Digital-on-top Physical Verification

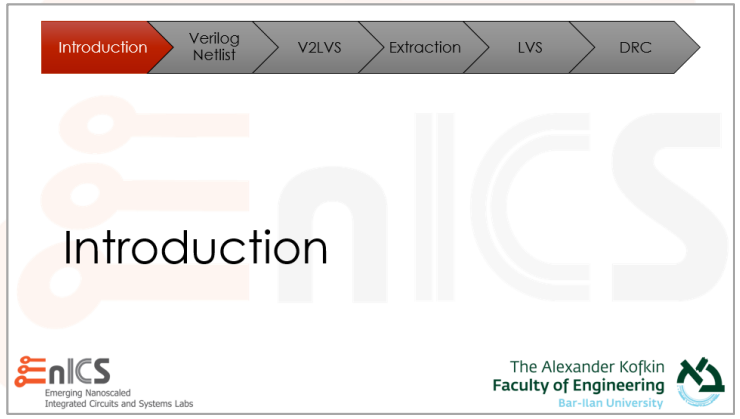
LVS and DRC using Innovus and Calibre

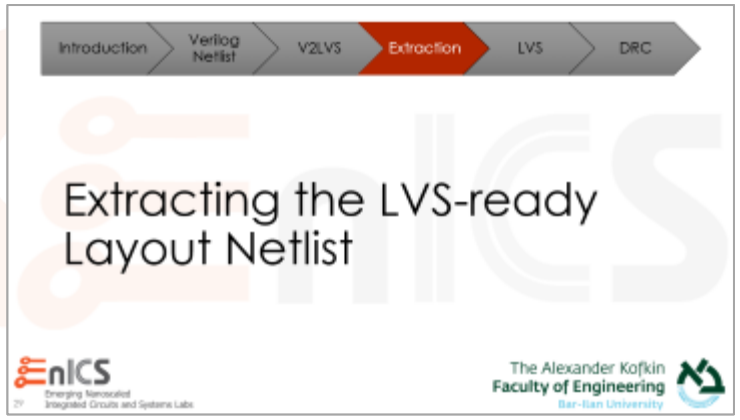
27 September 2020

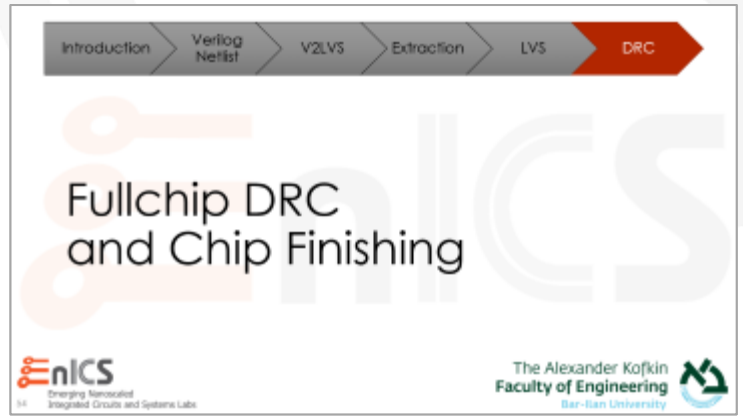
Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Outline

