



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)

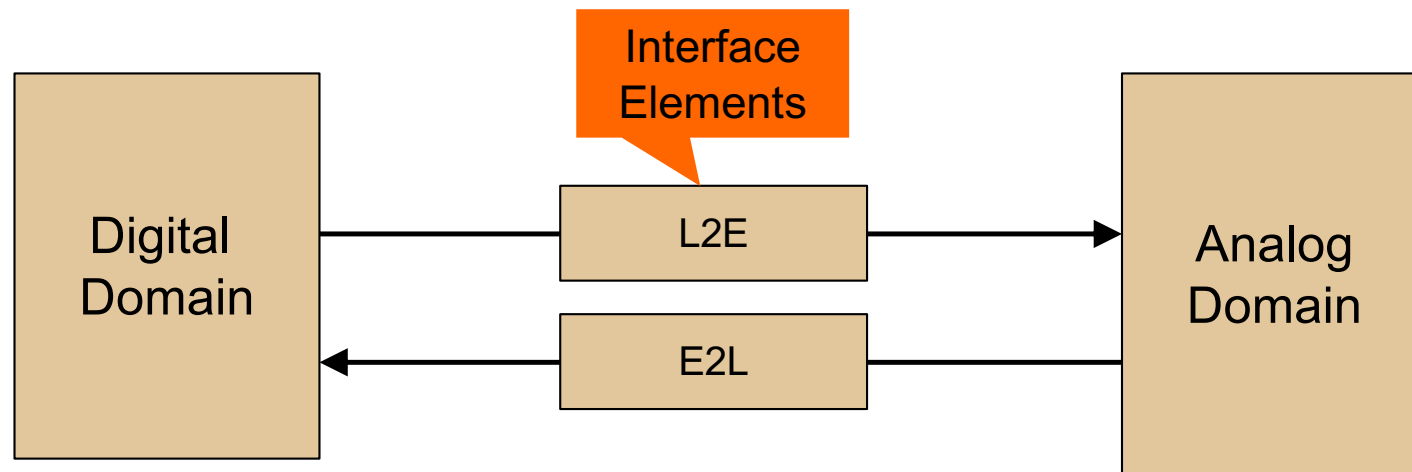
→ Mixed Mode Simulation:

- 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 Simulator for digital part
 - Analogue simulator for analogue part
 - *Interface Elements* translate between both domains
- Time must be (internally) synchronized



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

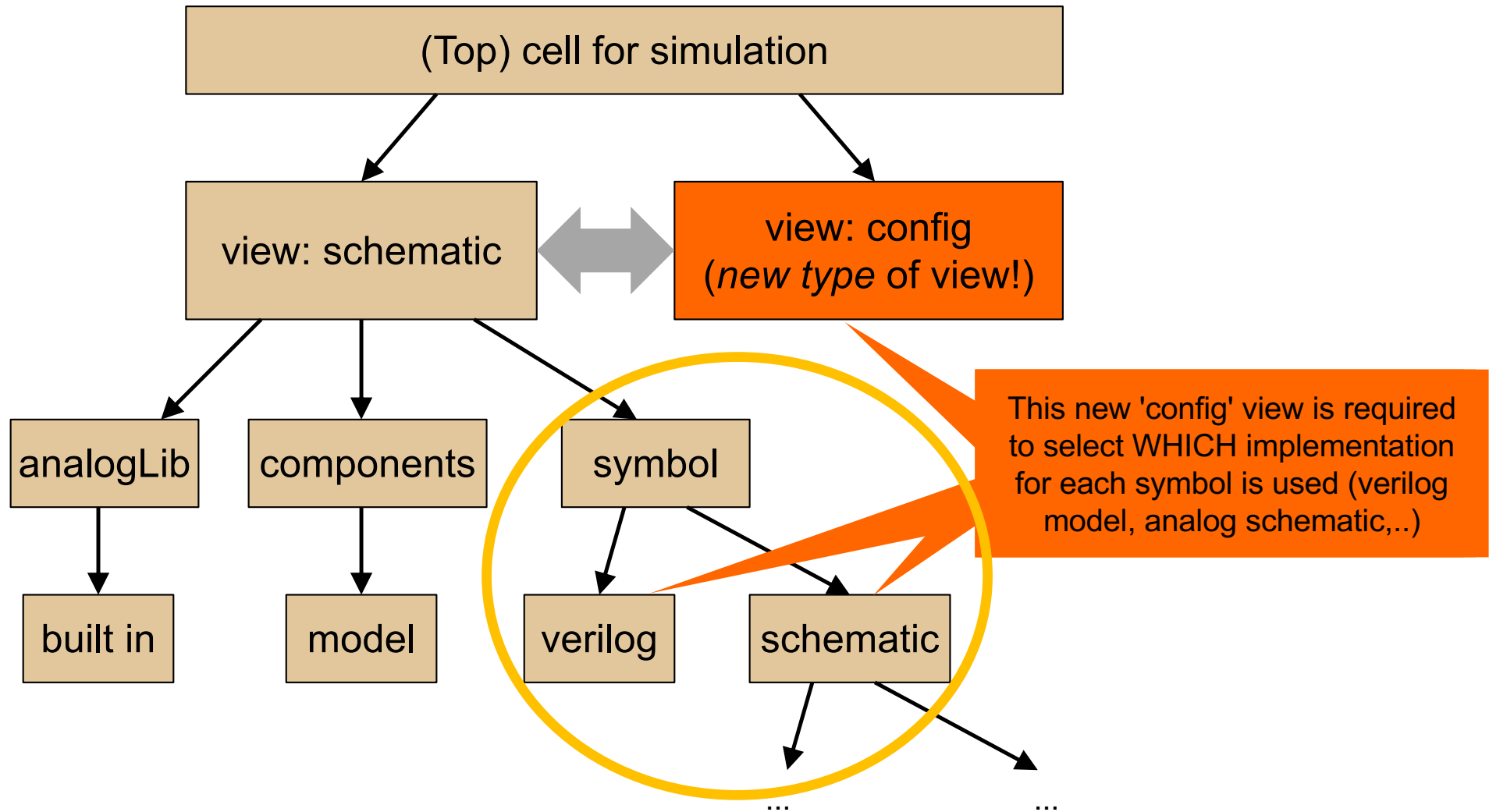


Note

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



What do we Need?



