RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Mixed Mode Simulation

P. Fischer

Lehrstuhl für Schaltungstechnik und Simulation
ZITI, Uni Heidelberg

(Based on slides from Florian Erdinger)

Why Simulate in Mixed Mode?

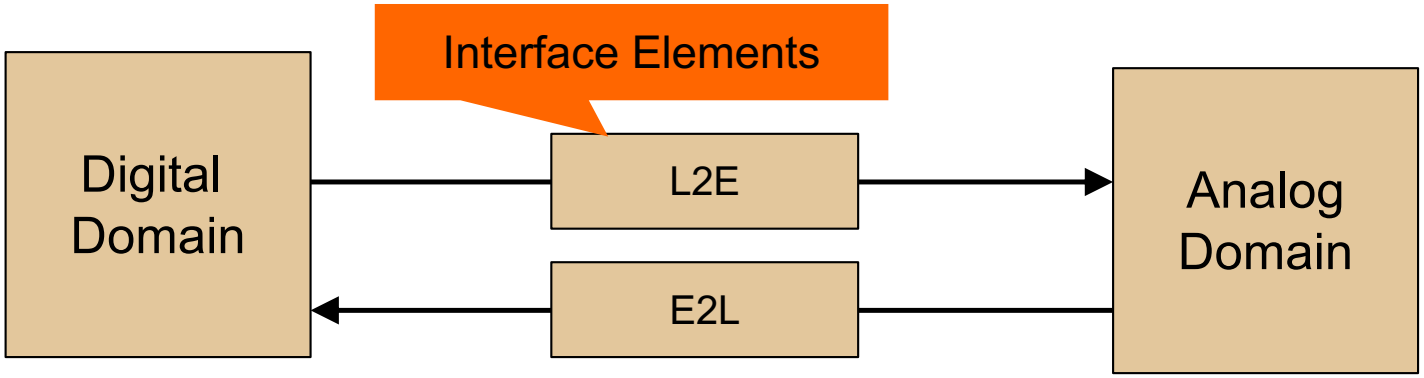
- Most analog circuits need interaction with digital circuits
  - control logic to steer the analogue part
  - processing / verification of results
- Simple digital functionality can be obtained by spice sources (vpulse, **vpwl**,…), but this is tedious, inflexible,…
- (More flexibility by using Verilog-A. Good for simple extensions (DAC..), but not suited for large digital parts)

→ **Mixed Mode Simulation:**

- Digital parts:
  - Hardware Description Language (Verilog, VHDL) – very flexible
  - Digital simulator
- Analog parts:
  - Schematics
  - Analog simulator

Mixed Mode Simulation

- Two simulators run in parallel
  - Digital Sim. for digital part, Analogue sim. for analogue part
  - *Interface Elements* translate between domains (what is 'True'?)
- Time must be (internally) synchronized between the sim.



- Advantages:
  - Complex steering / logic easy to implement
  - **Much** faster simulation in large designs (once it runs…)
- Drawbacks:
  - More complex.
  - Long simulator startup.

Note

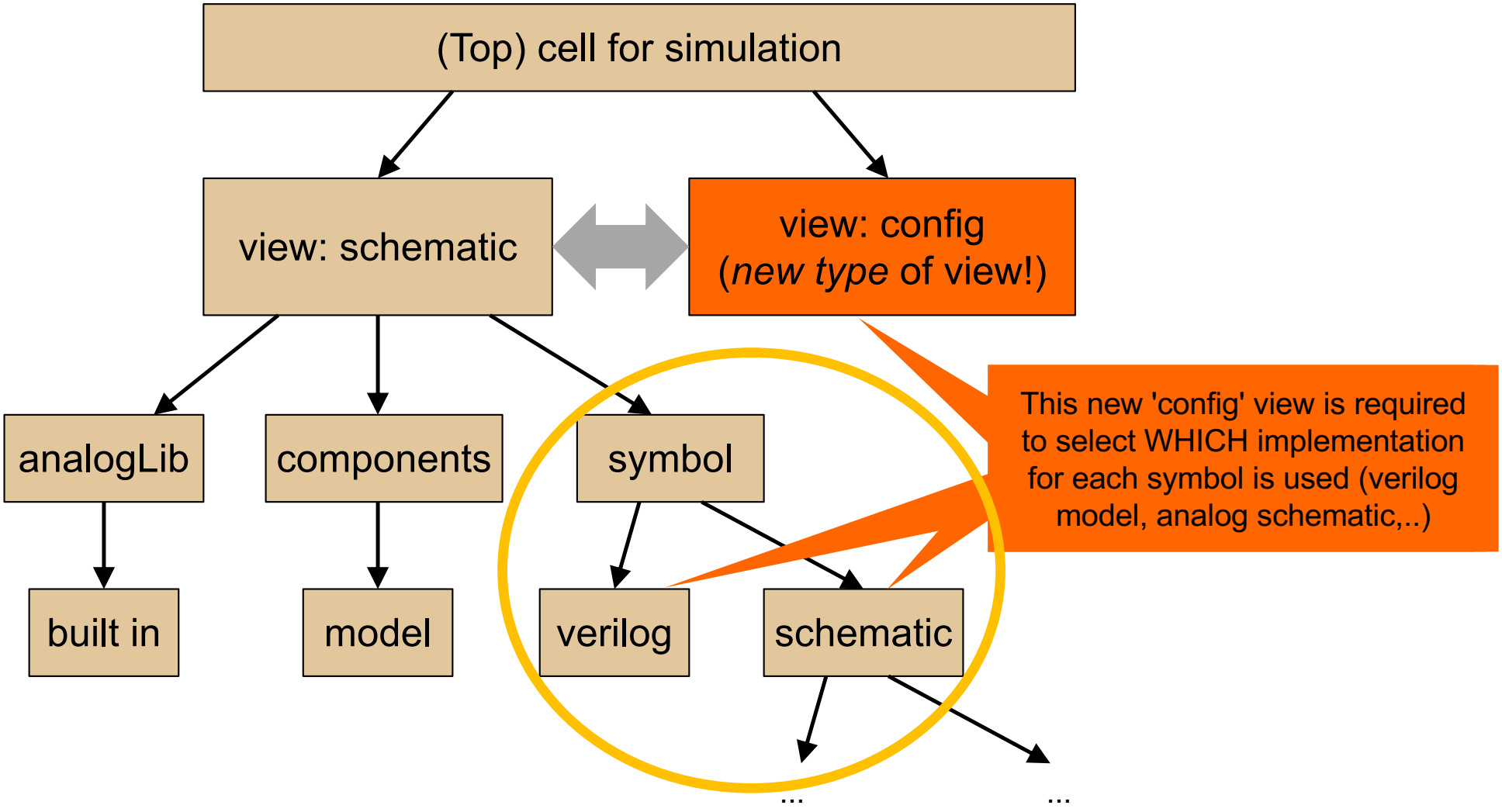
- There are many ways to do a Mixed Mode Simulation.
- I show here just one solution, which is easy to use for a start, but not so well suited for larger designs

- For large designs, it is efficient to
  - first produce all 'netlists'
  - work 'on the shell' to put things together and set up and run the simulation
  - Extract results with a graphical viewer of automatically

- Jargon: 'Digital on Top' vs. 'Analog on Top'

These slides

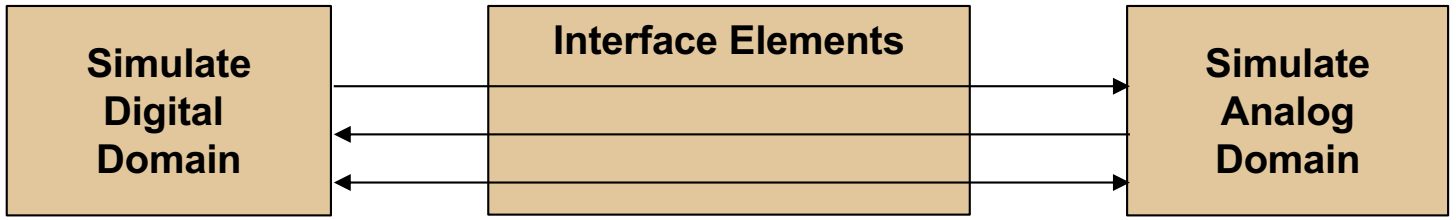
What do we Need?



