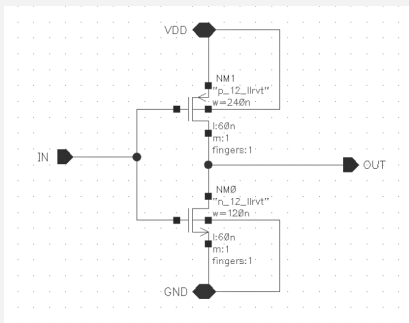
Take these suggestions as takeaways for the future. Though these are not the absolute truth, right ?

9 CORNER AND MONTE-CARLO SIMULATION

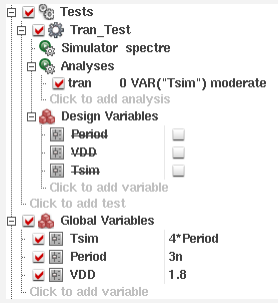
9.1 PROCESS CORNERS

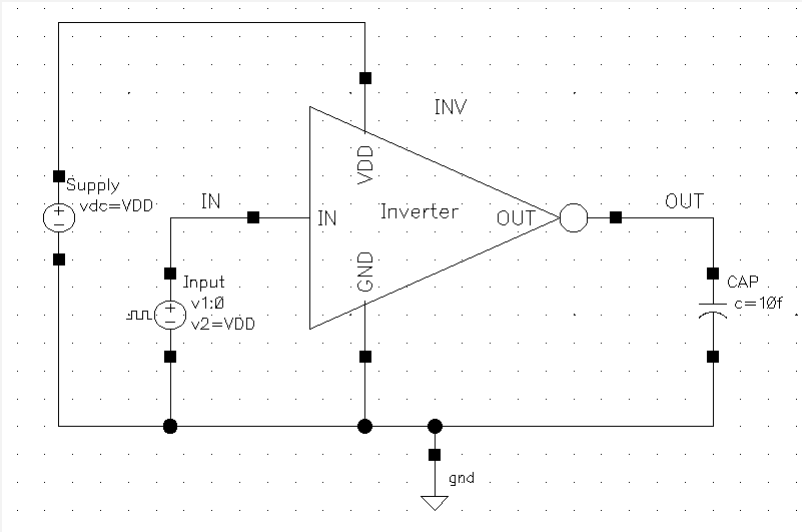
In integrated circuit design, corner simulations represent a technique to model the extreme cases of fabrication parameter variation and/or variation of other physical parameters such as temperature or supply voltage. Once fabricated, depending on the fabrication process inaccuracy, devices may exhibit different behavior and therefore be faster, slower, larger, smaller or in any sense vary from the ideal case. Moreover, an integrated circuit may be exposed to different environmental conditions such as high temperature or battery supply voltage drop. Corner simulations allow us to model these cases and guarantee that the circuit will still be functional. This is one of the methods that enables IC designers to estimate the ***yield*** or the percentage of fabricated circuits that will be functional.

- ✓ In the EDATP library, create a new schematic cell and name it **Inverter.** Use **N_12_LLRVT** and **P_12_LLRVT** transistors from **UMC65LL** library and draw a CMOS inverter as in the figure. Set ***Wn*** = *120nm* and ***Wp*** = *240nm* (***L*** should be 6*0nm* for both). ***Check and Save*** the schematic.
- ✓ You could also create global variables Wp and Wn, and use then in the W field of the two transistors.
- ✓ Create a symbol for the ***Inverter***.



- ✓ Create a new schematic cellview in the **EDATP** library and name it **Inverter _TB**. Draw the testbench schematic as in the figure (down).
- ✓ For the DC voltage of the supply, use a variable ***VDD.*** For the input pulses, ***Voltage 1*** should be zero and ***Voltage 2*** should be ***VDD. Period*** should be set to a variable: ***Period***. Set the rise and fall times to ***0.01*Period***.
- ✓ Load capacitor: ***C = 10fF.***
- ✓ Create a ADE assembler cellview for it.
- ✓ Create a moderate accuracy transient test with the simulation time ***Tsim (Tsim=4*Period, Period = 3ns, VDD = 1.2V)***.





- ✓ Setup the input and the output of the inverter to be plotted and define the output delay of the falling/rising edge with respect to the **previous** rising/falling edge of the input, respectively:

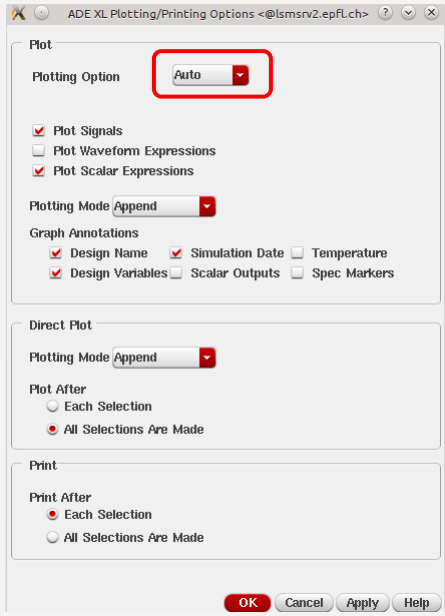
IMPORTANT COMMENT : here, carefully select VAR("VDD")/2 in the expression for the threshold value ! otherwise, comparing delays between circuits with different VDD values will not be apple to apple

e.g. for Delay_R: *delay(?wf1 VT("/IN") ?value1* ***(VAR("VDD") / 2)*** *?edge1 "rising" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/OUT") ?value2* ***(VAR("VDD") / 2)*** *?edge2 "falling" ?nth2 1 ?tol2 nil ?td2 0.0 ?stop nil ?multiple nil)*