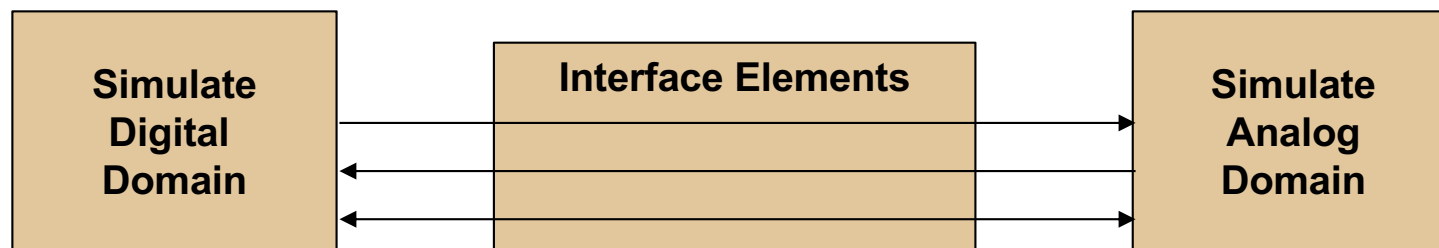


A SIMPLE EXAMPLE



A Simple Example

- The following slides show how to set up a simple mixed mode simulation in the *Virtuoso ADE* environment with the following steps:
 1. Creating a *Verilog module* 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, which describes the hierarchy
 4. Specifying 'Interface elements' which connect the digital and analog domains.





Before You Start:

- We need to make the 'connectlib' available:
- In file `cds.lib`, add the line

```
DEFINE connectLib  
/opt/eda/XCELIUM2203/tools.lnx86/affirma_ams/etc/connect_lib/connect  
Lib
```
- You can also use the Library Manager:
 - In the Library Manager: Edit → Add Library Path...
 - Edit → Add Library
 - ... (name must be connectLib)
 - Save
- This must only be done once, library definition is saved in `cds.lib`

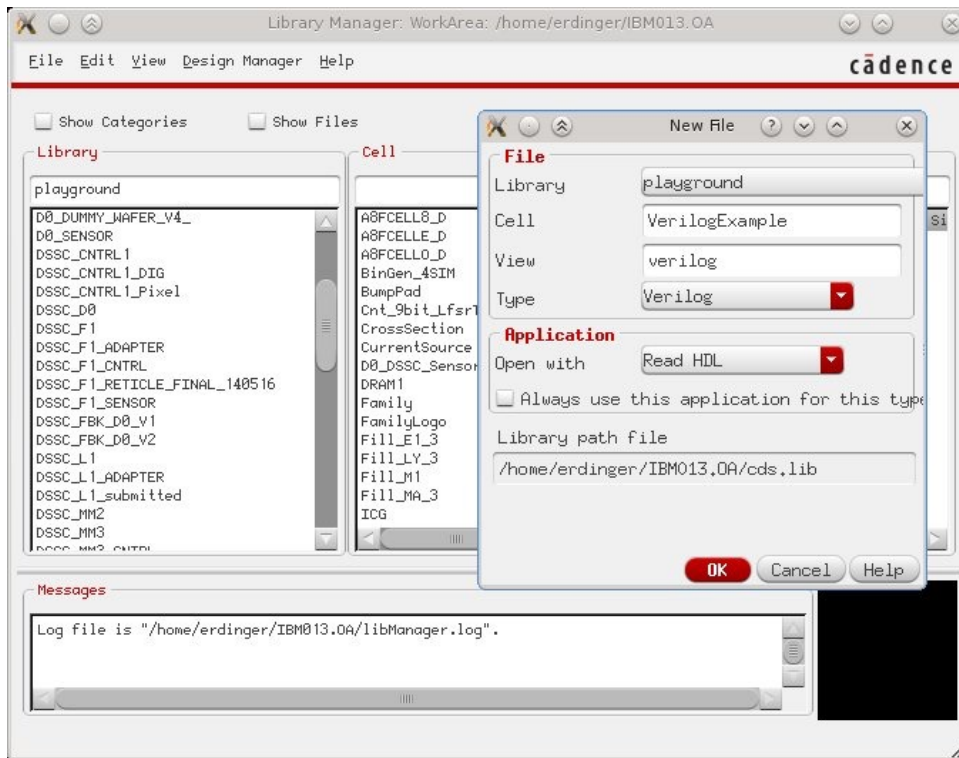


Editor

- The editor of your choice can be specified in `.cdsinit`
 - `editor="..."`
- You can also use the shell:
`export EDITOR=gvim`
- or the CIW:
`editor="gvim"`



1. Creating a New Verilog Module



- In 'Library Manager':
 - File → New → Cell View
 - 'Cell': name of verilog module
 - 'View': 'verilog' (Non-Capital!)
 - 'Type': Verilog

- The Cadence text editor opens with a 'naked' Verilog module



1. Creating a Verilog Module with its Symbol

- Fill the Verilog module with some code.
 - The code need **not** be synthesizable

- For instance

```
initial out = 1'b0;
always #10 out <= ~out;
```

- (1 time step = 1 ns by default)

- When you close the text file, it is automatically parsed. Correct it until there are no errors left.

```

module ClkGen (output reg CLK);
    parameter DEL=10;

    initial CLK = 0;
    always #DEL CLK <= ~CLK;

endmodule
-- INSERT --      1,1
    
```

Better use blocking statement

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



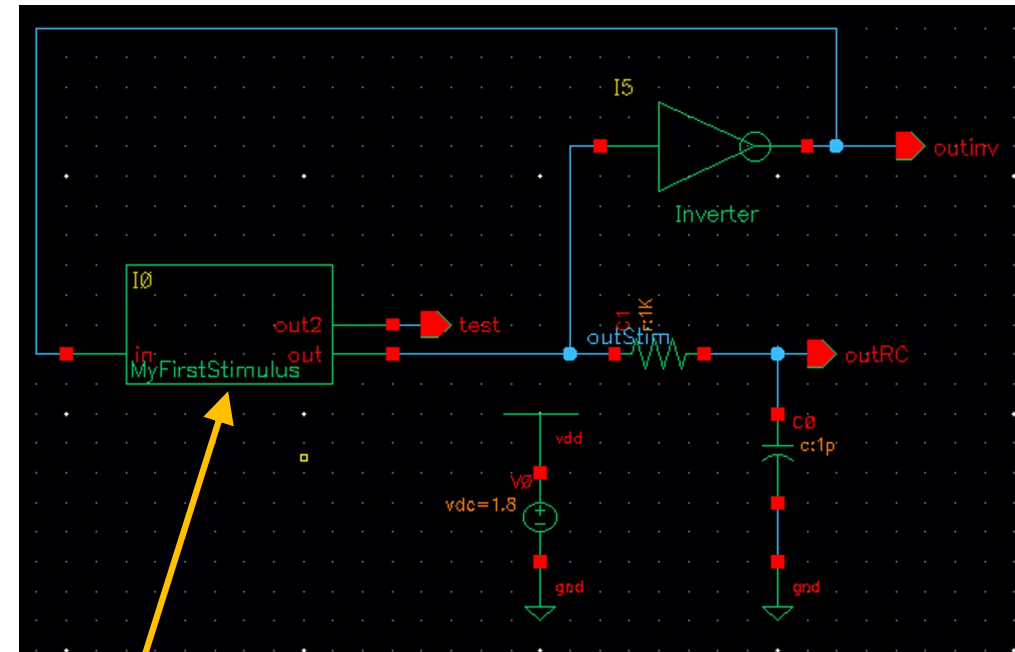
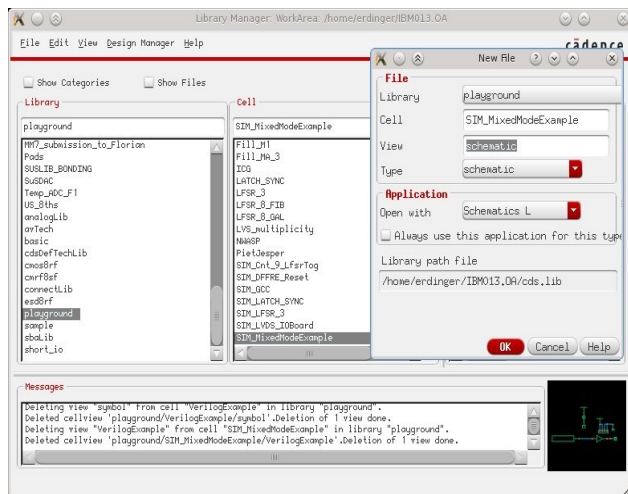
Note