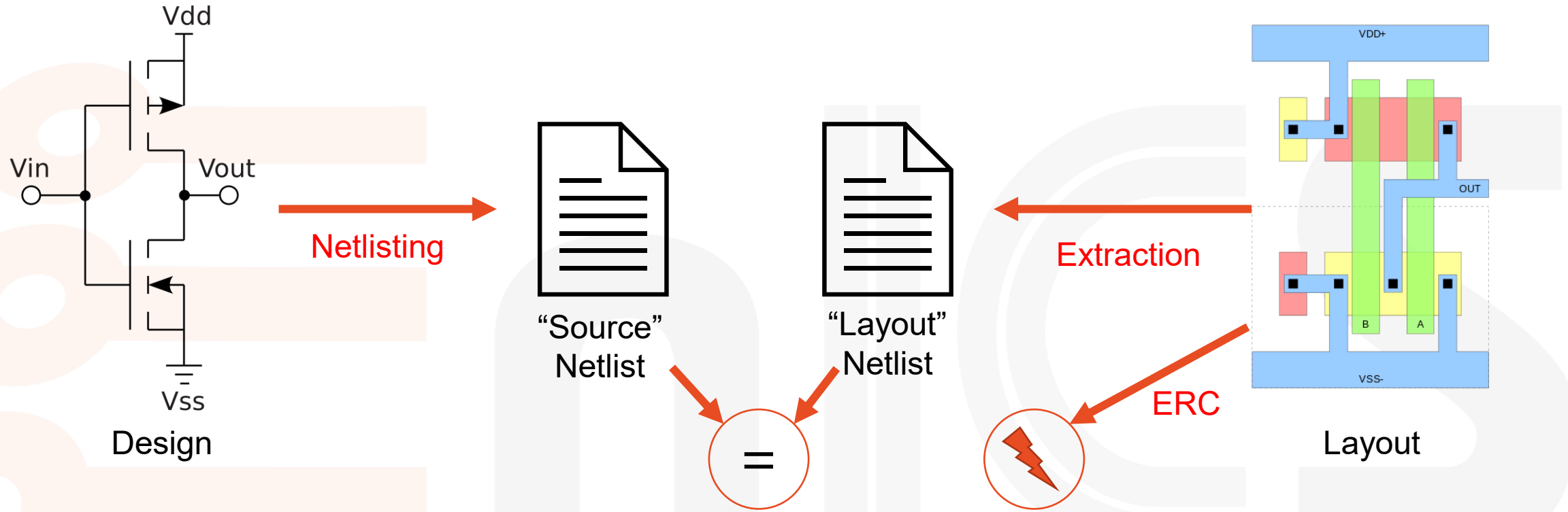






Introduction

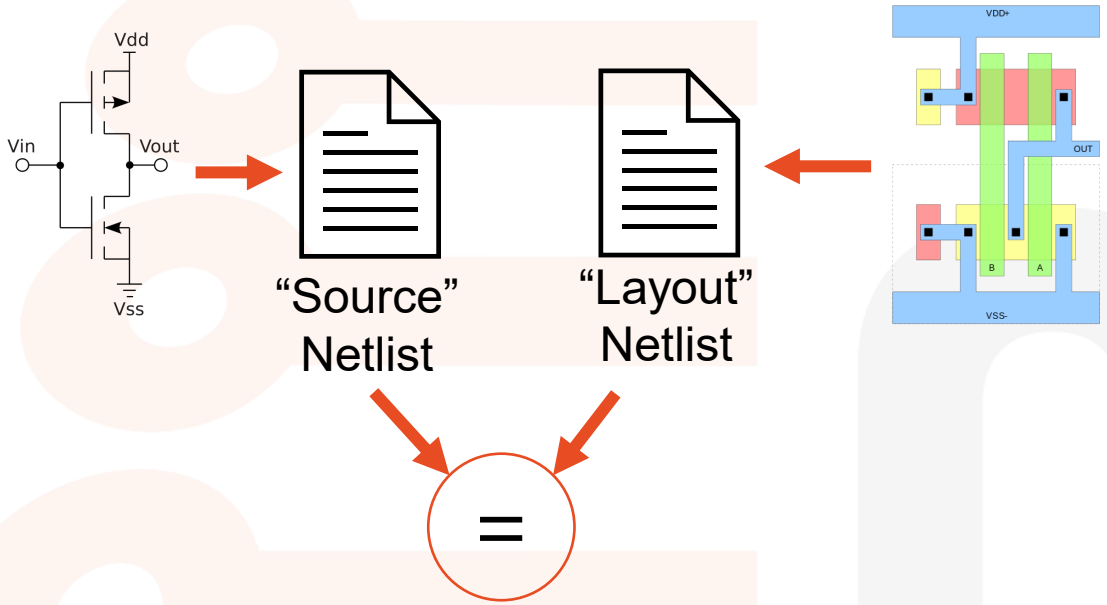
The LVS Flow



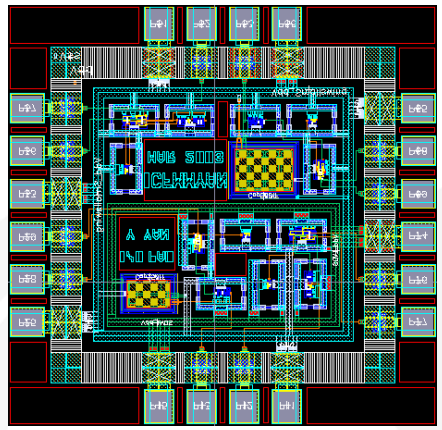
- **Preparation of the Source netlist**
- **Extraction of the Layout netlist**
- **Check for shorts, opens, and verify bulk connections (ERC)**
- **Comparison of Source vs. Layout**

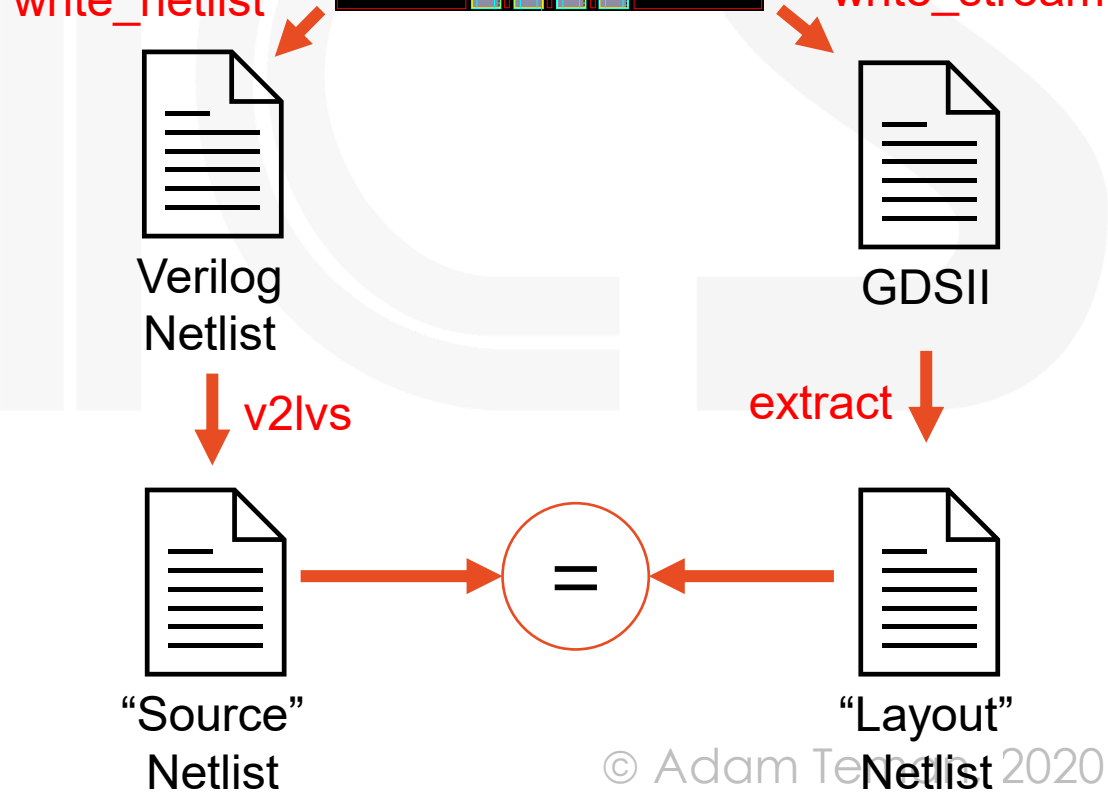
Custom vs. "Digital-on-Top" LVS

Custom (Virtuoso) LVS Flow



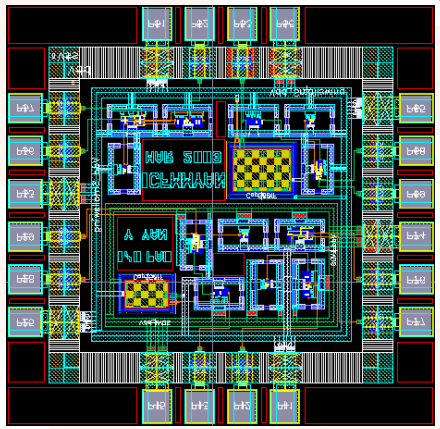
Digital-on-Top (Innovus) LVS Flow





- **Why is the digital-on-top flow problematic?**
  - Because of the integration of IP and hierarchical blocks!