A SIMPLE EXAMPLE

Workflow

- The following slides show how to set up a simple Mixed Mode Simulation in the *Virtuoso ADE* environment. Steps:
  1. Creating a *Verilog module* for stimulus with a matching *symbol*
  2. Creating a *top-level simulation schematic* instantiating the Verilog symbol and some analog circuit connected to it
  3. Creating a *'config'* view of the top-level simulation schematic
  4. Edit the config view telling the simulation what to do
  5. Simulate using the AMS simulator



Note: The interface elements are created more or less automatically…

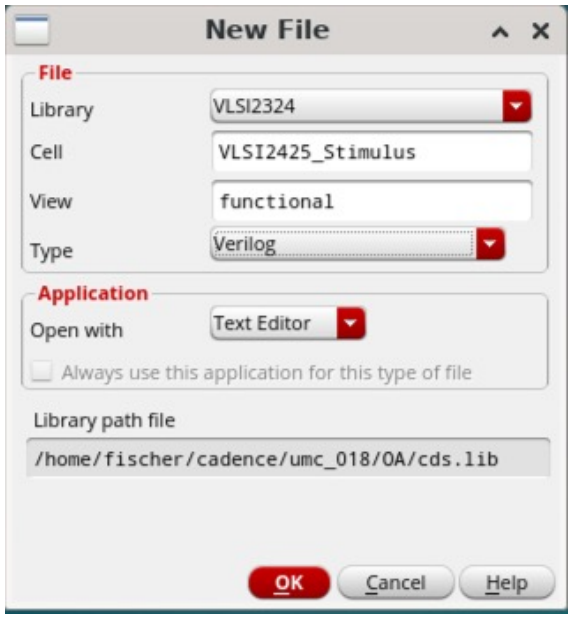
Before You Start (needed only once)

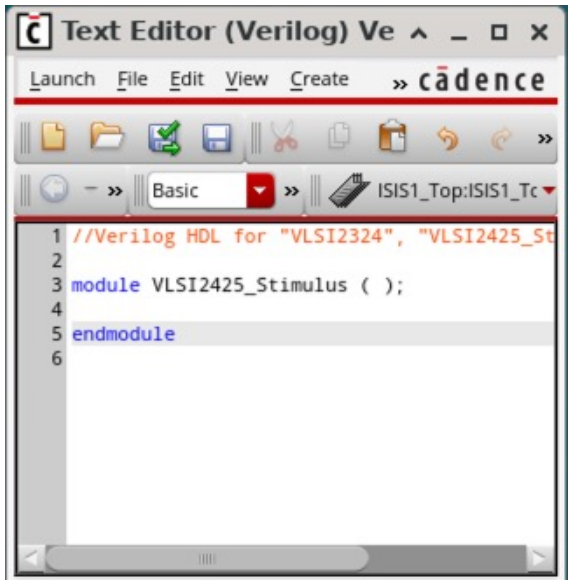
- We need to make the ‘connectlib’ available.
- Check your **`cds.lib`** or see in the library manager

- Make sure you have the line
  `SOFTINCLUDE $IUSDIR/tools/inca/files/cds.lib`

- If this does not work, add it explicitely by adding
  `SOFTINCLUDE /net/eda/INCISIV102/tools.lnx86/affirma_ams/etc/connect_lib/cds.lib`

1a. Creating a New Verilog Module



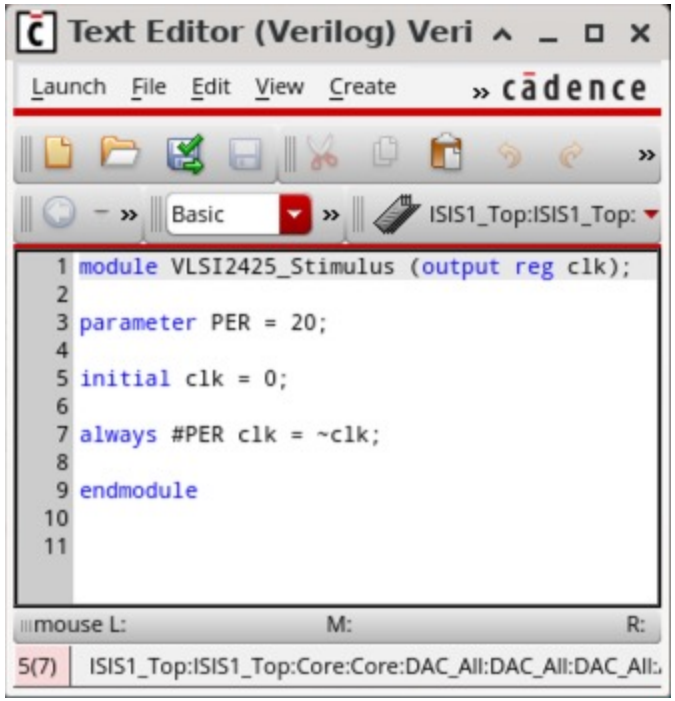


- We will use this module to generate a rectangular signal to stimulate our analogue circuit.
- In 'Library Manager:
  - File → New → Cell View
  - 'Cell': name of verilog module
  - 'Type': Verilog
  - 'View': will change to 'functional'
- The Cadence text editor opens with an empty Verilog module

1b. Fill the Verilog Module

- Fill the Verilog module with some code.
  - The code does ***not*** have to be synthesizable
- You may use parameters.
- For instance

  (1 time step = 1 ns by default)



- When you close the text file, it is automatically parsed. This takes a moment.
- Correct it until there are no errors left.
- In my editor, the log file can be seen with **`View->Parser Log`**
- Error messages of Verilog compilation end up in **`.cadence/dfII/TextSupport/Logs/Parser/verilog`**...
- Use the latest file there