- Error messages of Verilog compilation end up in
`.cadence/dfII/TextSupport/Logs/...`
- In my editor, the file can be seen with View->Parser Log



2. Creating A Top Level Simulation Module

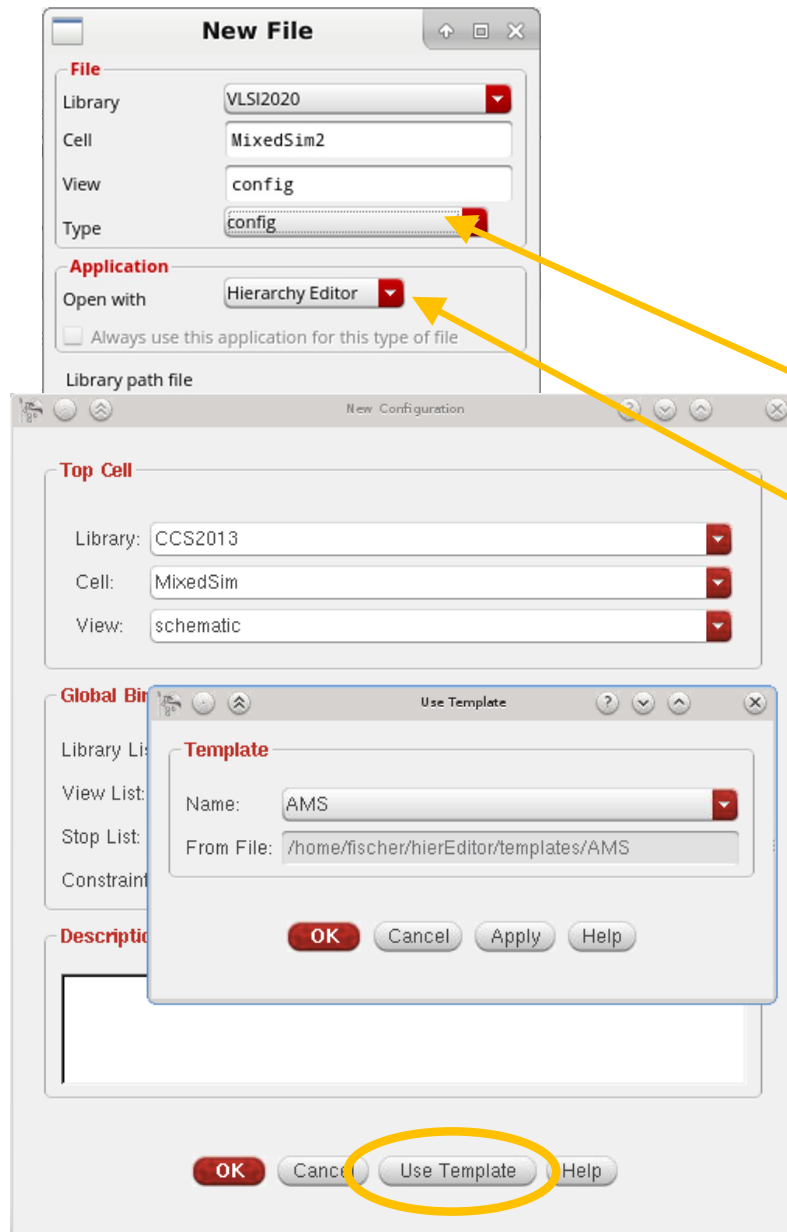
- In 'Library Manager'
 - File → New → Cell View
 - Create a schematic



- Put an instance of your Verilog module, i.e. the symbol
- If the Verilog contains *parameters*, the symbol inherits them.
 - To see them: In the instantiated symbol, select CDFParameter - > Verilog (not 'Use Tool Filter')
- Add some analog circuit (symbols, primitives, sources, ...)
- 'Digital' and analog circuits can directly be connected