The complete digital-on-top LVS flow



write_netlist

.v

Verilog Netlist

.cdl

.include

v2lvs

.cdl

"Source" Netlist

write_stream

.gds

GDSII binary

extract

.sp

"Layout" Netlist

.gds

merge

=

LVS Report

ERC

ERC Report

So let's make it simple

- **Before going into the problematic details, let's assume everything is fine:**
  - Write out Verilog netlist from Innovus
    - `write_netlist -phys -exclude_leaf_cells -flatten_bus my_module.v`
  - Run v2lvs to create the "Source" SPICE netlist
    - `v2lvs -sn -v my_verilog.v -o my_output_cdl.cdl -s my_includes_file.sp`
  - Write out GDSII from Innovus
    - `write_stream my_layout.gds -merge $ALL_GDS -map_file $mapfile -unit 1000`
  - Extract "Layout" SPICE netlist from GDSII
    - `calibre -hier -64 -hyper -turbo -spice my_layout_netlist.sp runset.extract`
  - Compare "Source" and "Layout" Netlists
    - `calibre -hier -64 -turbo -hcell heclls.txt runset.compare`
- **Now let's look at all those painful details…**

Creating the LVS-ready Verilog Netlist

Emerging Nanoscaled Integrated Circuits and Systems Labs

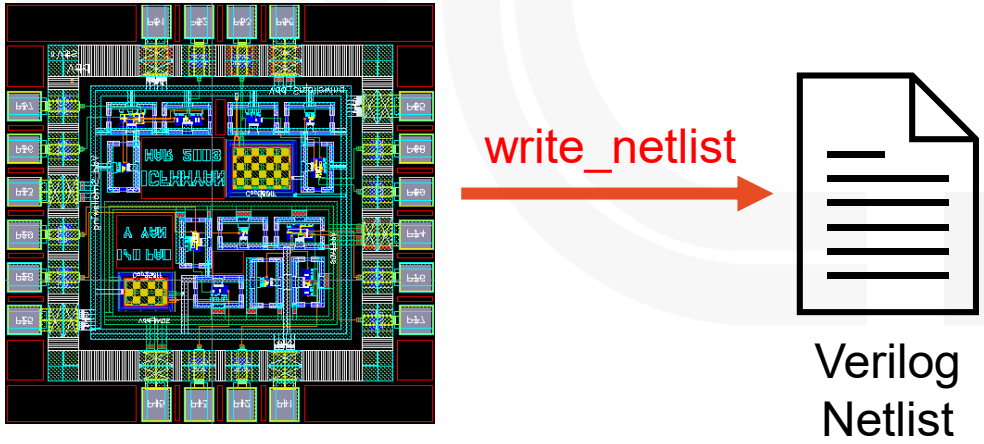
The Alexander Kofkin Faculty of Engineering
Bar-Ilan University

Writing out the Verilog Netlist

- **Basic command:**

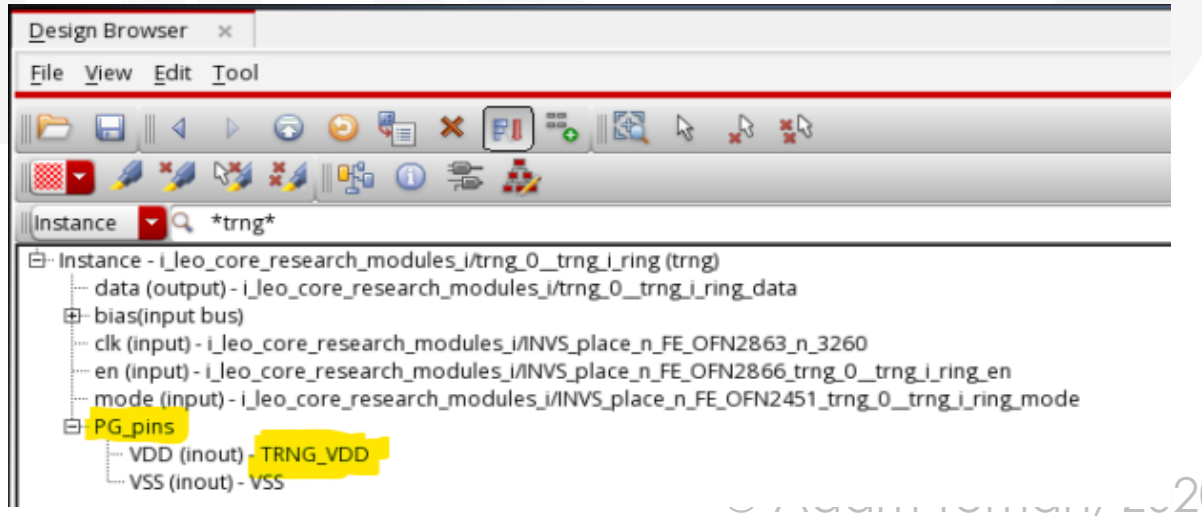
```
write_netlist -phys -exclude_leaf_cells my_module.v
```

- **But many problems:**
  - Global Net connectivity
  - Bus Notation
  - Assigns
  - Flipped Busses
  - Excluded Instances
  - Excluded Hierarchical Blocks