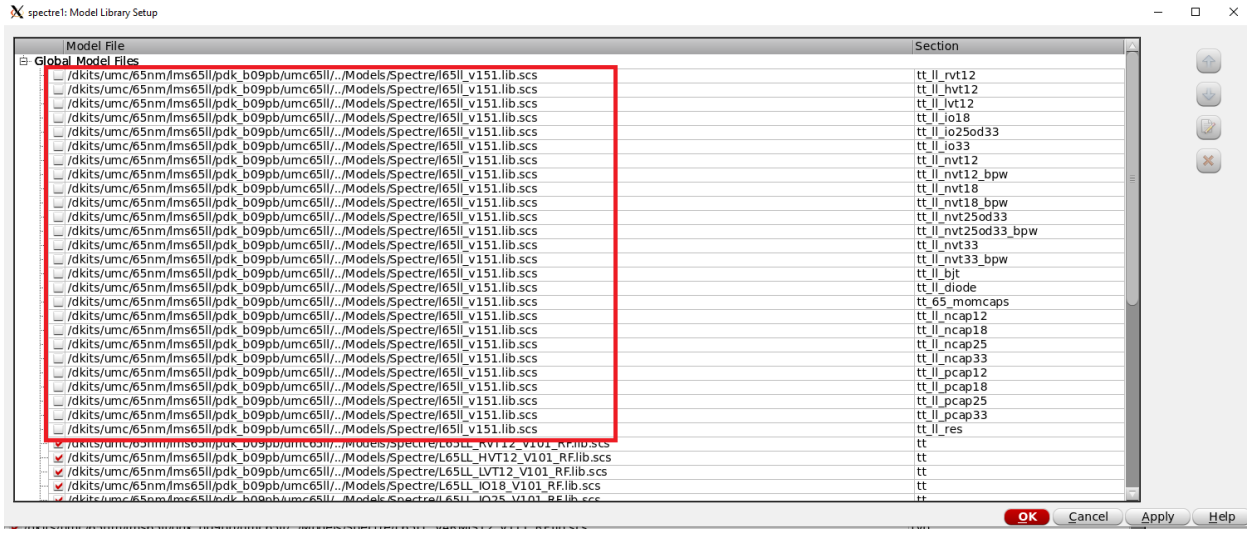
Test	Name	Type	Expression/Signal/File	EvalType	Plot	Save	Spec
Tran_Test	Input	signal	/IN	point	✓		
Tran_Test	Output	signal	/OUT	point	✓		
Tran_Test	Delay_R	expr	delay(?wf1 VT("/IN") ?value1 0....	point	✓		< 100p
Tran_Test	Delay_F	expr	delay(?wf1 VT("/IN") ?value1 0....	point	✓		< 100p

- ✓ Set the plotting/printing options as presented in the figure on the right.
- ✓ ***Check and Save*** the schematics and run the simulation .
- ✓ Analyze the results.
- ✓ In order to perform corner simulations, we have to use specific transistor models (often called statistical models), that take into account the process variation.
- ✓ Right-Click on the transient test in the ***Data View*** window, and select ***Model Libraries...*** The following dialog should appear (see below).
- ✓ Untick all the models coming from **l65ll_v151.lib.scs.** This action disables typical (deterministic) models.

  Note that there are two parts in this window. One being the file name, and one being the section. If you open the scs file, you'll see several sections one for each device and corner.



- ✓ Use the button ***Click here to add model file...*** Proceed with the browse button (...), and go to ***./Models/Spectre/Monte_Carlo*** folder.

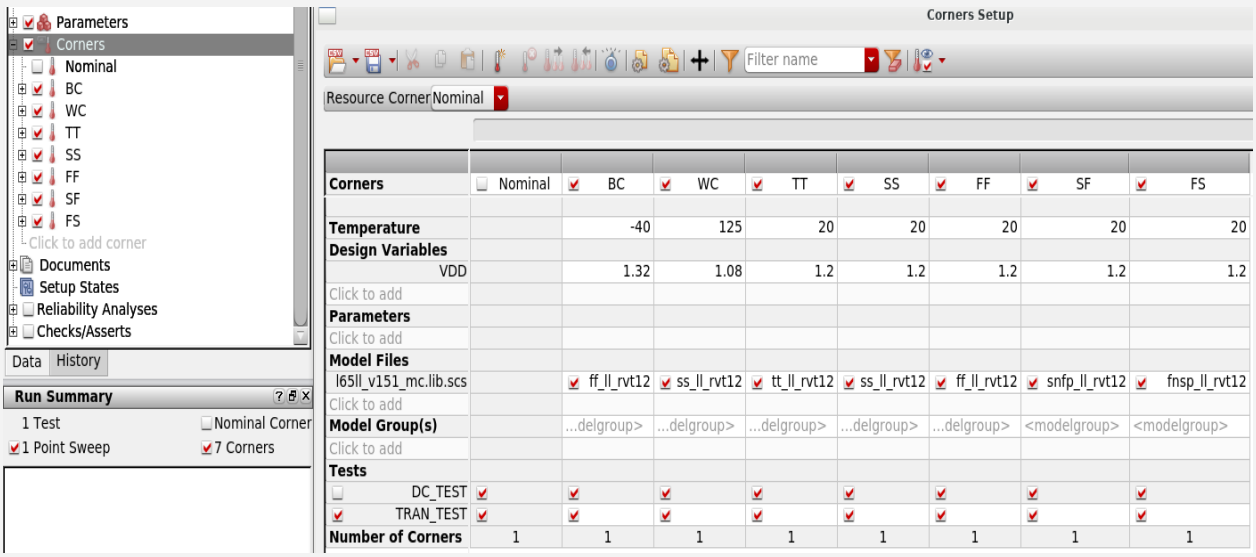

spectre1: Model Library Setup
Model File
Section
Global Model Files
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_rvt12
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_hvt12
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_lvt12
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_io18
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_io25od33
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_io33
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt12
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt12_bpw
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt18
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt18_bpw
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt25od33
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt25od33_bpw
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt33
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_nvt33_bpw
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_bjt
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_diode
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_65_momcaps
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_ncap12
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_ncap18
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_ncap25
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_ncap33
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_pcap12
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_pcap18
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_pcap25
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_pcap33
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/l65ll_v151.lib.scs tt_ll_res
tt
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/L65LL_HVT12_V101_RF.lib.scs tt
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/L65LL_LVT12_V101_RF.lib.scs tt
/dkits/umc/65nm/lms65ll/pdk_b09pb/umc65ll/../Models/Spectre/L65LL_IO18_V101_RF.lib.scs tt
OK
Cancel
Apply
Help

- ✓ Select the file: **l65ll_v151_mc.lib.scs**, and click ***Open***. In the Section field type: ***tt_ll_rvt12***, or select it from the dropdown menu.

  ℹ *You do not need it here (unlike in the screenshot), however some PDK will require the variable **sigma.** This variable determines the degree of statistical variation and generally depends on the target yield. It may needed for the statistical models to define the standard variation of the parameters. It will be used in the background by the simulator and the most common mean (and default) value is equal to 3.*

  ℹ *If you define it, Spectre will inform you through a warning that the sigma variable is not used.*

- ✓ Now, we will define several corners:



- ✓ In the ***Data View*** window, use the button ***Click to add corner*** under ***Corners.*** Click on , to add additional corners.

- ✓ The first corners we add are the BC (Best Case) and WC (Worst Case). These are generally used in digital design, to characterize the physical worst cases of a digital chip, taking into account voltage fluctuations (+/-10%), temperature range (max/min) and process variations. This will be more detailed in the second phase of this labs (from week 6). These corners are used to identify the slowest possible path and the fastest possible path in a digital circuit.

- ✓ The next corners are the process corners we generally define in full custom design, and could be adapted to various operating voltage and temperature. There are 5 of them. TT (Typical Typical), SS (Slow Slow), FF (Fast Fast), SF or SnFp (Slow N Fast P), and FS or FnSp (Fast N Slow P). These corners represent the extremum cases that the foundry (the company fabricating the chip) ensure about the variability of the MOS transistors This way one can explore the performance of their design considering the Vt variation of the transistors. For e.g. SnFp means Slow NMOS and Fast PMOS, which literally means "all the NMOS are the slowest possible and all the PMOS are the fastest possible" or "all the NMOS have a high Vt and all the PMOS have a low Vt". This being induced by the process variations.