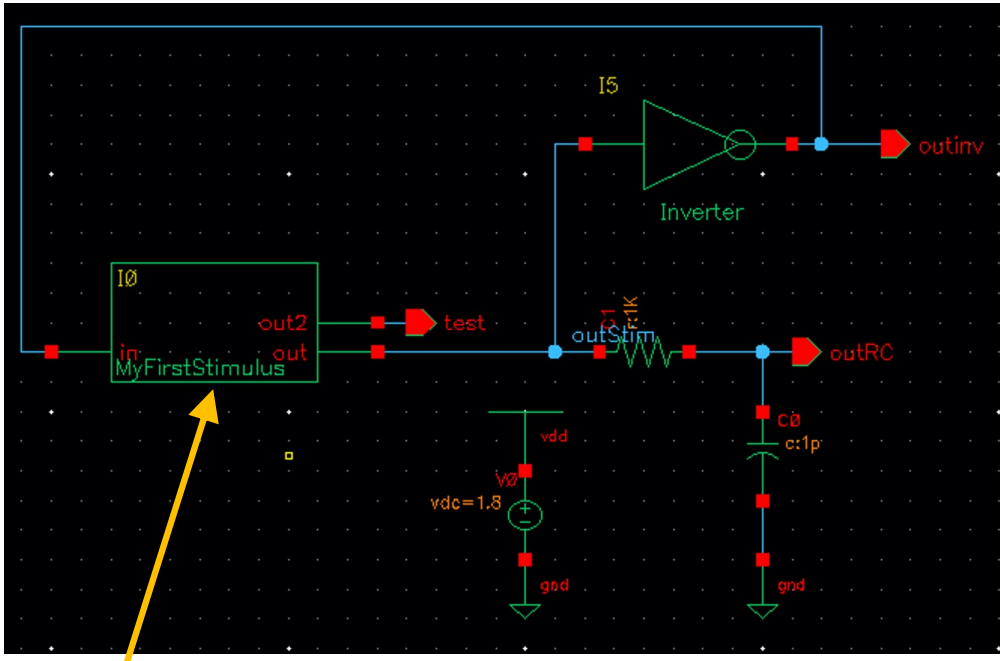
1c. Create the Symbol

- When the Verilog file is closed, the tool offers to create a symbol if there is none (or to modify an existing symbol which does not fit to the declared interface).
- Create the symbol.

- You may then edit the symbol to make it nicer.

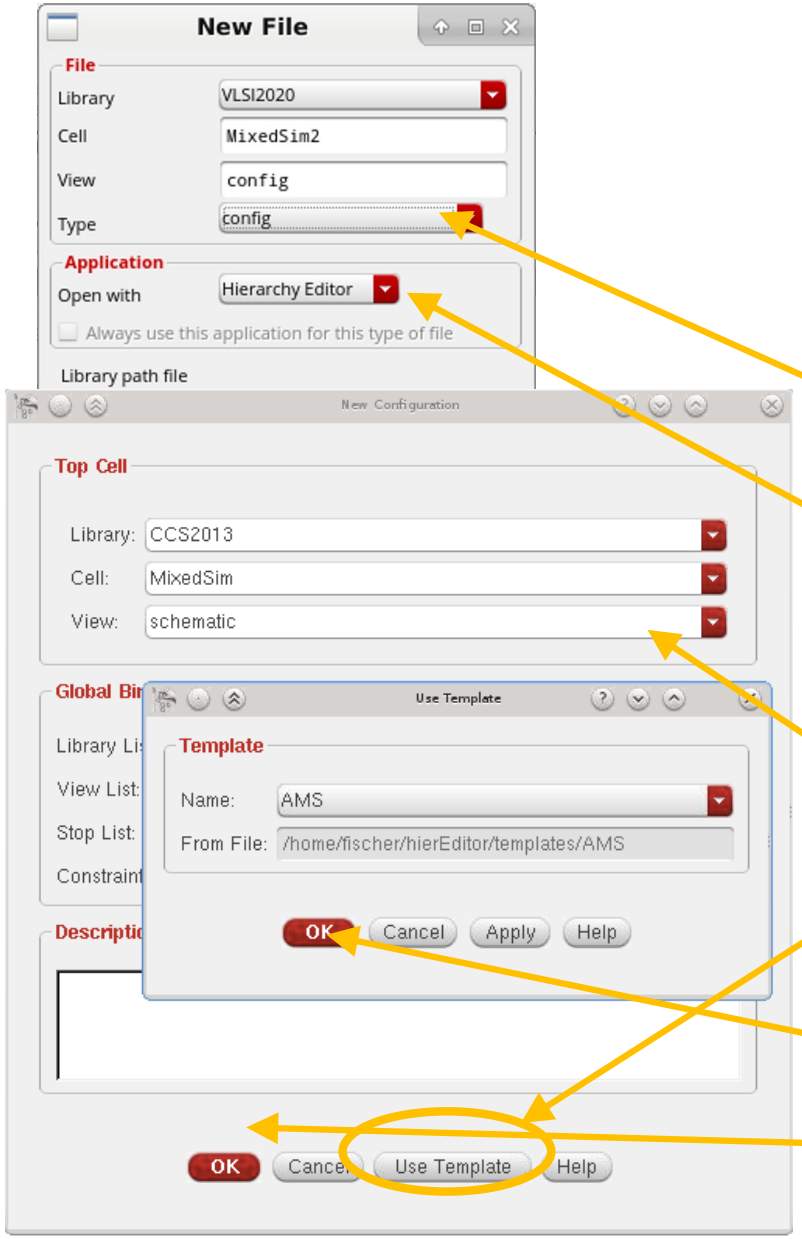
2. Creating A Top-Level Simulation Module

- In 'Library Manager'
  - File → New → Cell View
  - Create a schematic



- Put an instance of your Verilog module, i.e. the symbol
- If the Verilog contains *parameter*s, the symbol inherits them.
  - To see them: In the instantiated symbol, select CDF Parameter of view -> functional (not 'Use Tool Filter')
- Add some analog circuit (symbols, primitives, sources, ...) which can also be in symbols.

3. Creating the Simulation Configuration View



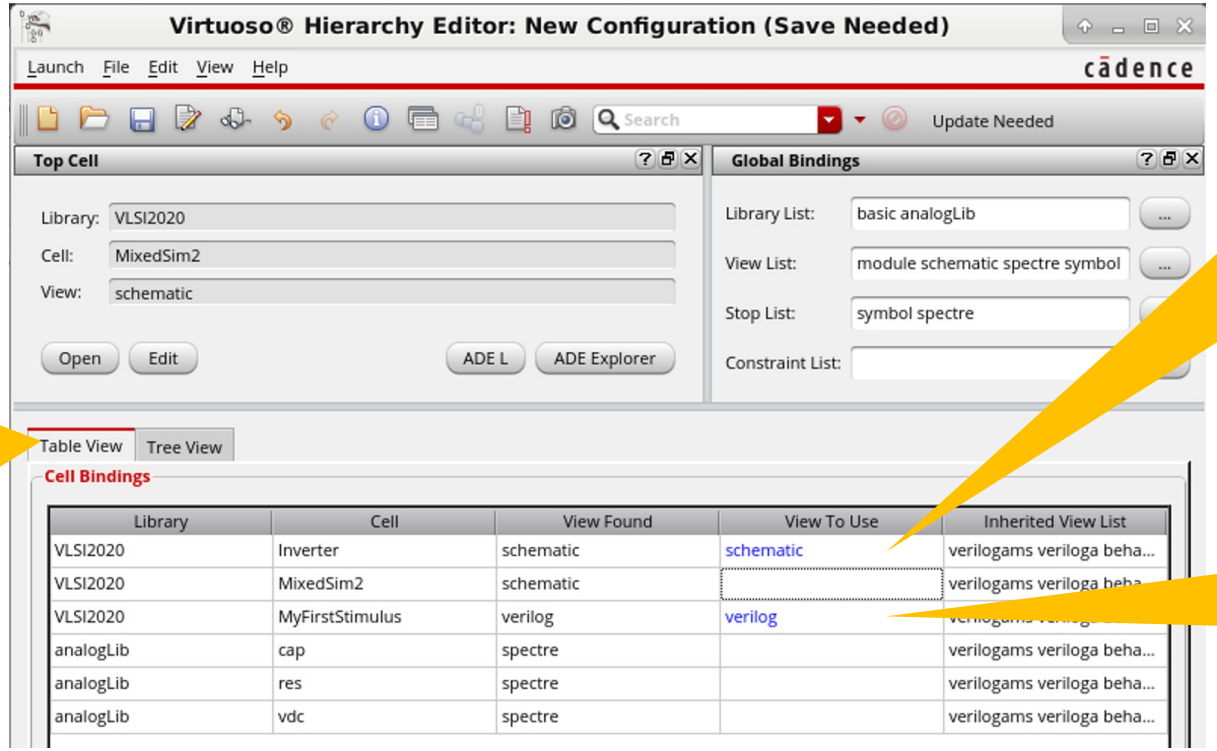
- The AMS simulator needs a 'config' view for the *simulation schematic*
- In 'Library Manager:
  - Select your simulation schematic
  - File → New → Cell View
  - 'Type': config
    (name changes to 'config')
- Note that ‘Application’ switches automatically to 'Hierarchy Editor'
- In the next window: change 'View' to 'schematic’
- Click 'Use Template’ (bottom)
  - Select 'AMS’ (this will be our simulator)
  - OK
- OK