Problem #1: Missing Global Nets

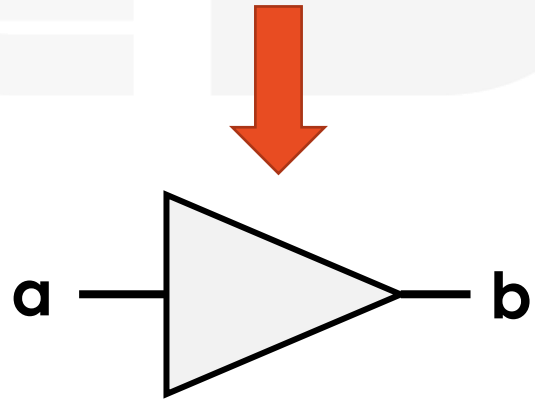
- **Issue:**
  - Logical connectivity (GTL netlist) doesn't require power nets
  - CMOS gates assume the existence of a logic '1' and a logic '0'
  - But these logic levels may come from different voltage sources
- **Solution:**
  - The `write_netlist -phys` flag writes out global power nets
  - However, you first need to initialize these in CPF/UPF or `init_design`:
    - `set_db init_ground_nets`
      `set_db init_power_nets`
  - And you need to connect them to the right pins
    - `connect_global_net`
  - To verify the connections, use the design browser



Problem #2: Assigns in your Netlist

- **Issue:**
  - RTL uses the `assign` keyword in Verilog quite frequently.
  - But many Gatelevel tools, don't like these `assign`s.
  - Synthesis *should* get rid of them, but it doesn't always.
  - **v2lvs** will convert some assigns into `*.connect` commands, but LVS will sometimes fail, for example, an `assign` connecting two inputs.
- **Solution:**
  - Remove `assign`s during `init_design`:
    - `set init_remove_assigns 1`
  - Just to make sure, remove `assign`s again after placement:
    - `delete_assigns -add_buffer`

```
assign b = a ;
```

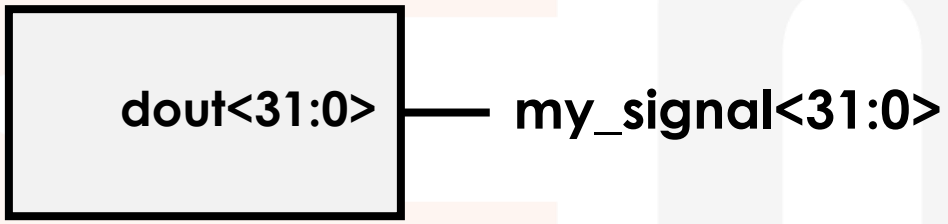


Problem #3: Bus Notation

- **Issue:**
  - Verilog uses square brackets for vectors

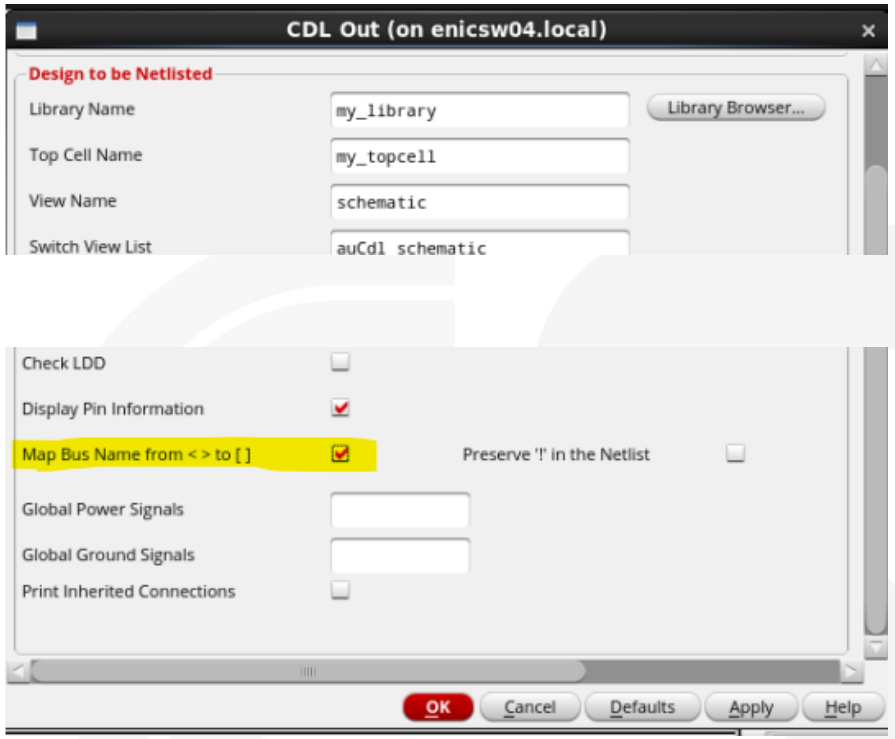
```
my_memory memory (.dout(my_signal[31:0]));
```

  - But Virtuoso uses triangular brackets…



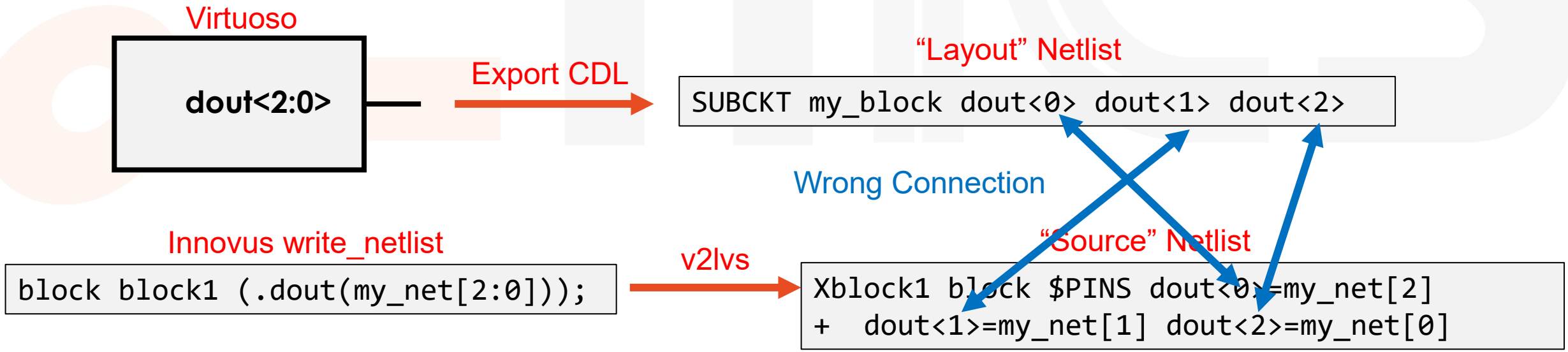
  - In the year 2020, this can still confuse the EDA tools…
- **Solution:**
  - When exporting your CDL from Virtuoso, select

"Map Bus Names from <> to [ ]"



Problem #4: Flipped Busses

- **Issue:**
  - Digital tools use Verilog and often consider busses as multi-bit vectors.
  - Analog (circuit) tools use SPICE and don't necessarily use vectors.
  - Both languages support connectivity by position, which can lead to mismatches.
- **Example of Flipped Bus:**



Problem #4: Flipped Busses (ctnd.)