  ℹ This is valid for MOS transistors. But if you had a look at all the files we showed before, you'll notice that this is also valid for all the device models the foundry provides (resistors, capacitors, diodes etc.). The principle being the same, but limited to the available parameters for this device.

- ✓ Set the temperature of all the additional corners to the same value as in the figure and the supply voltage to as described. Click to add a model file, and add the same as before: go to ***./Models/Spectre/Monte_Carlo/*** and select **l65ll_v151_mc.lib.scs.**

✓ Name the corners as ***BC, WC, TT, FF, SS, FS*** and ***SF***. Set the corresponding section (from the dropdown menu, or type them directly) accordingly: ***ff_ll_rvt12*** for ***FF***, ***ss_ll_rvt12*** for ***SS***, ***fnsp_ll_rvt12*** for ***FS*** and ***snfp_ll_rvt12*** for ***SF***. Click OK.

When defining the different corners, the nominal corner could be deactivated. It takes by default the files we did define before in the model libraries from the test TRAN_TEST. The other corners

✓ Untick the Nominal corner and run the simulations . Go to the results tab. Play with the different tabs available through the drop down menu on the top left of the results tab. The "detail" tab shows you the details results of your simulation. The "Detail – Transpose" shows you how each of the corners behaves with the delays you extracted.

Detail - Transpose | Filter ... | Append | (None)

7 rows

Corner	VDD	emperature	v151_mc.li
BC	1.32	-40	ff_ll_rvt12
FF	1.2	20	ff_ll_rvt12
FS	1.2	20	fnsp_ll_r...
SF	1.2	20	snfp_ll_r...
SS	1.2	20	ss_ll_rvt...
TT	1.2	20	tt_ll_rvt12
WC	1.08	125	ss_ll_rvt...

7 rows

Pass/Fail	/OUT	/IN	Delay_R	Delay_F
pass			67.23p	66.13p
pass			73.9p	80.12p
fail			84.15p	116p
fail			110.9p	89.6p
fail			132.6p	132.7p
near			96.78p	101.7p
fail			163.5p	183.3p

QUESTION 9-1 : What do you see there ? How different is the BC corner compared to the FF ? and why ? same comment with WC vs SS.

QUESTION 9-2 : What's the difference between SF and FS ?

QUESTION 9-3 : Here what would you do to make the circuit pass in all the corners? No need to run the simulation (or if you do so, it should be just to confirm your feeling). What parameters could you play with?

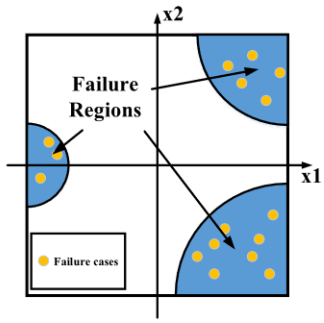
Hint to go further : there are knobs inside the inverter, but there could also be parameters <u>outside</u> of it… just saying.

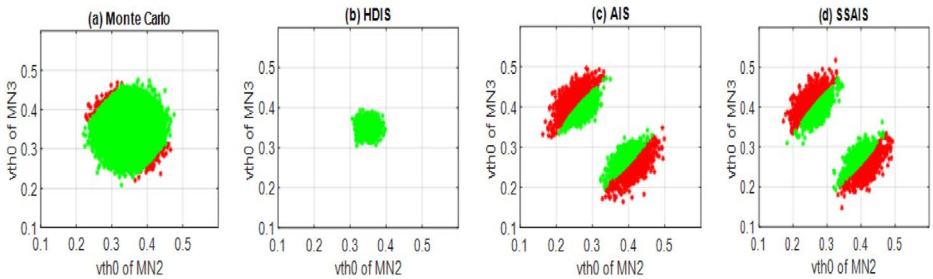
QUESTION 9-4 : What's your feeling about this kind of corner simulation ? can you identify already how limiting it can be ?

9.2 MONTE-CARLO SIMULATIONS

Corner simulations are useful for defining the extreme cases. Since there is in general a small amount of simulations to be run, the verification can be done quickly, and the designers can confirm the functionality of their circuit under the predicted extreme conditions. However, corner simulations have a couple of limitations. They can sometimes be very pessimistic, since they cover only extreme cases instead of the actual statistical distribution of the parameters. Another limitation is that corner analysis covers only predefined (predicted) limited set of parameter variations that have to be set manually. Finally, and this is the most critical aspect, corner simulation do only simulate cases where all the transistors are in the same conditions. In real life, it can happen that functionality is lost in cases where only a few transistors in a path become weaker/stronger. In other words, in some circuit types, failures do not always happen in the corners, and particularly not always with predictable voltage and temperature conditions. This is particularly true

the more technologies become advanced. For e.g., in SRAMs (static memories), a method called importance sampling is generally used to identify which part of the corner plane can be failing, and specifically running monte carlo simulations in this zone. Thereby saving a large amount of simulations. **Bottomline is : Corner simulations can be indicative for functionality verification, but are not a reliable verification for sign-off.**





Top figure[1] : model of how failure can happen in a complex circuit in a vtn/vtp graph. Here it is visible that failure do not always appear in the corners.

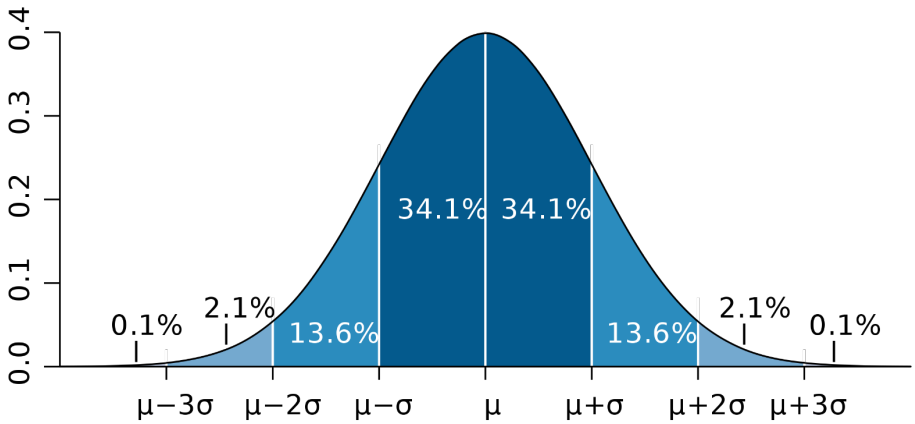
Bottom figure[2]: example of different statistical simulation algorithms that could be applied to some circuit showing failure in some specific area. Indeed, catching the same amount of failure can be done with less runs for a more specific algorithm that can be applied specifically to one area of the vtn/vtp plane. Still, this is a really advanced verification technique, and is only needed in circuits where extremely high failure coverage is required.