3. Creating the Simulation Configuration View

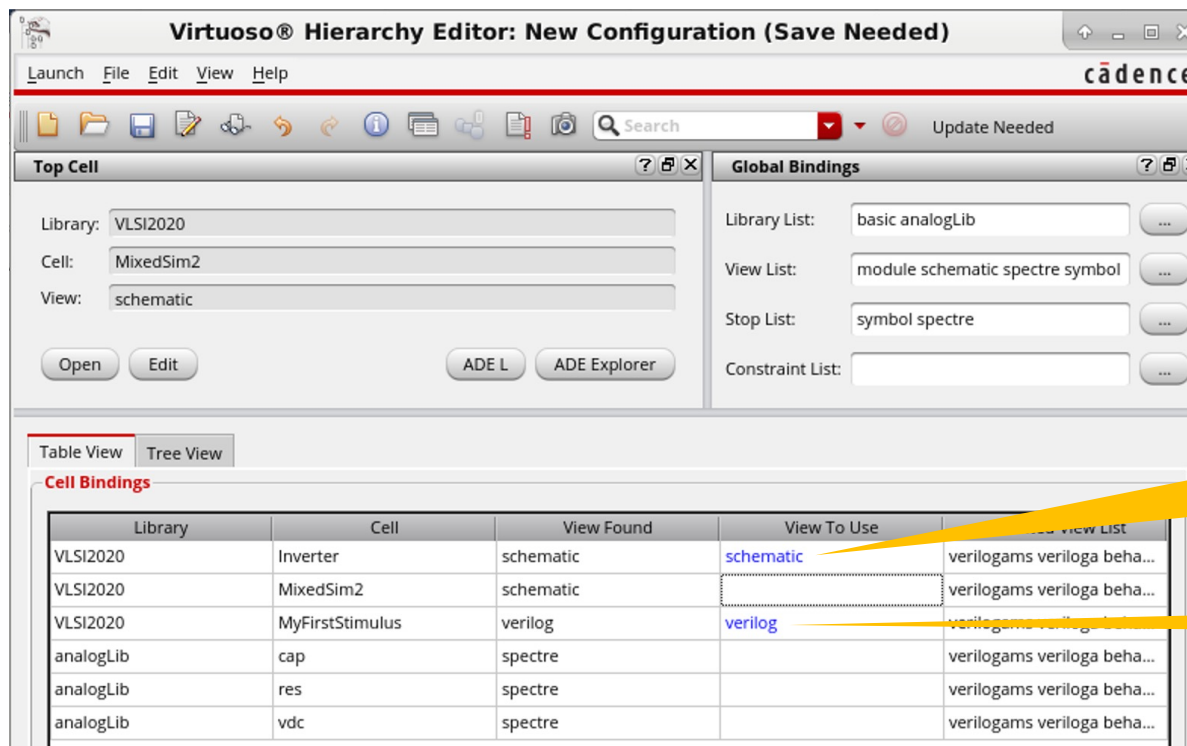


- The AMS simulator needs a 'config' view for the *simulation schematic*
- In 'Library Manager':
 - Select your simulation schematic
 - File → New → Cell View
 - 'Type': config (name changes to 'config')
- Note that 'Application' switches automatically to 'Hierarchy Editor'
- In the next window: change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



3. 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)



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

Right Click to select



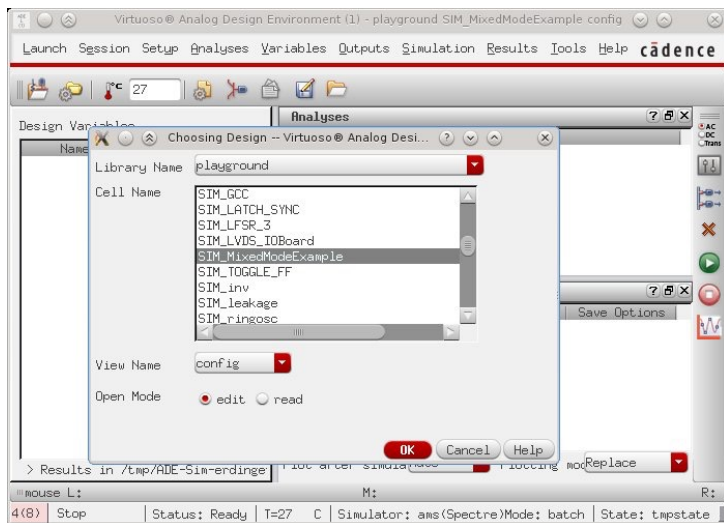
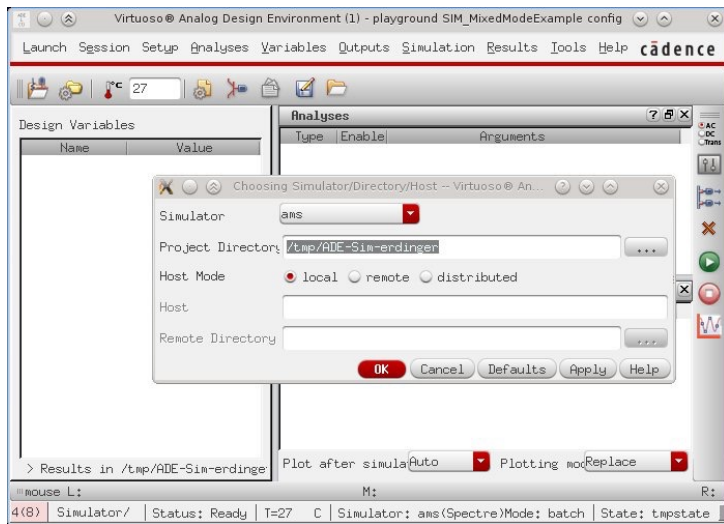
4. Adding the Interface Elements

- There are **built-in Interface Elements (IE)** supplied with the simulator (which can also be customized if necessary)
- They are located in the 'connectLib'
- The IEs to be used are selected in the ADE when setting up the simulation (specifics see later)
- They are inserted automatically (do not have to be placed in the schematic manually)



Setting Up the Simulation and Outputs

- Open the top level simulation schematic
- From the menu: Launch → ADE
- Setup → Design
 - Change 'View Name' to 'config' (which we have created before)
- Setup → Simulator/Directory/...
 - Change 'Simulator' to 'ams'
- Add a transient simulation





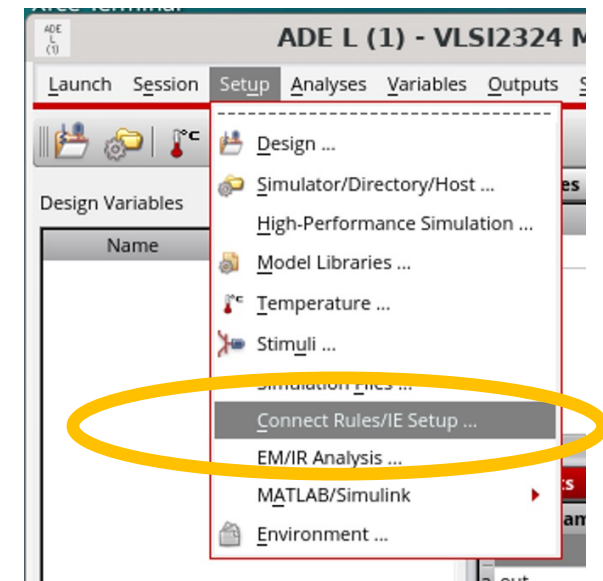
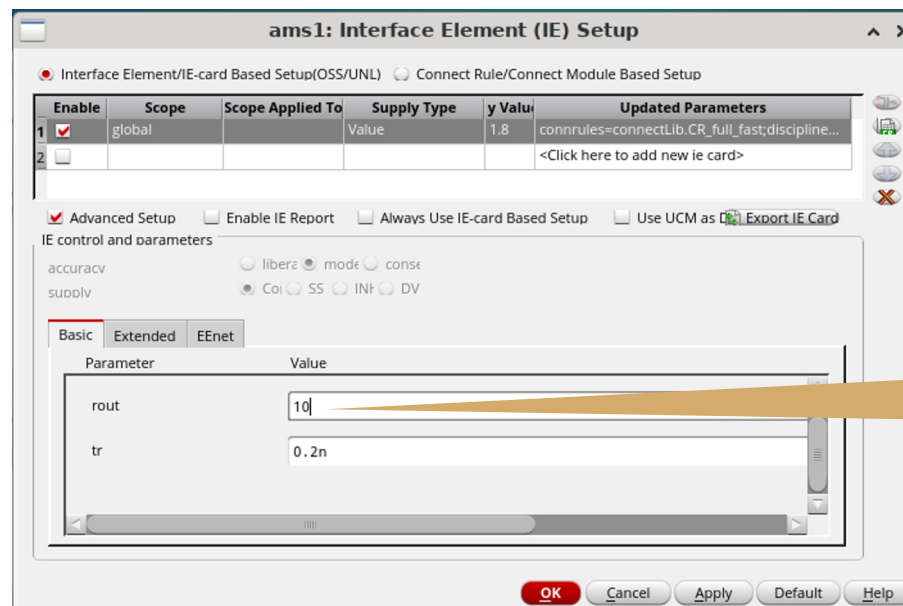
Choosing Information to Store