- **Solution:**
  - Connect independent signals and not busses
  - In Innovus, use the `-flatten_bus` option of `write_netlist`:

```
write_netlist -phys -flatten_bus
```

Innovus write_netlist –flatten_bus

```
block block1 (.dout[2](my_net[2]),
   .dout[1](my_net[1]),
   .dout[0](my_net[0]));
```

v2lvs

“Source” Netlist

```
Xblock1 block $PINS dout<0>=my_net[0]
+  dout<1>=my_net[1] dout<2>=my_net[2]
```

Problem #5: Excluded Instances

- **Issue:**
  - Some “Physical Cells” do not have a corresponding CDL in the library.
  - These include (among others):
    - Fillers (not Well Taps!)
    - IO Fillers
    - Corner Ios
    - Bond Pads
  - Therefore, when running v2lvs, the tool cannot translate these to SPICE.

```
FILLER2 FILL123 (
    );
CORNER  TOP_LEFT_CORNER (
    );
```

- **Solution:**
  - Option 1: Use the `-exclude_insts_of_cells` option of `write_netlist`.
  - Option 2: Post-process them away (e.g., `sed -i -e ‘/FILLER/,+1d’`)
  - Option 3: Create empty SUBCKT definitions

```
.SUBCKT FILLER2
.SUBCKT CORNER
```

Translating the Verilog netlist into SPICE

Gathering your CDLs

- **Innovus has provided us with a Gatelevel Verilog netlist, but LVS runs on SPICE netlists.**
- **Where do we get the SPICE netlists from?**
  - They're part of the library, of course
  - We want to use the "CDL" file, which is the one without post-layout parasitics!
- **So we need to create an include file that references the CDLs of:**
  - The standard cells
  - The IOs
  - The Compiled Memories
  - Any other Hard Macros
  - Any hierarchical blocks that passed LVS standalone (but you should flatten these for final LVS)

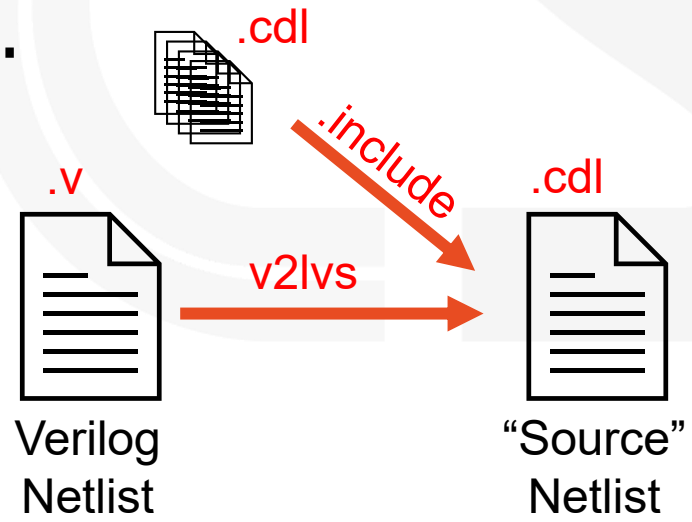
```
.INCLUDE /path/to/Standard_cells.sp
.INCLUDE /path/to/IOs.sp
.INCLUDE /path/to/SRAM1.sp
.INCLUDE /path/to/Custom_block.cdl
```

v2lvs

- **The Mentor tool for converting Verilog to SPICE is called v2lvs:**

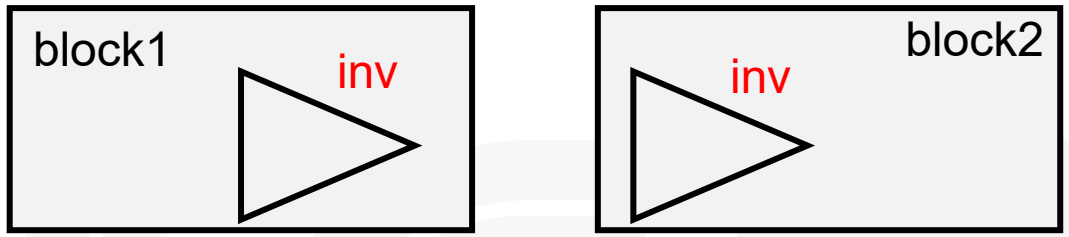
```
v2lvs -sn   -v my_verilog.v    -o my_output_cdl.cdl
      -lsr my_includes_file.sp -s my_includes_file.sp
```

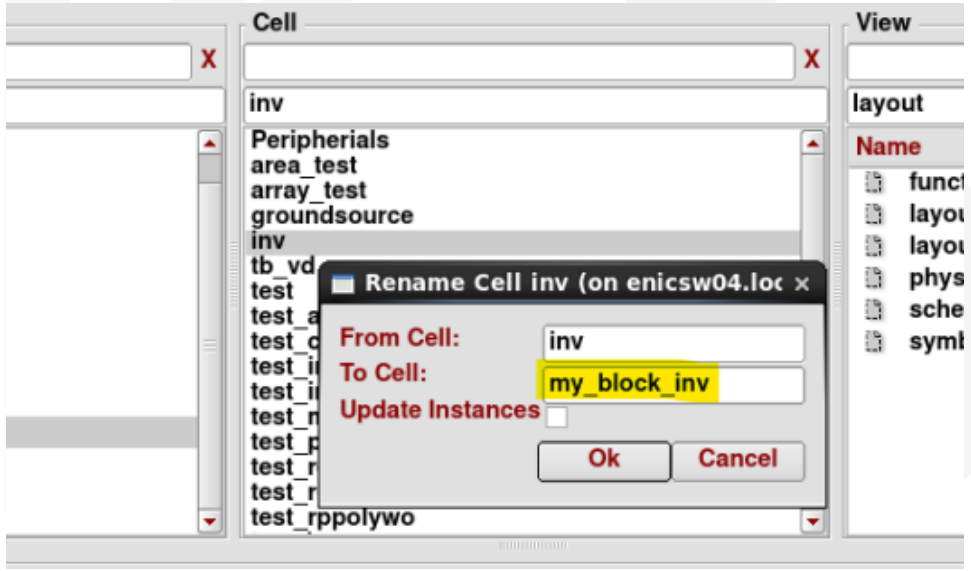
- **But, as usual, this comes with a bunch of problems…**
  - Duplicate Subcircuits
  - Globals (e.g., Vendor provided IOs)
  - Bulk Connections (e.g., Standard Cells)
  - Ports that are shorted outside the block (e.g., on the board)