Statistical simulations are generally defined using a normal law or empirically with something we generally call the 68-95-99.7 rule. This considers a mean (μ) and a standard deviation (σ). We generally define the coverage of a given simulation by considering a certain amount of sigma around the mean. 1 sigma covers for 68.2% of the distribution, 2sigma for 95.4% and 3sigma 99.73%. Typically, in SRAM design, a coverage of 6+ sigma (1.973ppb – 1 failure over 506 797 346) is required to ensure reliability. This means that many monte-carlo simulation runs are required to ensure coverage on this probability of error. As a rule of the thumb, for a relatively small circuit (10-100 transistors) 5 to 10.000 runs can be considered enough to cover for 3sigma. Though, again this is not the absolute truth, and some mathematical models can be used to make sure that a complete coverage is achieved. Again, as an engineer all is question of balance between the amount of work/resources spent, and the trust you have in your results.

[1] Figure reprinted from :

https://ieeexplore.ieee.org/document/7564452 and https://ieeexplore.ieee.org/document/6740007

[2] Figure reprinted from https://www.sciencedirect.com/science/article/pii/S0167926022001729



Sources :

https://en.wikipedia.org/wiki/Normal_distribution

https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule

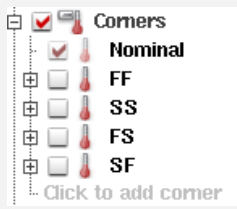
https://en.wikipedia.org/wiki/Six_Sigma

https://en.wikipedia.org/wiki/Importance_sampling

In IC design, we use Monte-Carlo simulations to evaluate the process variations of MOS transistors. Generally these variations are collapsed to variations in the Vt of the transistors. The more advanced technologies are, the more variation sources are introduced. In 65nm and older (*i.e., larger*) nodes, only one type of variability is generally available to simulate your transistors. As an example, in 28nm nodes, as a designer you will generally have access to several models : local and global variations, which you will have to chose from, based on the characteristics of your circuit. A small circuit (this is defined in the documentation of the technology) could generally need to only use local, while larger circuits, or more distant elements, would need to be simulated with global. **See ? this introduces physical design considerations in the schematic level design already.** This is an important lesson to learn for the future. **We will not use that in this lab though.**

Monte-Carlo simulations can be very CPU intensive and memory demanding. You should always be careful not to overload your system.

- ✓ Uncheck all the previously defined corners except the nominal one.
- ✓ Change the ***Run Mode*** to ***Monte Carlo Sampling,*** and click on the ***Simulation Options*** button (see below).



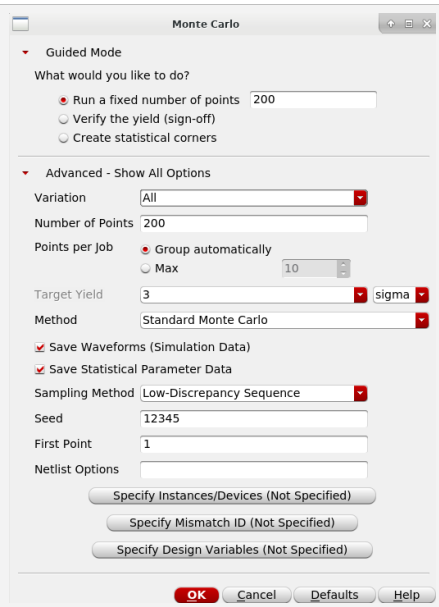
- ✓ Setup the simulation options as in the following figure (on the right) with 200 runs:

Don't forget to tick the two "save" boxes, otherwise it will not plot and save any results.

- ✓ Click OK and run the simulation .
- ✓ Once the simulations are finished, go to the results tab, and select the Yield for the result view.

  ℹ *The **Yield** can be followed in parallel during the simulations. If the yield is clearly low, the simulations can be stopped. This can also be set in the simulation options by **Auto Stop Using Significance Test**.*

  ℹ *You can track the evolution of the simulation from the Run Summary sub window.*



History Item	Status
MonteCarlo.0	running - 7/200 complete

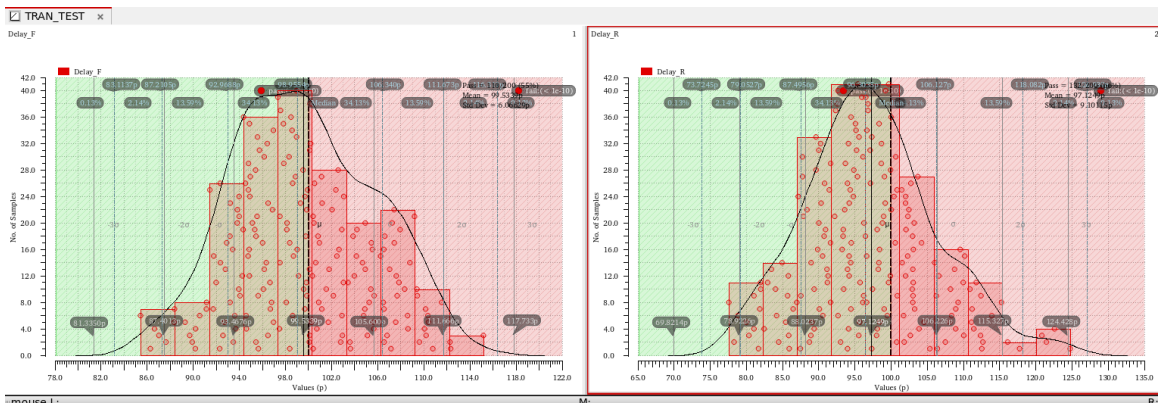
Yield | New Win

Test	Name	Yield	Min	Target	Max	Mean	Std Dev	Cpk	Errors
Yield Estimate: 37.5 %(75 passed/200 pts)			Confidence Level: <not set>		Filter: <not set>				
- TRAN_TEST									
	- Delay_R(summary)	66	77.58p		124.8p	97.12p	9.101p	0.105	0
	Delay_R	66	77.58p	< 100p	124.8p	97.12p	9.101p	0.105	0
	- Delay_F(summary)	55	85.43p		115.2p	99.53p	6.066p	0.0256	0
	Delay_F	55	85.43p	< 100p	115.2p	99.53p	6.066p	0.0256	0

- ✓ Comment on the results of the Monte-Carlo simulations with your colleagues. Note how long are these simulations. In practice, these runs can be heavily parallelized. Though for the class as you are around 60, we limit the amount of simulations you can run in parallel.