- AMS saves nothing by default:
- 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)

- This is a bit tricky. The mechanism has changed depending on software version.
- At the moment, you can access the connect rules from ADE->setup->Connect Rules
- In the window, go to 'advanced setup'. All interfaces are set here. An easy model is a series resistor and a rise time. Set them to small values.



Series R
And
Rise time



Using Active Devices

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

- If they are not set, a quick way to get them is:
 - Save the state of the (ams) simulation to a view
This view is visible in your linux library path under /libname/cellname/ams_state...
 - In this cell, there is a file modelSetup.state, which is probably empty (contains nil)
 - Copy a 'modelSetup.state' file from another simulation state, containing the library paths.



Running and Viewing the Simulation

- Run the simulation ('play button')
- In the log file you can see that there are several steps:
 - Compilation
 - Elaboration
 - Simulation
- Verilog **\$display** task prints to the log file
- Open the results browser to look at the results:
in the ADE menu: Tools → Results Browser ...

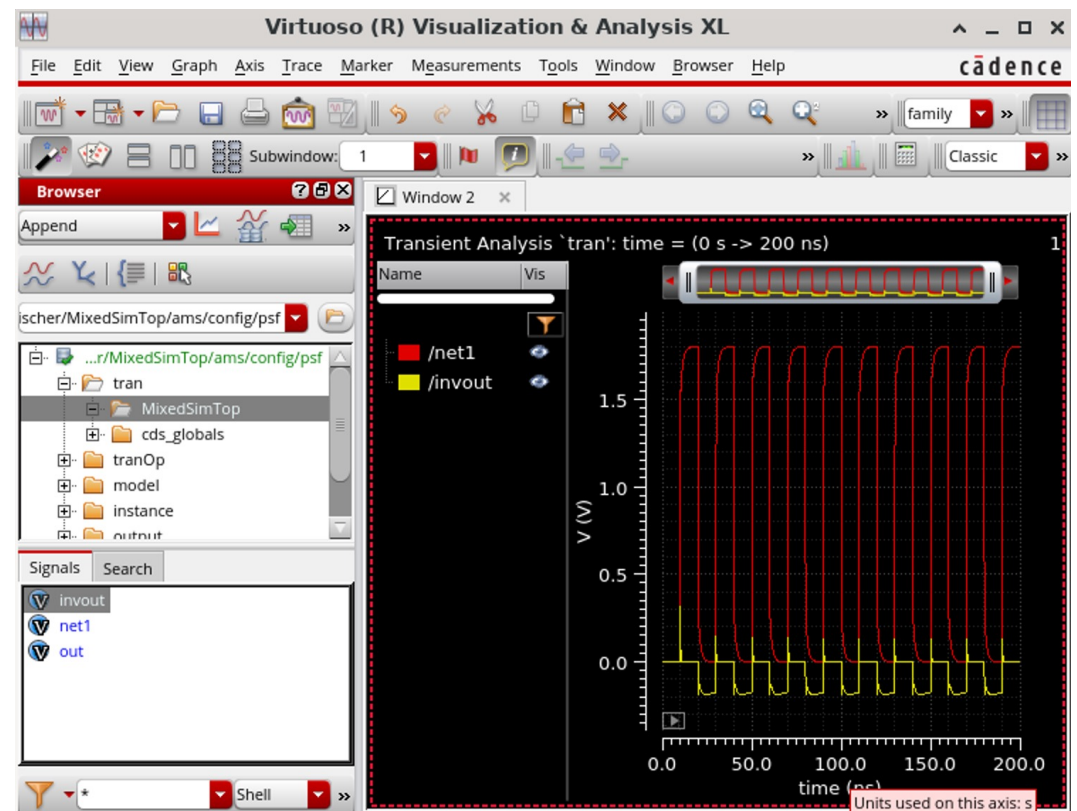
- Select Outputs in the left windows



Selecting Waveforms

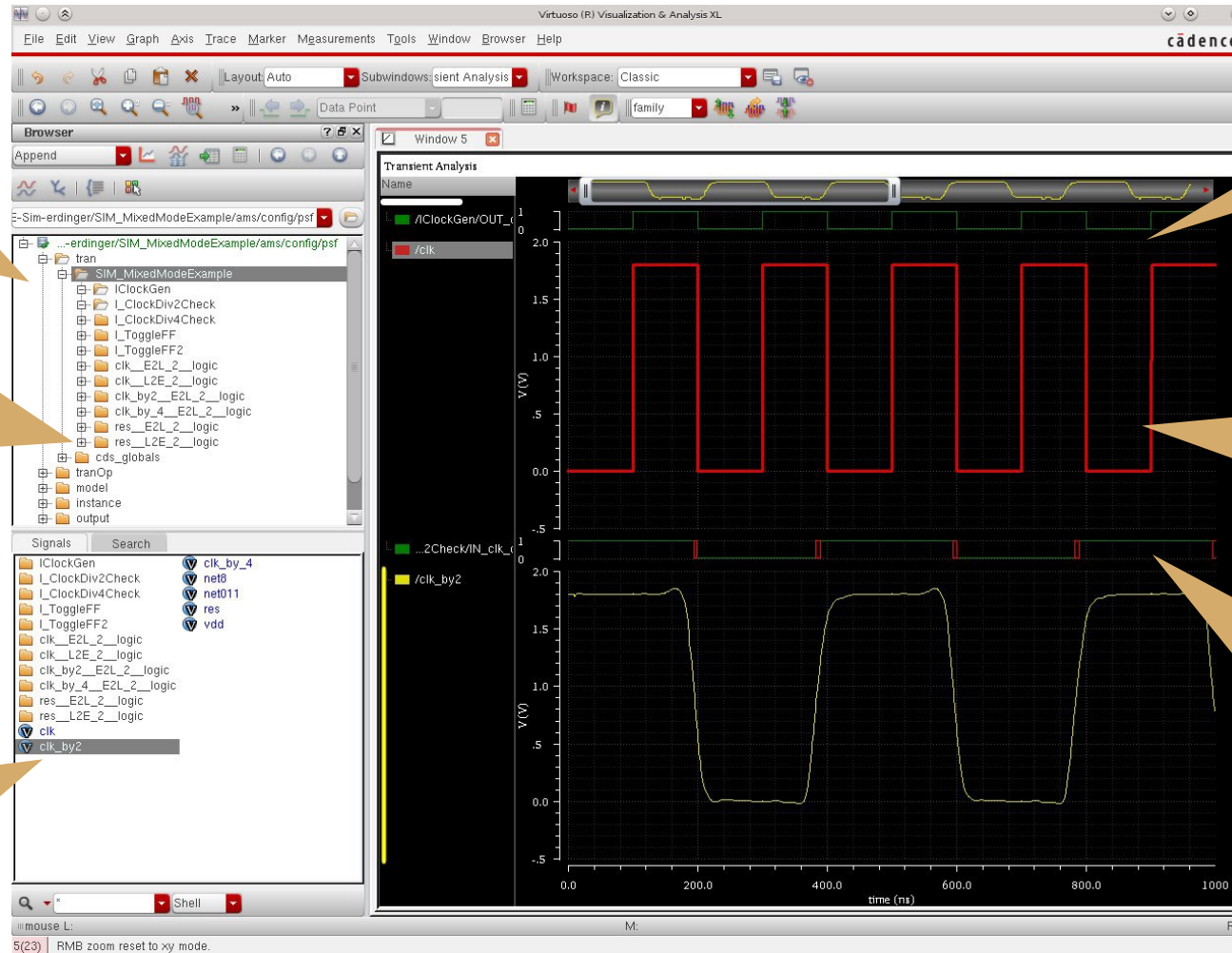
- To select digital waveforms, open the 'results browser'
 - for instance from the schematic window (next to 'calculator icon')
 - Or from the simulator -> tools -> ..
 - -> You get a pane with 3 entries (calc., res. browser, results)

- 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
transitions



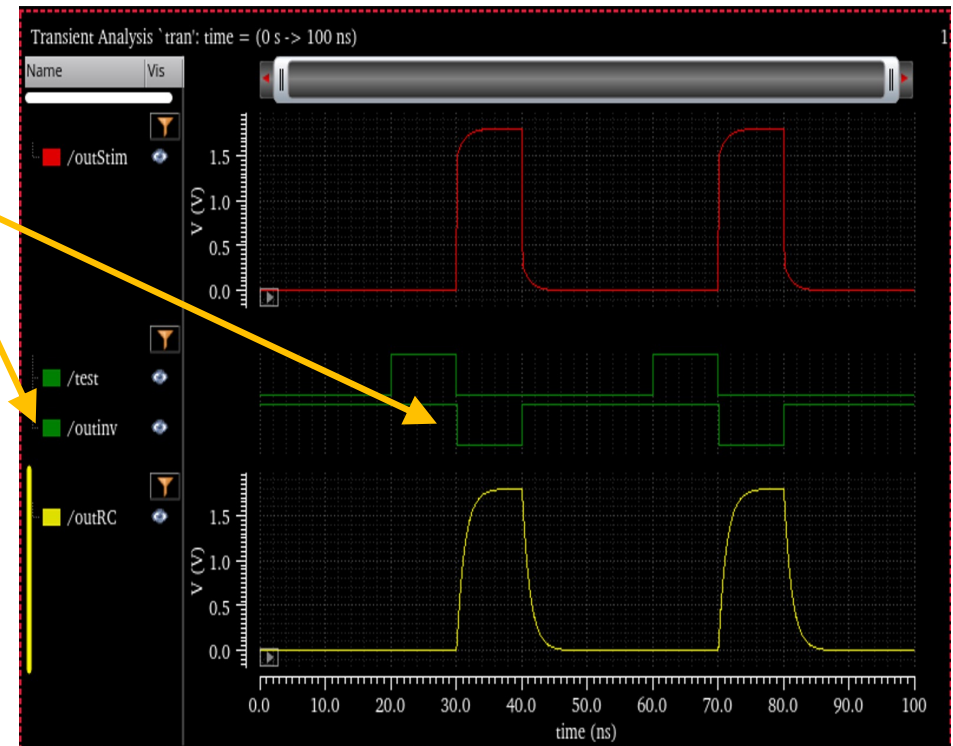
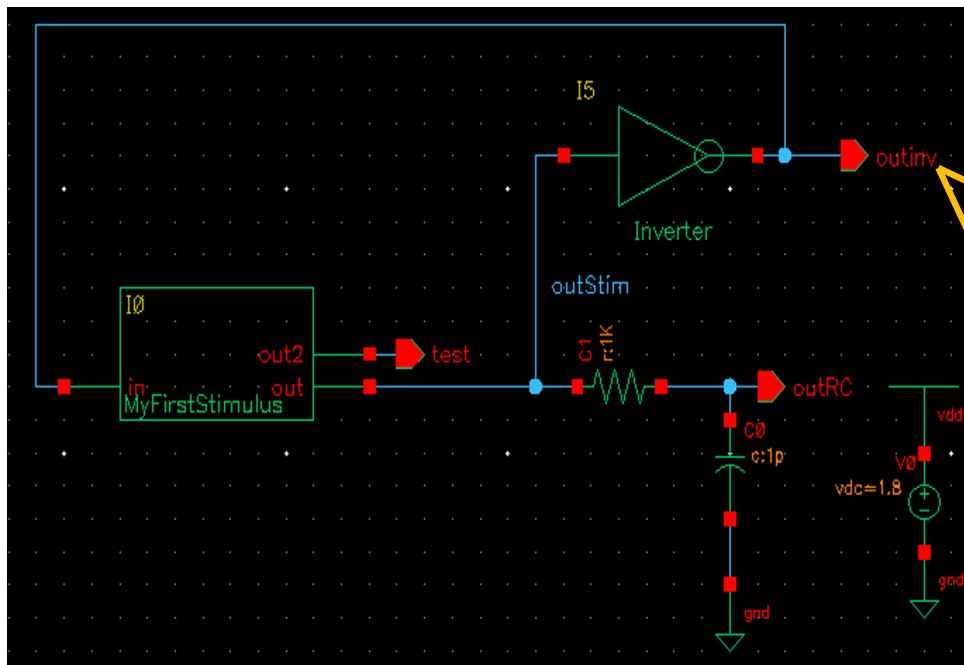
Sample output