Problem #1a: Duplicate Custom Subcircuits

- **Issue:**
  - When creating a custom block, we may have a cell with the same name as a cell in some other block in the chip.
- **Solution #1:**
  - In Virtuoso - give your custom instances a unique name by adding a prefix or suffix.
- **Solution #2:**
  - Post-process your CDL to give subcircuits unique names by adding a suffix.






```
grep ".SUBCKT *" ${BLOCK_NAME}.sp | cut -d " " -f 2 > uniquify.lst
sed -i "/${BLOCK_NAME}/d" uniquify.lst
sed -i "s/.*/s\/&\/&_tile_${BLOCK_NAME}\/g/g" uniquify.lst
sed -f uniquify.lst ${BLOCK_NAME}.sp > ${BLOCK_NAME}.unique.sp
```

Problem #1b: Duplicate Digital Subcircuits

- **Issue:**
  - When running a bottom-up hierarchical flow, the RTL or EDA tools may (will!!!) create modules with the same name.
- **Solution #1:**
  - Tell Genus to give your modules a unique name:
  - Tell Genus to rename modules before netlist export
  - Tell Innovus to change module names before netlist export

```
set_attr gen_module_prefix "my_block"
```

```
foreach module [get_db modules] {
        set name [get_db $module .base_name]
        rename_obj $module "my_block_$name" }
```

```
update_names -module -suffix/prefix “my_block”
```

- **Solution #2:**
  - Post-process your CDL to give subcircuits unique names by adding a suffix.

Problem #2: GLOBALs

- **Issue:**
  - A global signal in SPICE (`.GLOBAL`) propagates to the entire design and **takes priority** over local signals with the same name.
  - To clarify this, if you have VDD as a global signal in a block CDL any internal net called VDD will be connected to this signal.

```
.GLOBAL VDD
SUBCKT annoying_block
M1 VDD VDD VDD VDD NMOS
.ENDS
```

- **Solution:**
  - Don't use GLOBAL signals!

```
.SUBCKT BLOCK VDD
M1 VDD VDD VDD VDD nmos
.ENDS

.SUBCKT my_chip VDD1 VDD2
XBLOCK1 BLOCK $PINS VDD=VDD1
XBLOCK2 BLOCK $PINS VDD=VDD2
XANNOYING annoying_block
.ENDS
```

Example: IOs for 65nm

- **Unfortunately, the IOs we have for 65nm *do use* global signals**

```
.GLOBAL VDD VSS VDDPST POC
```

  - This is a huge pain in the neck
- **Removing the Globals isn't enough:**
  - CDL → No port connections to the relevant subcircuits.
  - Verilog Netlist →Since the IOs don't have any port connections for the globals, the Verilog netlist is exported without these connections.
  - LEF → If the LEF would have power connections, then they would be connected in Innovus and exported in the Verilog netlist.
- **Solution**
  - Modify the LEF and CDL of the IOs

Problem #3: Bulk Connections

- **Issue:**
  - A transistor has a bulk terminal.
  - The standard cell LEF *may not* have a bulk terminal
    → there is no connection to the bulk for gates in Innovus
    → the exported Verilog netlist has no bulk connections
- **This leads to two major problems:**
  - The standard cell SPICE (CDL) views *have to have* bulk connections
    → They are incompatible with the gate instantiation in the Verilog netlist.
  - If we would globally define a bulk connection to VDD/GND, this wouldn't support power domains, body biasing, special power nets.

Problem #3: Bulk Connections (ctnd.)

- **Solution to problem #1:**
  - Use the `-addpin` option in v2lvs:

```
v2lvs -sn -v my_verilog.v -o my_output_cdl.cdl
      -s my_includes_file.sp -addpin VPW -addpin VNW
```

  - This adds a connection with the same name as the pin, i.e.:

```
XCELL INVX1 $PINS A=in Z=out VDD=VDD VSS=VSS VPW=VPW VNW=VNW
```

  - This is okay for a single bulk bias, but not if several bulk biases are used.
- **Solution to problem #2:**
  - Post-process the CDL to connect the correct VPW/VNW.
  - But it is much better and safer to

    MODIFY THE LEF!

Problem #4: Shorted Ports

- **Issue:**
  - Sometimes ports with different names are connected to each other – either at the board level or even for macro-level LVS.
  - For example, if separate VSS bulk connections are used, but they are connected through the substrate or to propagate the VNW/VPW signals from the previous slide.
- **Solution #1:**
  - Use the `*.CONNECT` statement

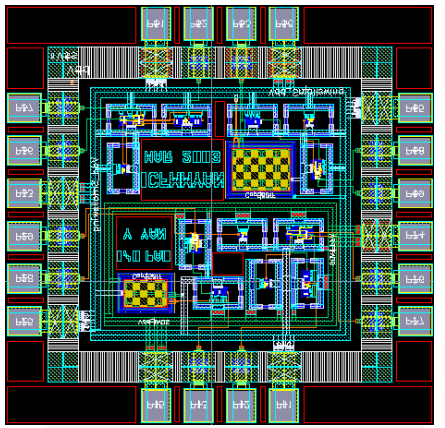
```
*.CONNECT VDD VNW
```

Note that this connects VNW to VDD so that VDD propagates through the circuit. If you switch the order, VNW will propagate through and your circuit will not pass LVS!