ℹ *As the result of Monte-Carlo simulation, joined transient waveforms and the resulting histograms will appear. Histograms allow us to see what number of runs (out of 50 runs: y-axis) were found to be within the specific range (bin) of the delay value (x-axis).*

ℹ *The button allows to access many study parameters. Feel free to explore it.*



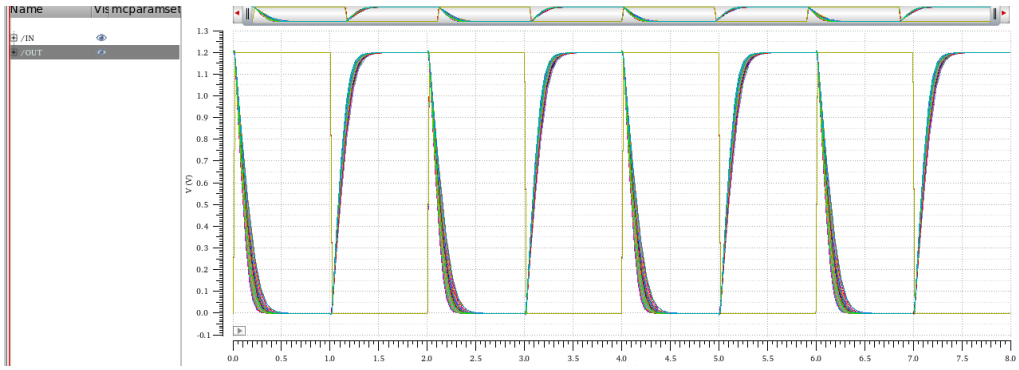


Figure 8 - The Results of the Monte-Carlo Simulations

Note how the transient simulation is not anymore a single line, but a network of curves. The more MC runs are being performed, the more the distribution spreads, and covers all the possible cases. i.e., ensuring that your circuit can work in all conditions.

- ✓ Go to ***Inverter*** schematic and change ***Wn = 280nm*** and ***Wp = 280nm.***
- ✓ ***Check and Save*** the schematic.
- ✓ Run the monte-carlo simulation , and observe the results.

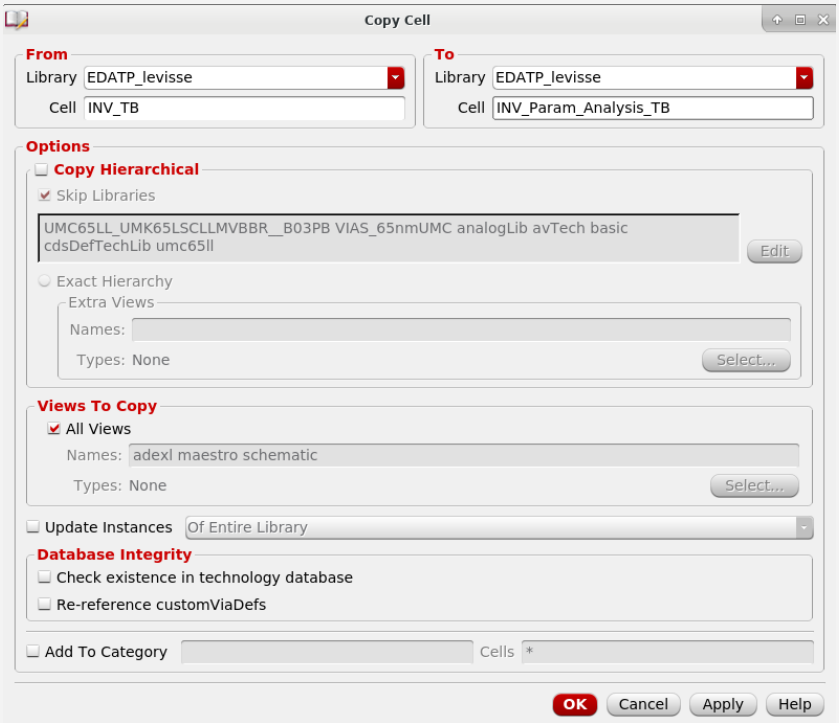
  QUESTION 9-1 : What happened? Which part the circuit is limiting the performances and why? How would you solve it, and which size would you chose ? verify it with a simulation.

10 Parametric analysis or How to explore a design space?

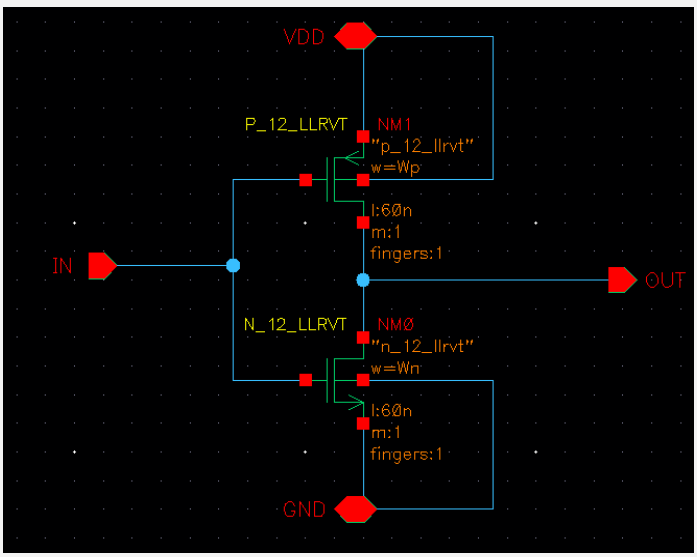
Now comes the most important question in life (okay… in this lab… but still, it is an important question).

How to size a circuit with regard to a sizing constraint ?

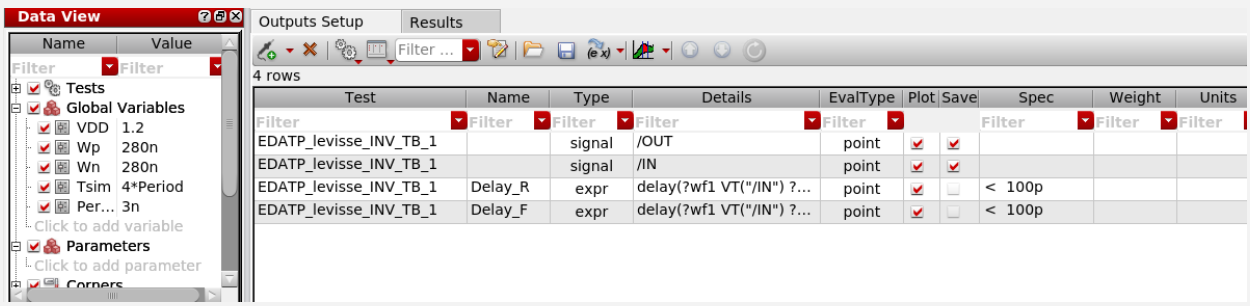
- ✓ Create a new testbench in which you add your Inverter you just designed. (you could also right click on the IV_TB and select copy)
- ✓ Call it INV_Param_Analysis_TB



✓ Make sure that your inverter has its PMOS and NMOS parametrized with Wp and Wn respectively in the schematic.



✓ Your testbench should be configured as before. Only use the Nominal corner. And select *Single Run*.