- 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



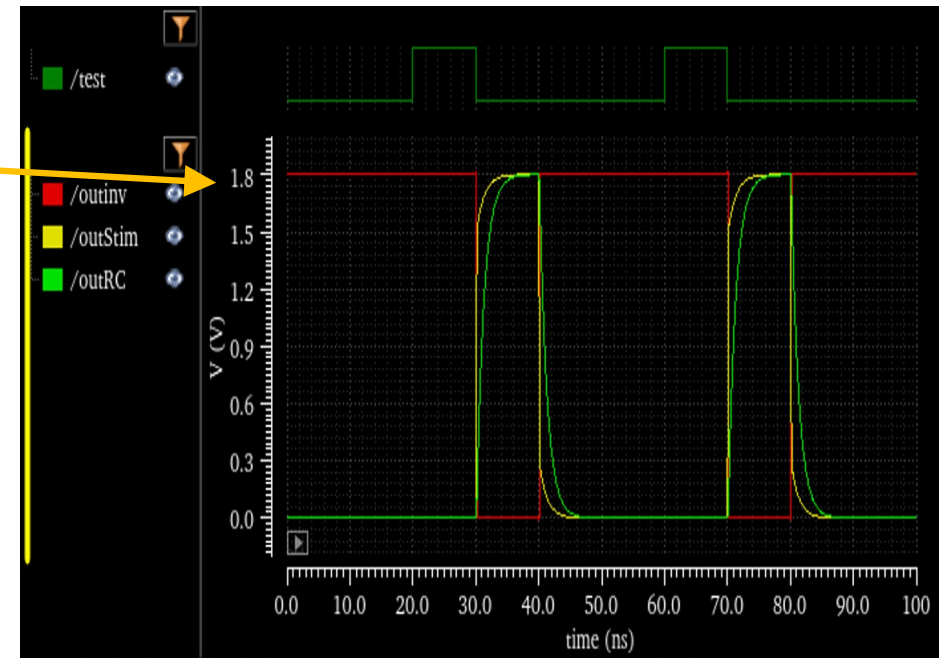
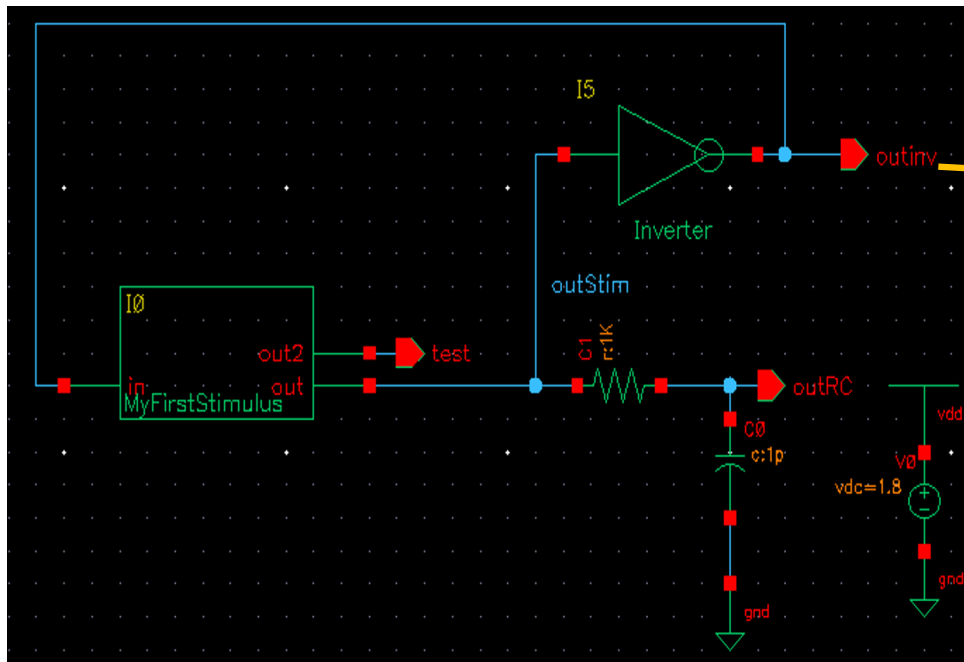


Sample output

- 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 an analogue signal





CHALLENGES IN ANALOGUE / MIXED MODE DESIGN



Challenges in Analogue Design. Noise

▪ **Noise** (physical, unavoidable)

- Thermal noise in resistors and MOS is rather well described and quite technology independent
- 1/f noise in MOS technology dependent and not well modelled.
- More details in lecture 'Advanced Analogue Building Blocks'

▪ **Remedies:**

- Chose insensitive circuit topologies, avoid resistors
- Reduce noise by reducing g_m of MOS where possible (for instance in current sources)
- Reduce 'relative' noise by increasing current in MOS

▪ **Study with**

- Mathematical analysis
- AC noise simulation (see later)
- transient noise simulation (see later)



Challenges in Analogue Design: Mismatch

▪ Device Mismatch

- From fluctuations in fabrication
- → Mismatch in current mirrors
- → Offset voltage of differential amplifiers
- → ...

▪ Remedies:

- Use large devices to reduce geometric uncertainties
- Use identical layout, common centroid, guard devices,...
- For MOS: Use high V_{GS} to reduce effect of V_{Th} mismatch
- See slides on layout...

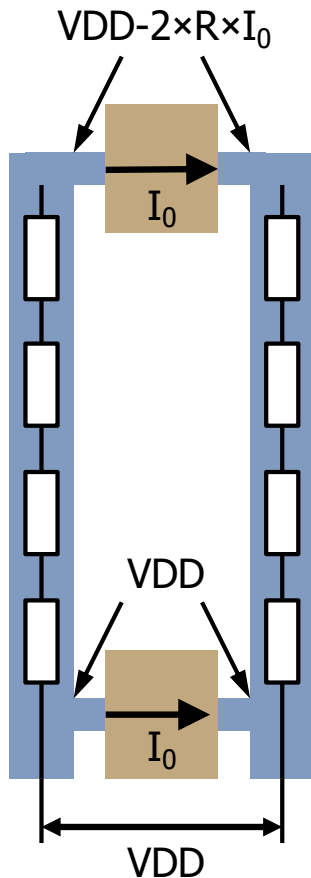
▪ Study with

- Back-on-the envelope calculations
- Monte Carlo Simulations (see later)