- **Solution #2 (recommended):**
  - Make a Wrapper that connects the two nets and run LVS on the wrapper.

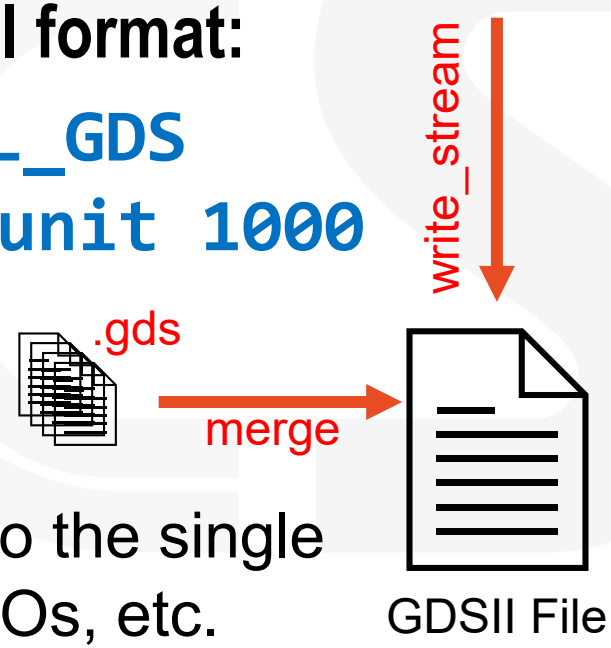
Extracting the LVS-ready Layout Netlist

Streaming out from Innovus



- **Now that our "Source" Netlist is ready, we need to prepare the "Layout" Netlist.**
- **We start by exporting the layout from Innovus in the GDSII format:**

```
write_stream my_layout.gds -merge $ALL_GDS
    -mode NOFILL -map_file $mapfile -unit 1000
```



- **A few options to know about here…**
  - `-merge $ALL_GDS`: Merges the GDSII files of macros into the single output GDS file. Make sure you have all standard cells, IOs, etc.
  - `set_db write_stream_cell_name_prefix`: Add a prefix for unique naming
  - `set_db write_stream_text_size`: Sets the size of labels for readability

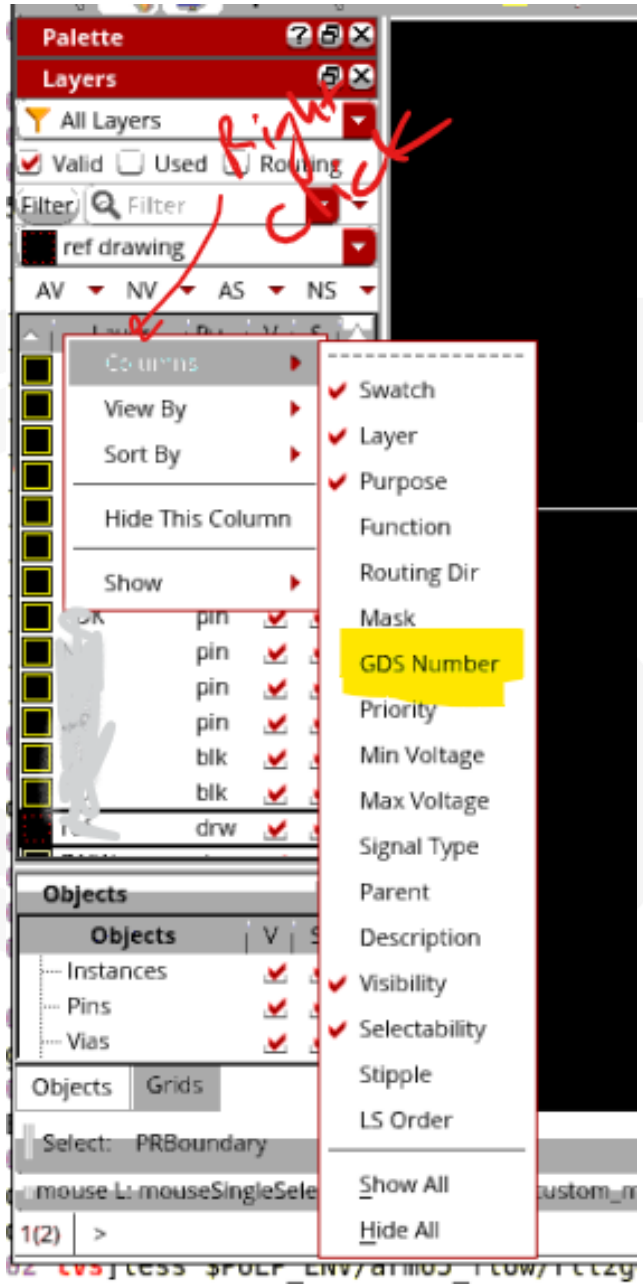
The Mapping File

- There are various types of GDS map files in EDA tools, which is confusing, but they all basically translate a layer name to its use and layer number.
- The StreamOut MapFile used by Innovus and Virtuoso is a simple table:

```
M1          drawing     15          0
M1          PIN         15          32
```

Layer Name Layer Type Layer Number Data Type

- It can be important to find the layer numbers for various purposes. One way is to turn on the GDS number in Virtuoso's LSW.