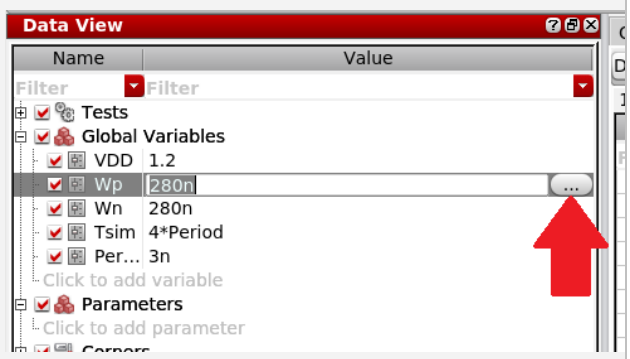
- ✓ Run a simulation. You should get the following values :

Outputs Setup | Results

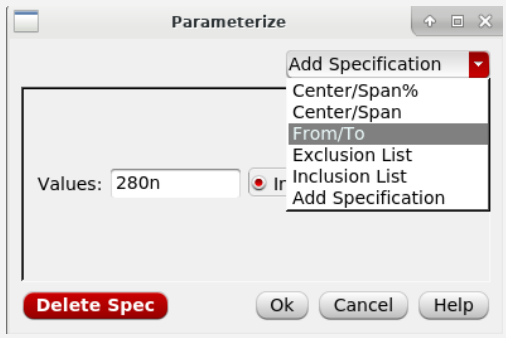
Detail | Filter ... | Replace | (None)

4 rows

Test	Output	Nominal	Spec	Weight	Pass/Fail
Filter	Filter	Filter	Filter	Filter	Filter
EDATP_levi...	/OUT				
EDATP_levi...	/IN				
EDATP_levi...	Delay_R	51.04p	< 100p		pass
EDATP_levi...	Delay_F	91.11p	< 100p		pass



- ✓ As expected, the circuit is unbalanced. The rising time is much shorter than the falling time.
- ✓ What Wp value is necessary to get a 50ps Delay_F value ?
- ✓ In the Data View panel, double click on the value of the Wp Global Variable, and click on the "…"
- ✓ A new window will appear, showing you how to set a parametric analysis.
- ✓ First, click on "Delete Spec" to remove the 280nm point.
- ✓ Then click on the "add specification" drop down menu, and select "From/To"
- ✓ Set the "Step Type" to auto. From 120n to 1.2u with a Total steps of 10
- ✓ This will make the tool run 10 simulations with steps of 100nm from 120nm to 1200nm (1.2um) on the parameter Wp.
- ✓ Click OK.
- ✓ Now, the Wp global variable has a specific syntax. Which corresponds to what you just did.

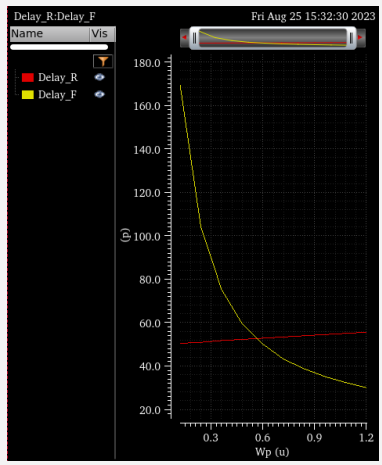


- ✓ Run the Simulation
- ✓ Use the "Detail – Transpose" view to identify which value corresponds to your target (here 50ps).
- ✓ You can plot the simulation waves. When extracting expressions (here delay), the tool plots the extracted expressions versus the input parameters (here Wp).

Detail - Transpose | Filter ... | Replace | (None)

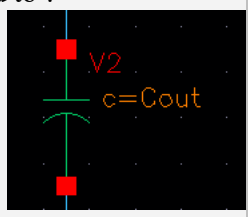
10 rows

Point	Corner	Wp	Pass/Fail	/OUT	/IN	Delay_R	Delay_F
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	nom	120n	fail			50.45p	169.5p
2	nom	240n	near			50.88p	104.3p
3	nom	360n	pass			51.72p	75.3p
4	nom	480n	pass			52.24p	59.42p
5	nom	600n	pass			52.81p	50.06p
6	nom	720n	pass			53.37p	43.13p
7	nom	840n	pass			53.92p	38.69p
8	nom	960n	pass			54.47p	35.1p
9	nom	1.08u	pass			55.01p	32.37p
10	nom	1.2u	pass			55.55p	29.99p



QUESTION 10-1 : how does the Delay_R and Delay_F trend ? and why ? Why do you think the two trends are opposed ?

QUESTION 10-2 : what value to you obtain for Wp ? what does this value correspond to ?

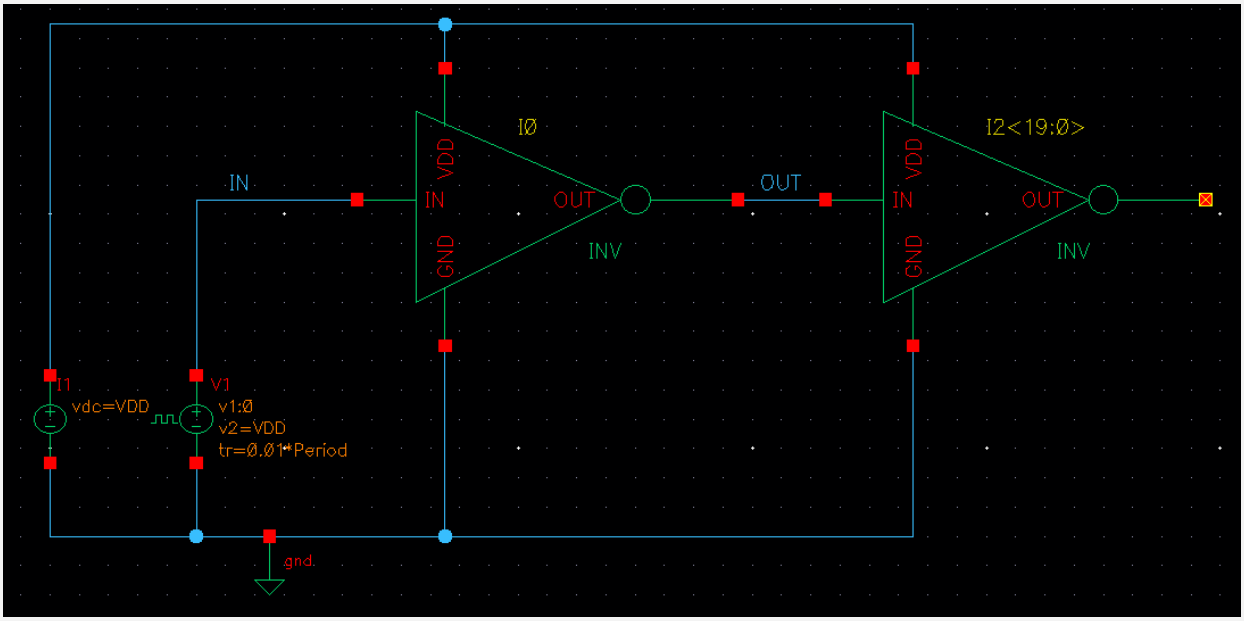


- ✓ Put Wp= 2*Wn on the Global Variables
- ✓ Put a Cout value parameter on the output capacitance in the Testbench. Check and save.
- ✓ Update the design variables in the test, and add a new global variable Cout. Put it at 10f.
- ✓ Define a parametric analysis for it. And make it range from 0.1f to 50fF with 10 automatic steps.
- ✓ Analyze the table and waveform results

QUESTION 10-3 : What maximum output capacitive load is this gate able to handle for a transition time of 100ps ? what parameter can you play with, if you have to meet a 100ps for a 50fF output capacitive load ?