4. Changing 'config' view with the Hierarchy Editor

- The **config view** is edited in the **'Hierarchy Editor'** and configures the netlisting procedure for simulation.
- **Cells** can have **multiple representations**, for instance a 'verilog' view and a 'schematic' view at the same time.
- The config view specifies the view to use for netlisting for each cell (or even instance)

'Table view' lists cells per type.

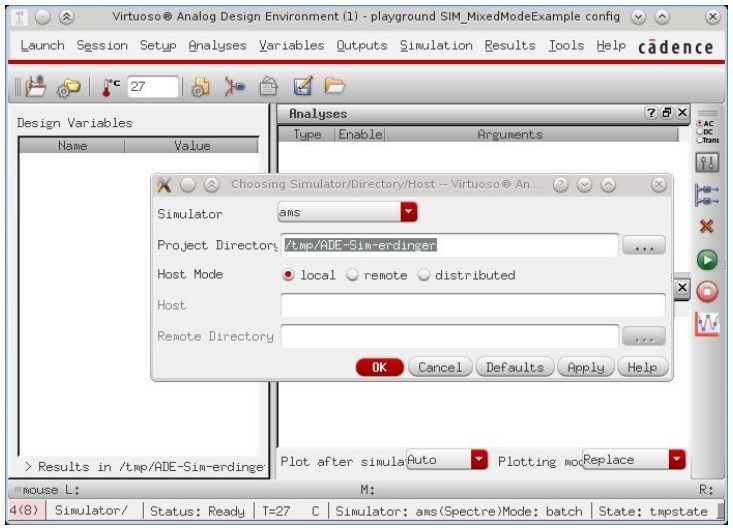
'Tree view' shows all instances

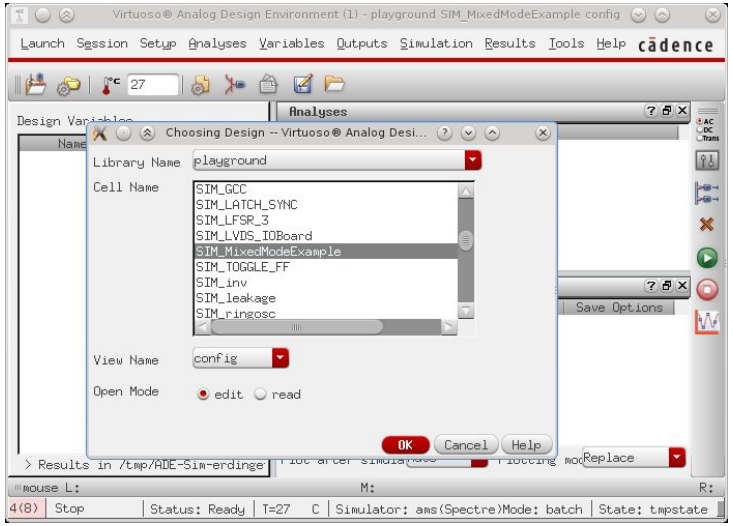


A cell can have several view, e.g. 'verilog', 'functional' or 'schematic'. The view to use is specified **here**

Right Click to select (now 'functional')

5a. Setting Up the Simulation and Outputs





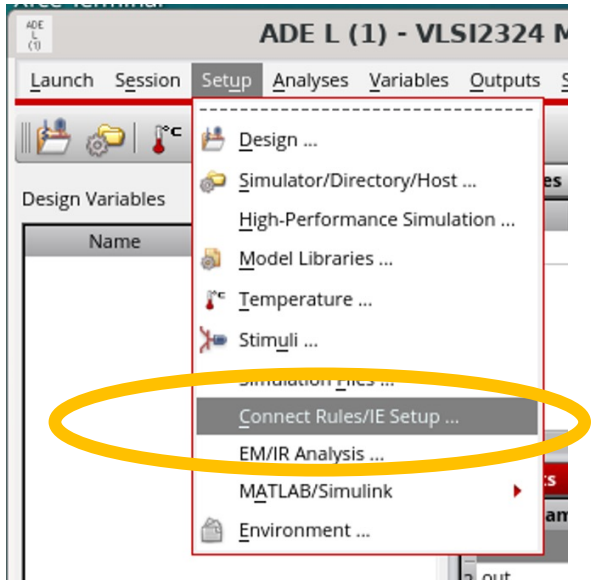
- Open the top level *simulation* schematic
- In menu: Launch → ADE L (or higher)
- Setup → Design
  - Change 'View Name' to 'config' (which we have created before)
- Setup → Simulator/Directory/...
  - Change 'Simulator' to 'ams' this takes a moment…
- Add a transient simulation

5b. Choosing Information to Store

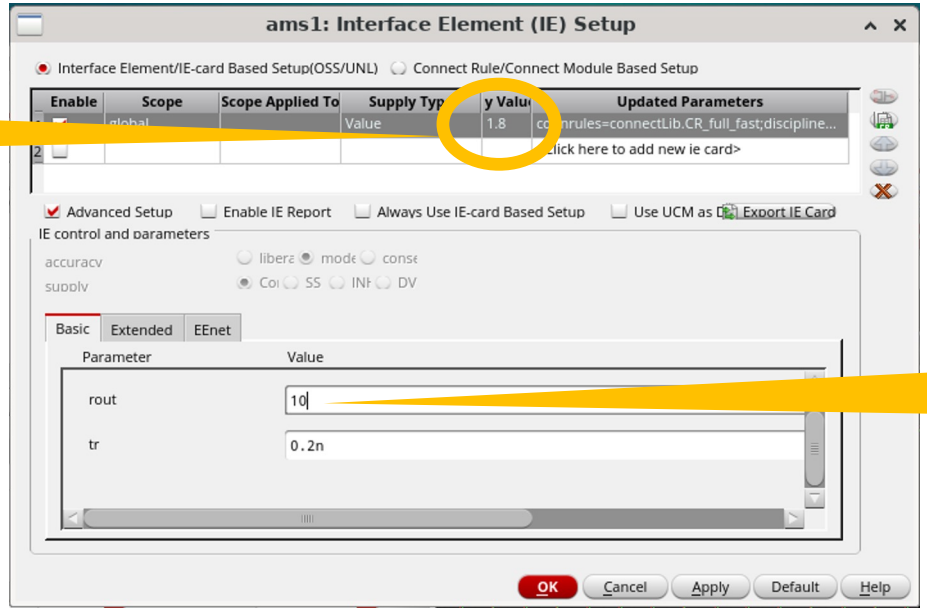
- AMS saves only selected nets by default.
  - In small designs, it is more convenient to save all, so that you do not have to re-run the simulation if you want to view more signals
- In Simulation window, go to 'Outputs → Save All…'
- In the category NETS, select 'all' to save all node voltages
  - If you want to also save nets INSIDE of modules / instances, select Levels -> 'all'
- If you want to look at currents:
  - In the category CURRENTS, select 'all'
  - Do not do this all the time, because the larger choice of signals makes it more confusing later to select the right ones…