Challenges in Analogue Design

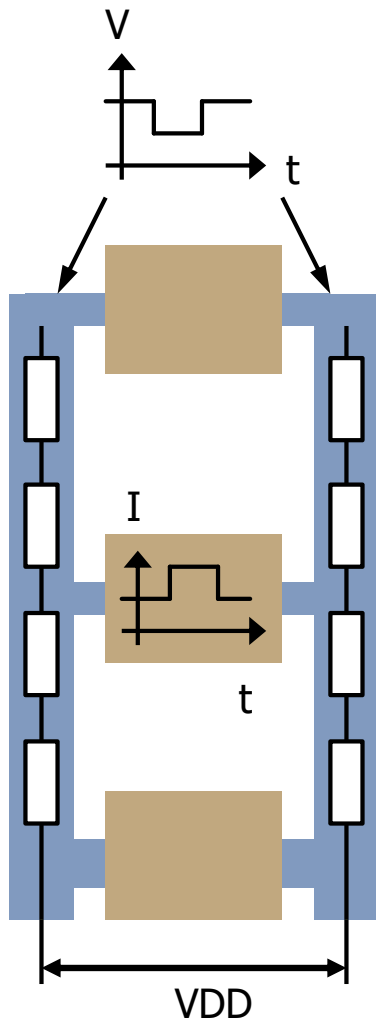
- (Static) power supply voltage drops
 - From 'IR-drop' in supply traces



- Remedies:
 - Separate voltage rails for different circuit blocks
 - Voltage drop compensation circuits
 - Circuits with good (low freq.) power supply rejection (PSRR)
 - Multiple (local) references
 - Trimming
- Study by:
 - Simulation of extracted design (or estimated resistances)



Challenges in Analogue Design

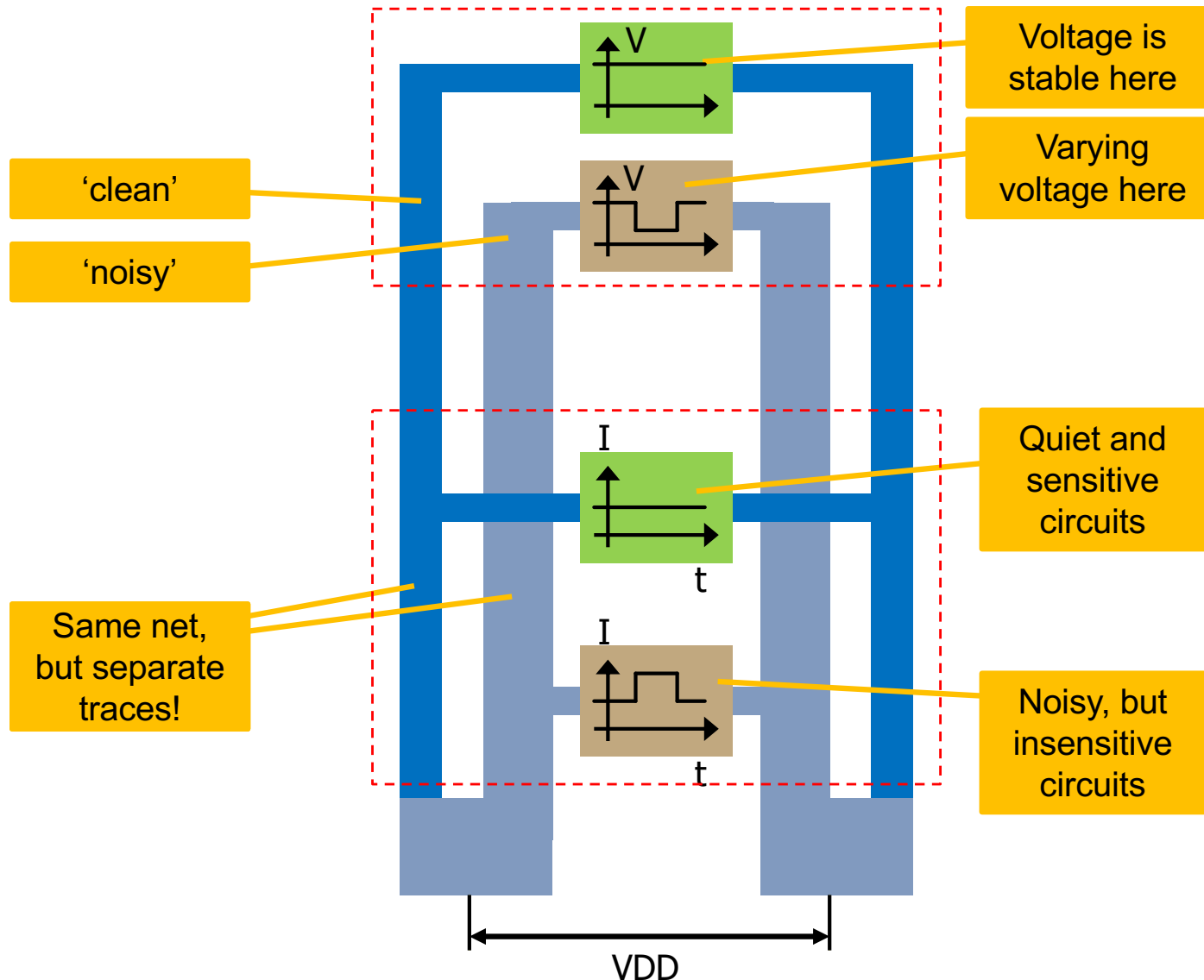


- (Dynamic) power supply glitches
 - Shortly increased current consumption leads to extra voltage drop and thus quickly changing supply voltage
- Measures against Supply issues
 - Separate supplies or supply traces
 - Local decoupling of power ('energy storage')
 - Local decoupling of bias signals (between gate and SOURCE!)
 - Circuits with good high freq. 'power supply rejection ratio'
 - Differential circuits
- Study with:
 - Transient sim. of extracted design (or estimate res.!)
 - Often must simulate large circuits and special situations



Trick: Multiple supply traces

- Use 'clean' and 'noisy' supply traces:





Mixed Mode Challenges

- Digital and Analogue on one chip:
 - Digital activity has very varying currents (clock edges, different data patterns) and leads to dynamic glitches on power supply.
 - Large digital signals can (ac-) couple to sensitive analogue nodes:
 - From trace to trace through parasitic C
 - From MOS gate to channel (cap is high!)
 - From bulk/well to channel

- Some Measures:
 - Separate supply / ground NETs for analogue / digital (vdda, vddd)
 - PMOS well or NMOS substrate (triple well NMOS!) connected to clean (analogue) supply / ground
 - Metal shielding between traces (but connect shield to where ?)
 - Distance between analogue and digital (group analog/digital)
 - Use differential structures so that glitches cancel out
 - ...



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 least 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 it 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.