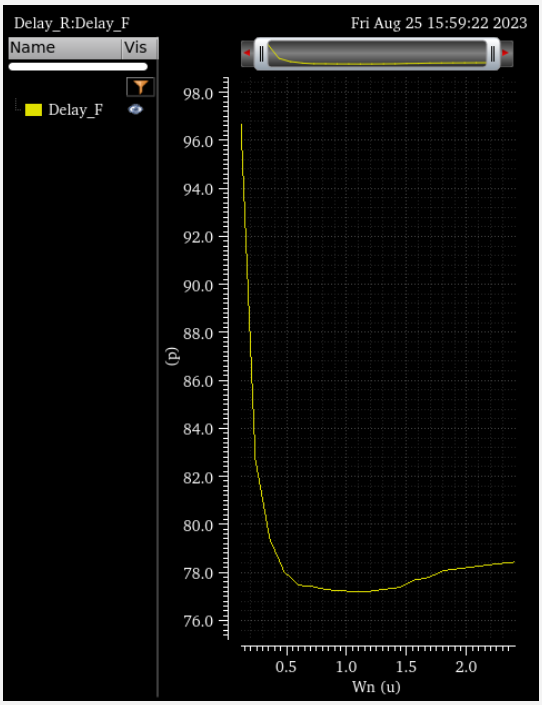
What about more complex situations ?

- ✓ Modify the testbench and replace the capacitor by an inverter of the same type as the INV gate.
- ✓ Place it carefully to make sure that the OUT signal is still in between the INV I0 and INV I2 (these names are based on the screenshot but you may use other names)
- ✓ Replicate the output inverter 20 times by putting I2<19:0>



- ✓ Check and save will issue a warning about the output pin of I2. This is normal and you can ignore it.
- ✓ Run a parametric analysis on Wn. From 120nm to 2.4um with 20 points. Keep Wp = 2*Wn.
- ✓ And check the delay between OUT and IN nets.
- ✓ Run the simulation and analyze the results.

QUESTION 10-4: Focus on Delay_F. Why such a trend ? Why does the output increase again after a point ?



11 GOOD PRACTICES

11.1 BEING MINDFUL ABOUT YOUR RESOURCES UTILIZATION

As an engineer, you will NEVER work alone.

- The servers you use will almost ALWAYS be shared with other users
- The disk space you use will almost ALWAYS be shared with other users
- Filling the disks will impact your work, but also the work of your colleagues
- The software you use are being paid for, by your structure, and have a limited amount of license tokens. Using all the tokens will block other users from getting them.

It is YOUR duty to monitor your resources utilization, and generally not over-use resources which you have access to. Or if you need to use more resources, generally make sure your supervisors or IT department are aware of your needs.

QUESTION 11-1: Summary the previous section in a few sentences **with your own words**.

11.2 ERASING SIMULATION FILES

As the result of every simulation, a large amount of files is typically created. These files store simulation info and save simulation data, such as schematic netlists, variable definition file, simulation report files, output log files which are used to save voltage and current waveforms, and many others. In the case of transient simulation, output log files can become very large (several GB) if sufficiently long simulations are performed on a relatively complex circuit. If special care is not taken, these files can easily fill the disk space of your system, causing the simulations to crash and making the system unstable. Therefore, simulation output files should be handled carefully, keeping in mind their size and location at all time.

- ✓ By default, starting from the **EE429_FULLCUSTOM** the path to the directory where the simulation data is saved is: **./simulation**
- ✓ Go back to your linux terminal. If you are not already in your Virtuoso project directory (**EE429_FULLCUSTOM**), enter it:

  > **cd ~/EE429_FULLCUSTOM/** ↵
- ✓ Explore your project directory by typing:

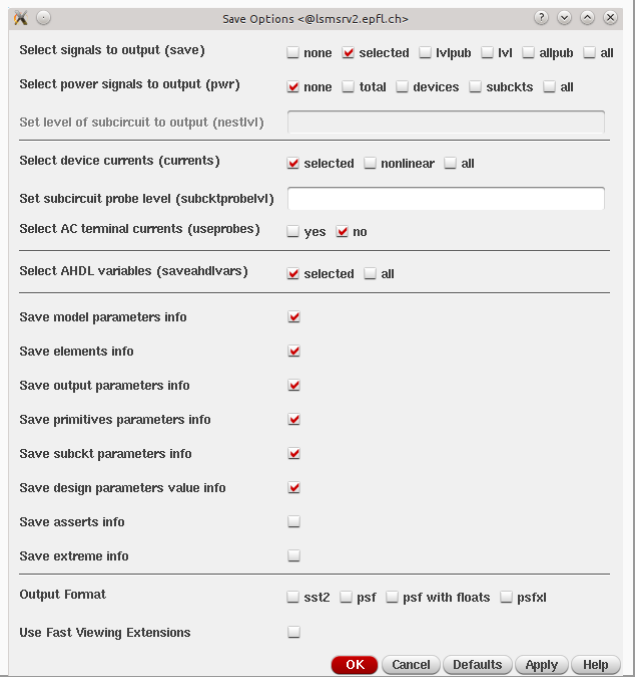
  EE429_FULLCUSTOM > **ls** ↵
- ✓ Notice several configuration, library and log files. You will also find a folder named ***simulation***, where all the simulation data is stored. Check the size of the ***simulation*** directory by typing:

  EE429_FULLCUSTOM > **du -sh simulation/** ↵
- ✓ Check the size (write it down):

  EE429_FULLCUSTOM > **du -sh simulation/** ↵

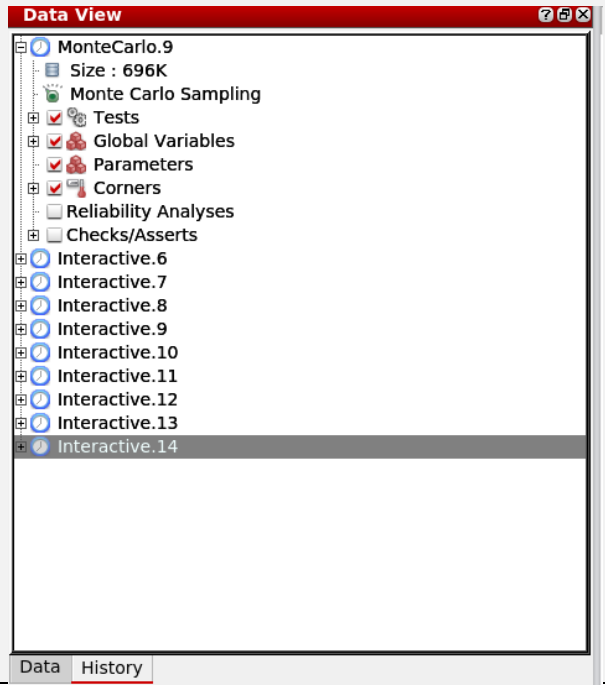
*It is also possible to remove the complete context of the **simulation** directory if no information about the previous simulation runs is no longer needed. This can simply be done by typing: **rm -rf simulation/***