(Connect Rules – defaults should be ok…)

- This is a bit tricky. The mechanism has changed depending on software version.
- At the moment, you can access the connect rules from Setup->Connect Rules
- In the window, go to ‘advanced setup’.
  - Make sure 1.8 V are set!
  - The simple model assumes a series resistor and a rise time.



Must have 1.8 V here!



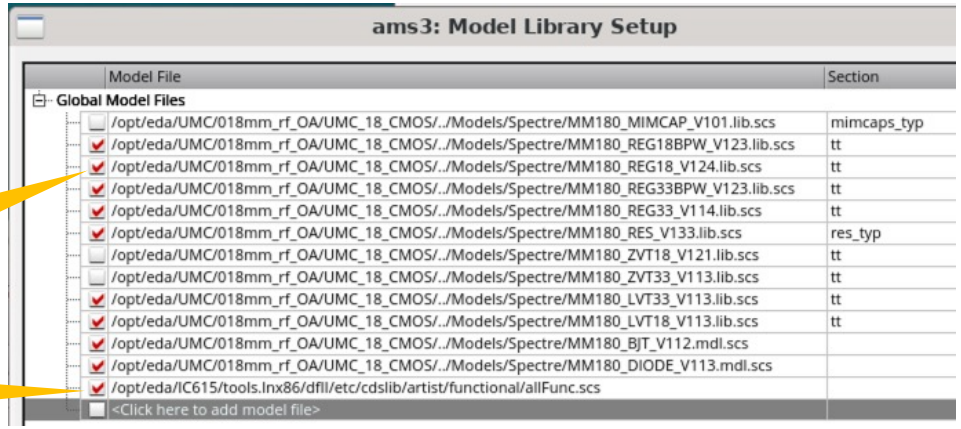
Series R and rise time

5c. Setting up The Library Path

- If you want to use active devices, like transistors, the models must be defined in Setup->Model Libraries
  - Check that they are set!
  - Note: You may have to pick a 'section' (right column)

We mainly need MM180_REG18_...scs

Must have IC618 here! Right click on line!



- These files are defined in your .cdsinit.
- If the are wrong (because UMC or cadence versions have changed), update them there and re-run .cdsinit (load ".cdsinit")

5d: Running and Viewing the Simulation

- You may want to save the state before you run…

- Run the simulation (‘play button’)
- In the log file you can see that there are several steps:
  - Compilation
  - Elaboration
  - Simulation

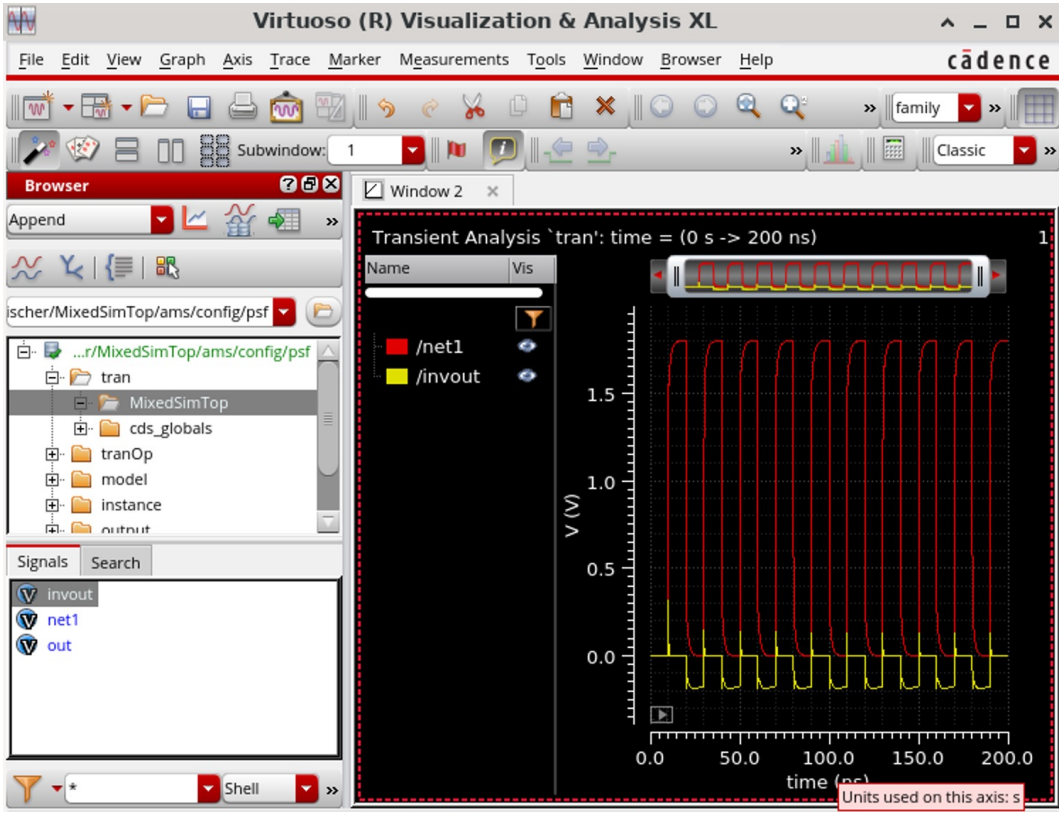
- Errors are in Simulation->Output Log->..

- Verilog **$display** task prints to the log file
- Open the results browser to look at the results:
  in the ADE menu: Tools → Results Browser …

Selecting Waveforms

- To select digital waveforms, open the ‘results browser’
  - for instance from the schematic window (next to ‘calculator icon’)
  - Or from the simulator→tools→ ..
  - → You get a pane with 3 entries (calc., res. browser, results)

- Select Sim->tran->Top
  - In Verilog Instance, you can select internal variables.

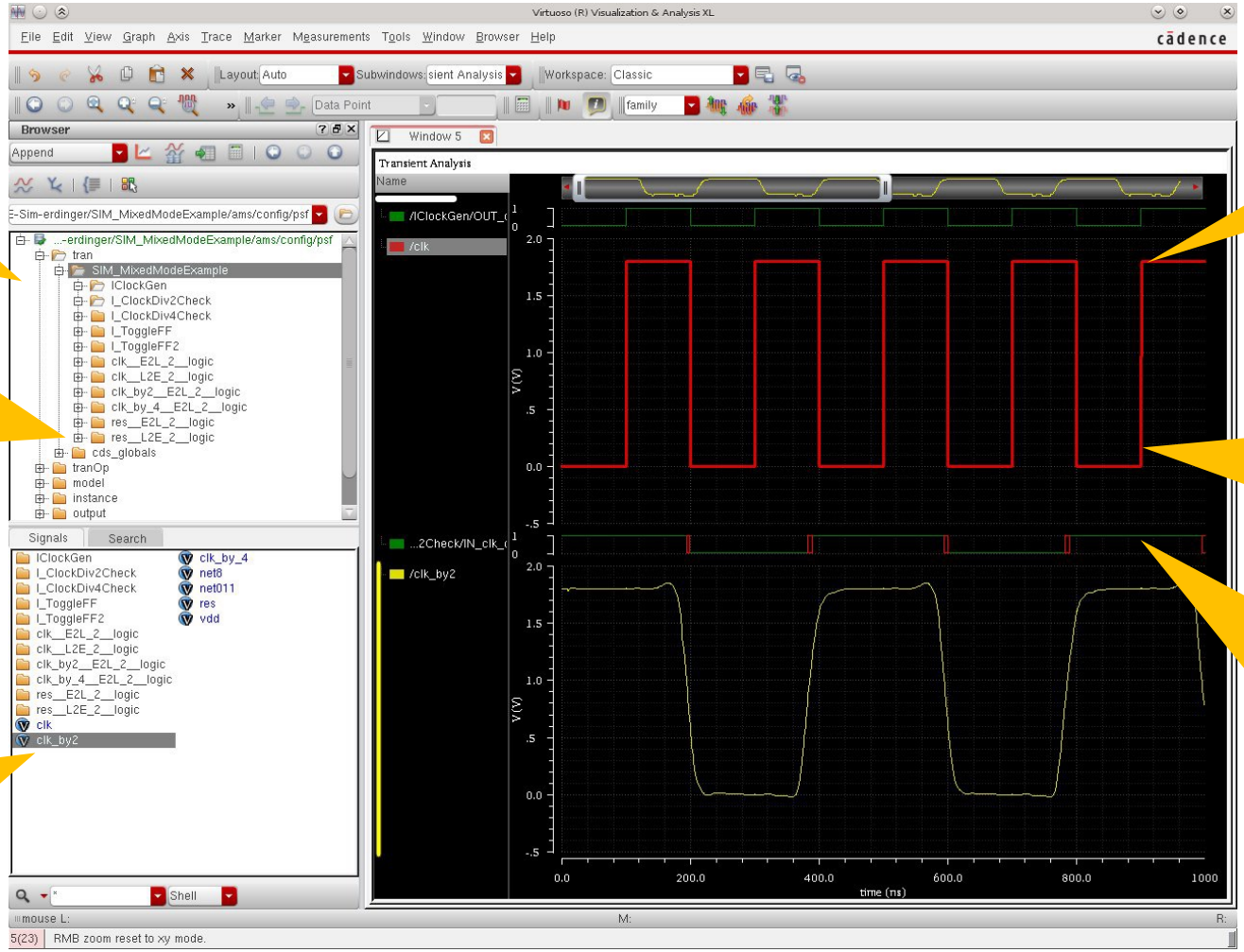


The Results Browser

Browse the design here

E2L → Electrical to logical interface element

Plot waveforms from here



Digital Waveform

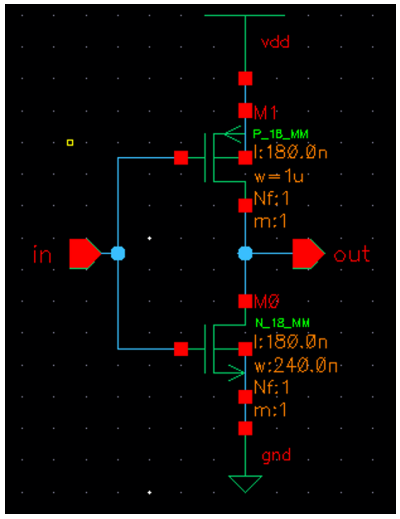
Translated to an analog by an L2E (Logical to Electrical IE)

An analog waveform translated to a digital is temporarily undefined during transistions

Adding Functional Models

- You can add a functional (e.g. Verilog) model to a schematic, for instance to a gate
- You can start from the schematic and then Create->Cellview->From Cellview -> Verilog-Editor

- Add the Verilog Code to describe the behavior. Note: This code is *not* compared to the real behavior of the schematic. You can get it wrong!!




```
1 //Verilog HDL for "VLSI2020", "Inverter" "functional"
2
3 module Inverter ( output out, input in );
4   assign out = ~in;
5 endmodule
6
```