- ✓ In order to reduce the size of the generated files, it is possible to specify that only the signals selected to be plotted will be the ones to be saved. This way, we can avoid storing unnecessary data on our disk space. In the ***Data View*** window right-click on the specific test (**Tran_Test**) and open the ***Open Test Editor***. Select ***Outputs → Save all…***
- ✓ Here, it is possible to specify the desired properties as we did in the figure (right).



QUESTION 11-2: how much simulation space do you use ? In your EE429_FULLCUSTOM folder use the "ncdu Simulation" command. You can navigate with the keyboard. Right arrow to enter and left arrow to come back one level up. Which cellview take the most space ?

- ✓ As an alternative to the previous solutions. Maestro views allow you to manage disk space directly inside virtuoso.
- ✓ This solution is not global, as it can only be handled per maestro views. So if you used 5 maestro views, you would need to clean the 5 of them.
- ✓ In the data view, click on the History panel on the bottom. Each simulation you did run is visible here, and can be cleaned off.
- ✓ Once you are done and extracted all the results you need, delete all the result history from this ade assembler view. Select them all (sift + click), right click and select Delete.
- ✓ Compare the results of the ncdu command after that.



11.3 REMOVE LOCK FILES ON VIRTUOSO

Cadence Virtuoso was originally designed as a collaborative tool. Several users could work in parallel on the same library at the same time. Though, we generally advise not to work on the same cellviews.

In that sense, if a cellview is opened, virtuoso will create something called a lockfile. When a view is locked by virtuoso, it cannot be accessed by any other user.

If you open a view, and, for some reason, do not properly close virtuoso, or change server, or if you are unlucky (the tools sometimes crash). You sometimes cannot reopen your cellviews, with an error message about lock files.

Two solutions are possible :

1- You could manually erase the lockfiles, which are generally located inside the design library with the .cdslck extension. **This solution is dangerous as it involves using a rm command, and you may delete the wrong files.**
2- Use the cadence tool clsAdminTool. **Advised solution.**

```
> cds clsAdminTool
> ale yourlib
```

This command will **list** the lockfiles in the lib called "yourlib" (update it accordingly)

```
> are yourlib
```

This command will **remove** the lockfiles in the lib called "yourlib" (update it accordingly)

11.4 CHANGING SIMULATION DIRECTORY (OPTIONAL)

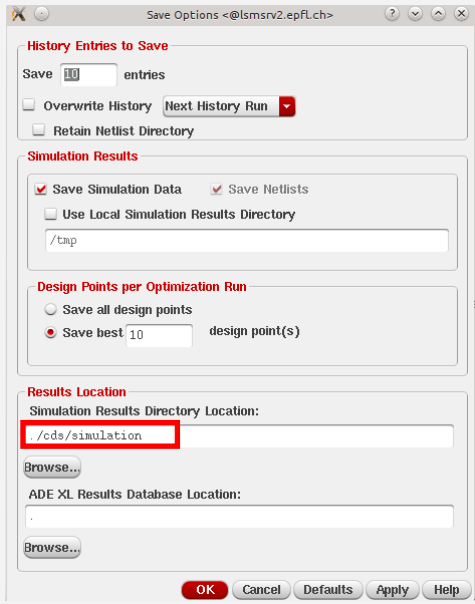
It is also possible to change the location at which the output files are generated. This is especially useful if you are running simulations on servers or if local machine has multiple hard drives and/or partitions.

- ✓ In Analog Design Environment (ADE Assembler) window, select ***Options→Save...***
- ✓ As you can see, the default directory is set to ***./simulation***. Depending on the design-kit (different technology options), the default directory may be different.
- ✓ Create a new directory within your project directory. You can name it as you want, we propose ***simulationOut***:

```
EE429_FULLCUSTOM > mkdir simulationOut ↵
```

- ✓ In your Analog Design Environment window, select ***Options→ Save...*** and edit the project directory path towards your newly created directory (type: ***./simulationOut*** instead of the default: ***./simulation***).
- ✓ Run the transient simulation (corner or Monte-Carlo) once again, and check where is the output simulation data.

  Note that our manipulation only changed the directory within the Virtuoso project directory (EE429_FULLCUSTOM) where the data is going to be saved. Of course, a completely different path can be specified as well, such as a path to a directory on another hard-drive or on a specific server.



12 EXERCISE ON DC ANALYSIS

Now that you know how to create cell views, create testbenches and run monte-carlo analysis, we propose you a practical guided exercise to make sure you understood correctly everything. The moodle quiz will have questions associated with this exercise.

Task 1 : build an inverter testbench with a 10fF load on the output, a NMOS w of 120nm. Apply a ratio of 2 on the PMOS. Take a 1.2V VDD.

Hint : You could reuse one inverter you did before and just make a new testbench.

Task 2 : run a DC analysis and plot the Vout versus Vin graph as you did earlier.

Task 3 : From the VTC. Identify the VM (the threshold of the inverter) , where Vin = Vout.

QUESTION 12-1: Does the inverter seem balanced ?

Task 4 : plot the derivative of the Vout(Vin) curve

Using the calculator, click on

, then select the waveform of the output voltage from the waveform viewer. This will automatically print the name of the waveform. Then, apply the function "deriv" to it.

Hint :Once the wave object has appeared in the textbox (should be something like "leafValue(VS("/OUT"))") , click once on "deriv" in the functional panel. It will update the text (you could also simply type the name of the function there). Then, click on *to add the function output to your ADE view.*

Task 5 : From the derivative, identify the noise margin of the inverter.

Hint : you will find two different values for the noise margin, which one makes the most sense ?

Task 6 : add the FS, SF, FF and SS corners. And run the simulations again.

QUESTION 12-2: How does VM and the noise margin change with the different corners and why ?

Hint : if you have the derivative function already inside your ADE view, you do not need to do it again through the calculator. It should plot it automatically.

Task 7 : setup a monte-carlo simulation, and make 200 runs (this may take some time, you can go take a coffee break).

QUESTION 12-3 : how does the VM and the noise margin change after the monte carlo simulation?

At this end of this exercise once you are sure that you understood the results. Go through the 11.1 and 11.2 sections of this document, and clean the simulation files generated during the exercise.