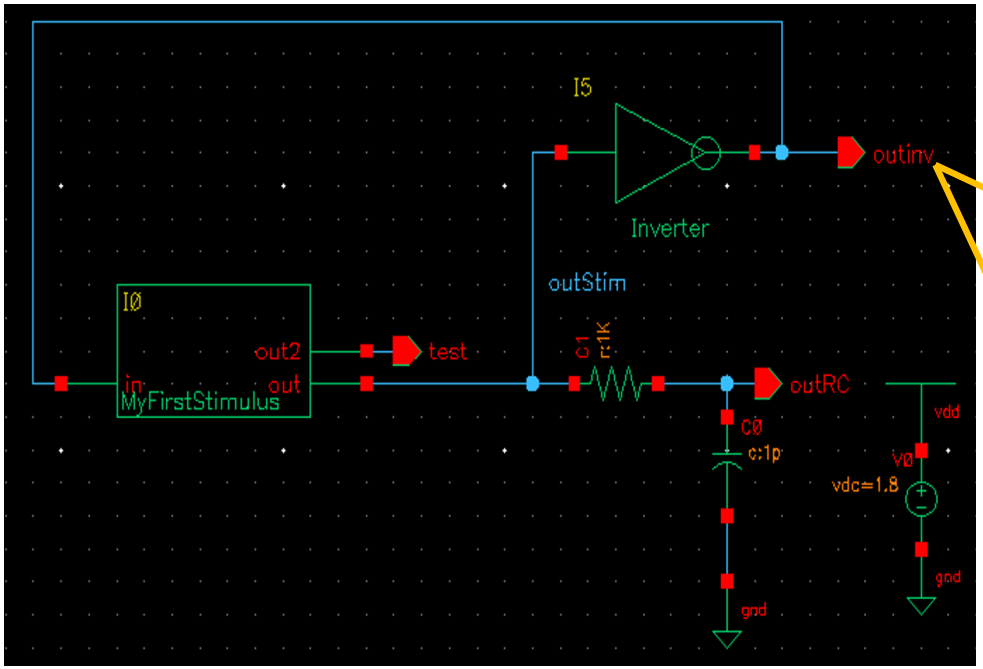
Sample Output

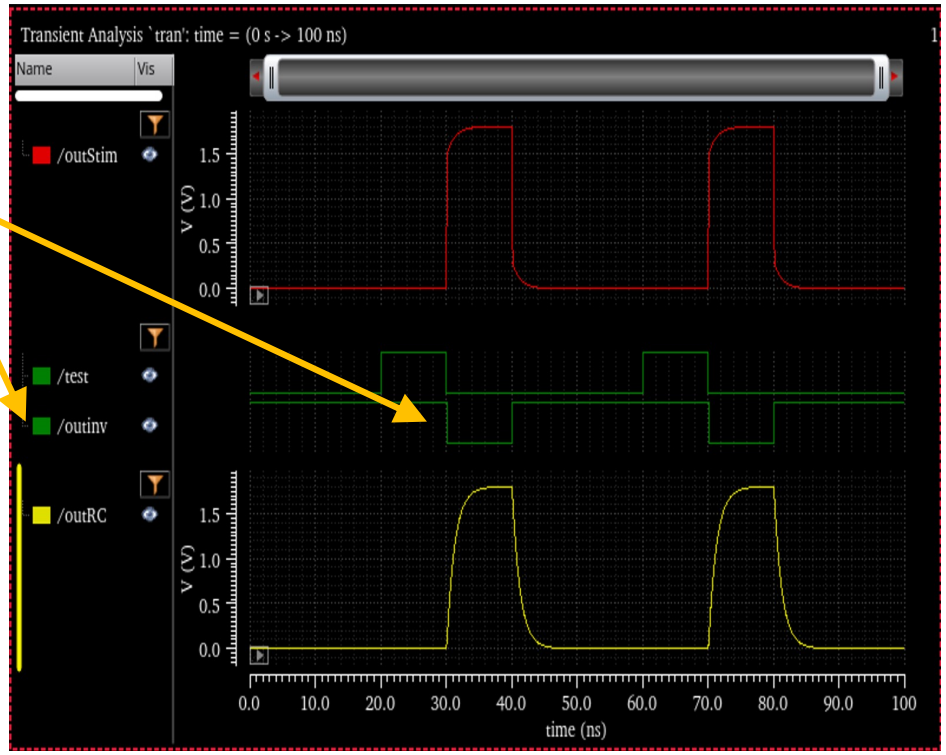
- Here, The inverter is simulated with its ‚***functional***' model:

Library	Cell	View Found	View To Use
VLSI2020	Inverter	functional	functional
VLSI2020	MixedSim_Top	schematic	
VLSI2020	MyFirstStimulus	verilog	verilog

```
module Inverter ( output out, input in );
  assign out = ~in;
endmodule
```

- 'outinv' is a *digital* signal



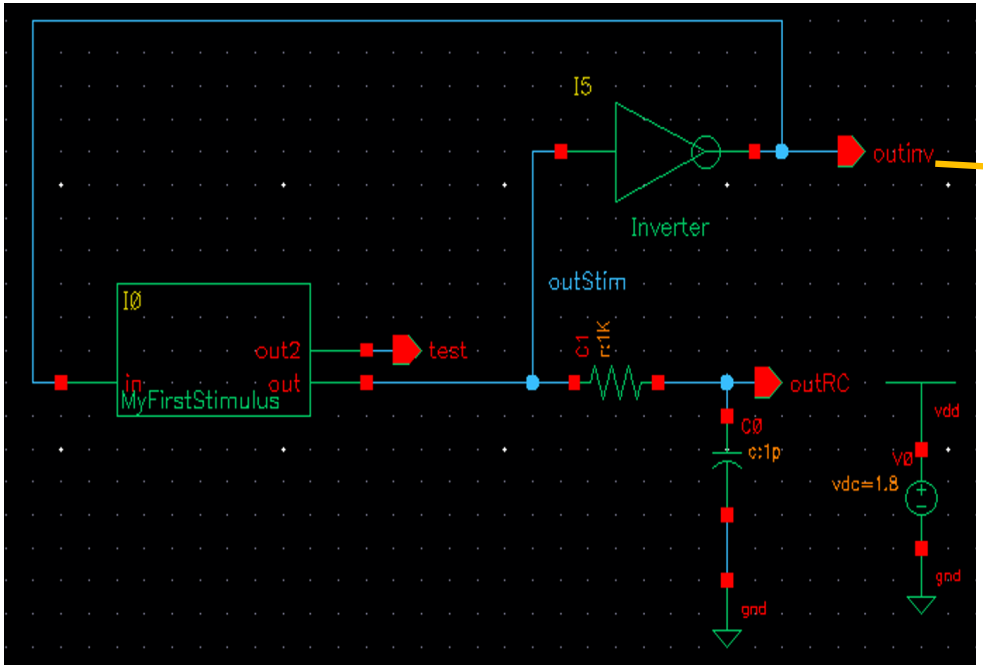


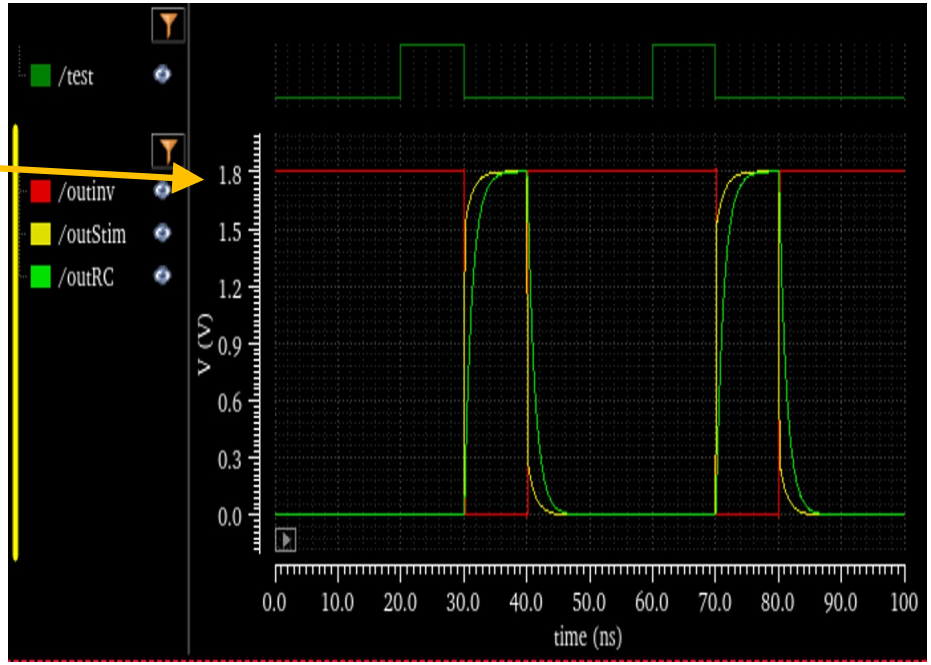
Sample Output

- Now, The inverter is simulated with its ***schematic*** model:

Library	Cell	View Found	View To Use
UMC_18_CMOS	N_18_MM	spectre	
UMC_18_CMOS	P_18_MM	spectre	
VLSI2020	Inverter	schematic	schematic
VLSI2020	MixedSim_Top	schematic	
VLSI2020	MyFirstStimulus	verilog	verilog

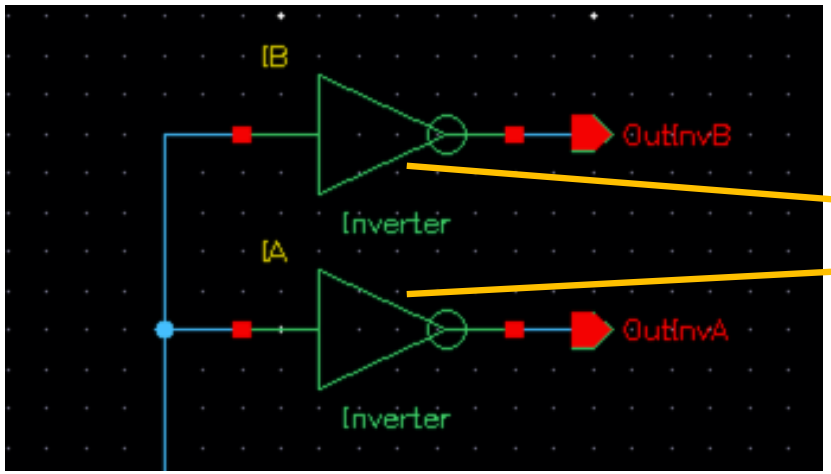
- 'outinv' is now an *analogue* signal

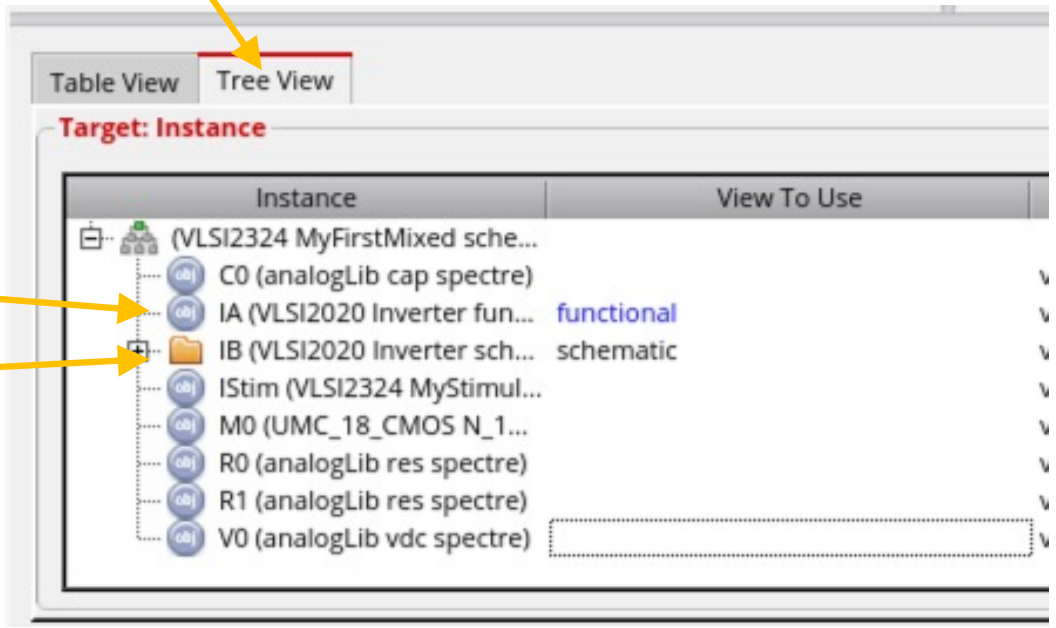




Mixing stuff

- If you have several instances of the same type, you can select for each instance separately which view to use.
- You must use the ‘Tree View’ tab of the Hierarchy Editor:





More Sophisticated Verilog…

- In the Verilog you can
  - Define tasks to better structure your code
  - Use memories to store data
  - Use $random to create random data
  - Use Boolean to collect error messages
  - Print to console (using $display())

```
task LoadRORegister;    // Load the content of the accumulator to the readout shift register
  begin
  ROLoad = 1'b1;
  DoROClk;
  ROLoad = 1'b0;
  end
endtask


initial begin
  ClkShift   = 1'b0;  // clock of vertical shift register, adder, accumulator
  ROClk      = 1'b0;  // clock of configuration register and readout register
  ResetHit   = 1'b0;  // async.
  Inject     = 1'b0;  // async.
  HorSI      = 1'b0;
  LoadShiftB = 1'b0;  // 1: Load, 0: shift
  ACCClear   = 1'b0;  // strobe to clear accumulator (clocked by ClkShift)
  ROLoad     = 1'b0;
  LoadMask   = 1'b1;  // Must be 1 to inject top horizontal shift register signals to column and to pa

  $display("//=============================================================================================");
  $display("// Generate a random mask pattern for each chip and write all patterns to the matrices");
  $display(" //============================================================================================");

  GenerateRandomMaskPatterns;                          // Generate the mask pattern with 50% 1/0
  LoadMask = 1'b1;
  for (irow=0; irow<NROW; irow=irow+1) begin           // We write row after row
    for (ichip=0; ichip<NCHIP; ichip=ichip+1) begin    // clock in the NCOL*NCHIP values
      for (icol=0; icol<NCOL; icol=icol+1) begin
        HorSI = Wmask[ichip][irow][icol];              // take it from the Wmask register
        DoROClk;                                       // Clock horizontal register
      end  // icol
    end // ichip
    DoShiftClk;                                        // One row done. Clock it down the column.
    $display("Row %d DONE.", irow);
  end // irow
  LoadMask = 1'b0;                                     // Freeze mask pattern and set top input of shift
  $display("DONE Writing Mask pattern.");

```

Checking Inputs

- You can pass analogue values (after conversion in an IE) to the Verilog and inspect them.

```
 3 module MyFirstStimulus (output out, input in, output out2);
 4 parameter freq = 10;
 5
 6 reg [1:0] cnt;
 7 assign out  = cnt[1] && cnt[0];
 8 assign out2 = cnt[1] && ~cnt[0];
 9
10 initial begin
11   cnt = 2'b0;
12   $display("****************************");
13   $display("In MyFirstStimulus: initial");
14   $display("****************************");
15 end
16
17
18 always begin
19   #freq begin
20     $display("****************************");
21     $display("In MyFirstStimulus: count by 1");
22     $display("****************************");
23     cnt = cnt+1;
24   end
25 end
26
27 always begin
28   #1 begin
29     $display("****************************");
30     $display("In MyFirstStimulus, at time %3d, input is %3d", $time, in);
31     $display("****************************");
32   end
33 end
34
35 endmodule
```

input

Define a register for a 2 bit counter

Initial condition of counter

Increment the 2 bit counter value every #freq ns

Print the value of ***in*** to console every ns

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

EXERCISE: MIXED MODE SIMULATION

Exercise: Clock Generation

- Step 1: Create a 'ClockGenerator' cell
  - Generate a Verilog view
  - Use a parameter
    ```
    parameter del=10;
    ```
    to set the clock period. (Parameters can be overwritten in the properties of the symbol. You may have to change the 'CDF Parameter of view' combo box to 'verilog')
  - Follow all steps until you have the symbol
- Step 2: Create a new schematic (for simulation)
  - Instantiate the ClockGenerator
  - Add an inverter or at lease a RC element to do something with the clock
- Step 3: Mixed mode simulation
  - Follow all described steps to setup and run a mixed mode simulation
  - Browse through the results

Exercise: Clock Divider

- Step 4: Divide by 2:
  - Create an edge triggered flipflop from two latches (or take if from a SUSLIB..)
  - Use it to divide the clock by 2.
- Step 5: Checking via Verilog: 'ClockChecker' cell
  - Make a Verilog module which has a clock output and an ***input*** for the divided clock
  - Use Verilog code to verify that the clock is divided correctly

- NOTE: When re-running the simulation, the results in the lower hierarchy might be missing despite for 'save all'.
  → Closing and re-opening the results browser should fix this.