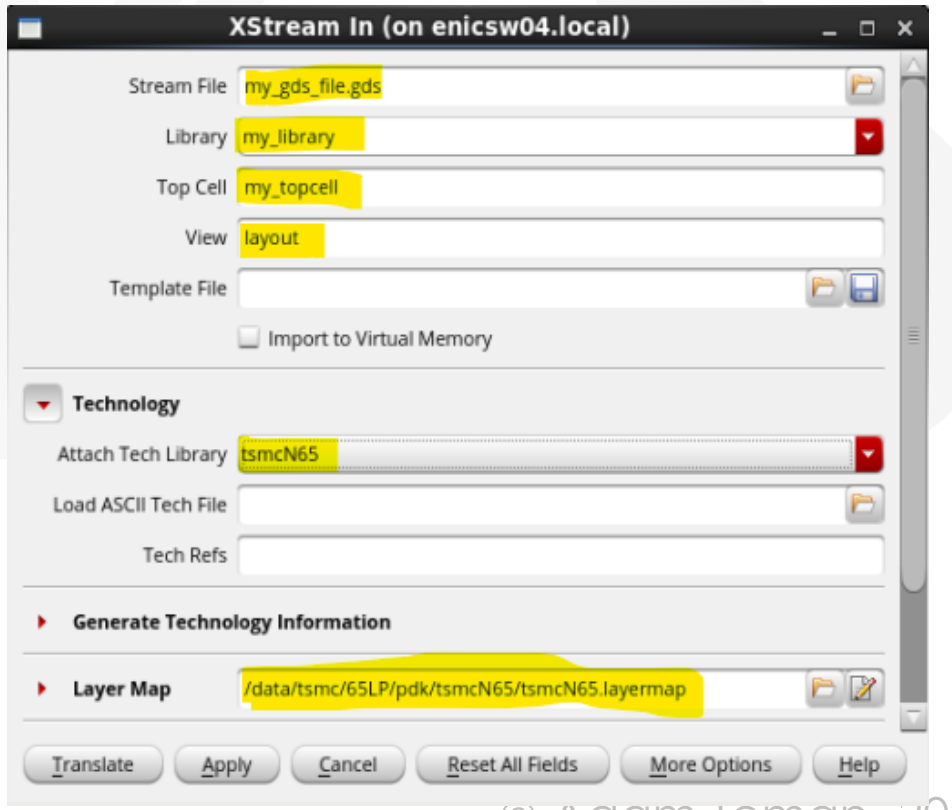


Streaming Into Virtuoso

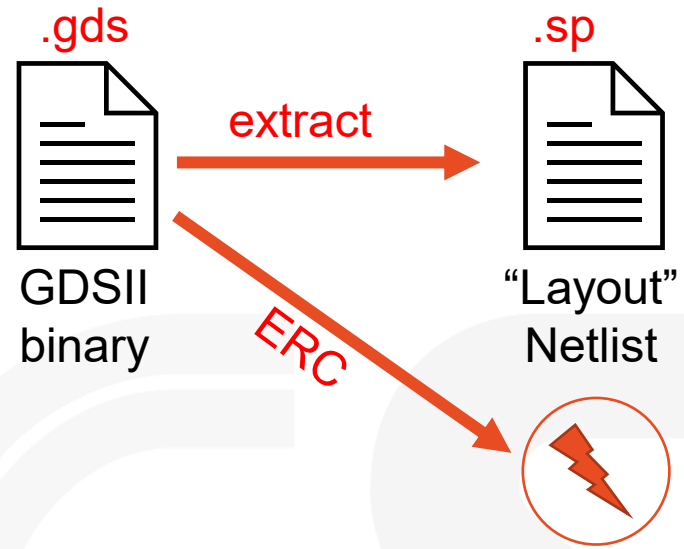
- **Streaming out from Innovus isn't enough**
  - What if we want to visually debug LVS or manually fix DRCs?
- **We will use the same mapping file to stream in to Virtuoso**

```
strmin -topCell my_topcell \
  -library my_library -view layout \
  -attachTechFileOfLib techfile \
  -layerMap /path/to/layermap \
  -strmFile my_gds_file.gds
```

- **Or with the GUI:**
  - File→Import→Stream



Netlist Extraction



- So now we have the GDS and we have to extract the devices and connectivity to create the Layout Netlist.
- We can use Calibre to do the extraction:

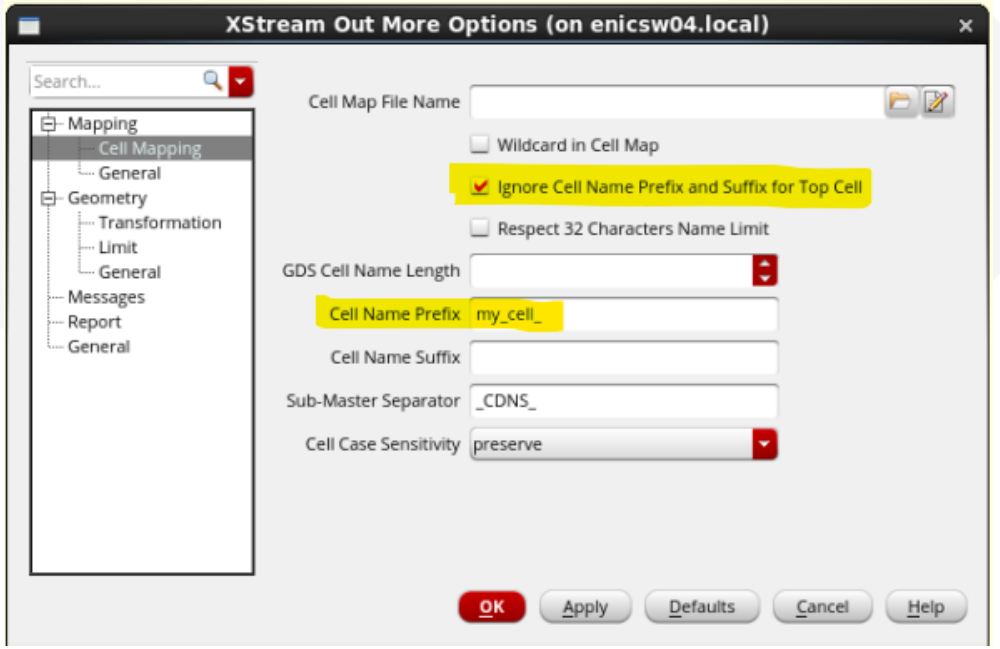
```
calibre -hier -64 -hyper -turbo \
    -spice my_layout_netlist.sp  /path/to/runset.extract
```

- The runset file tells the tool how to run the extraction.
- First and foremost, this includes the path to the GDS file:

```
LAYOUT PATH "$MY_GDS"
LAYOUT PRIMARY "my_toplevel"
LAYOUT SYSTEM GDSII
```

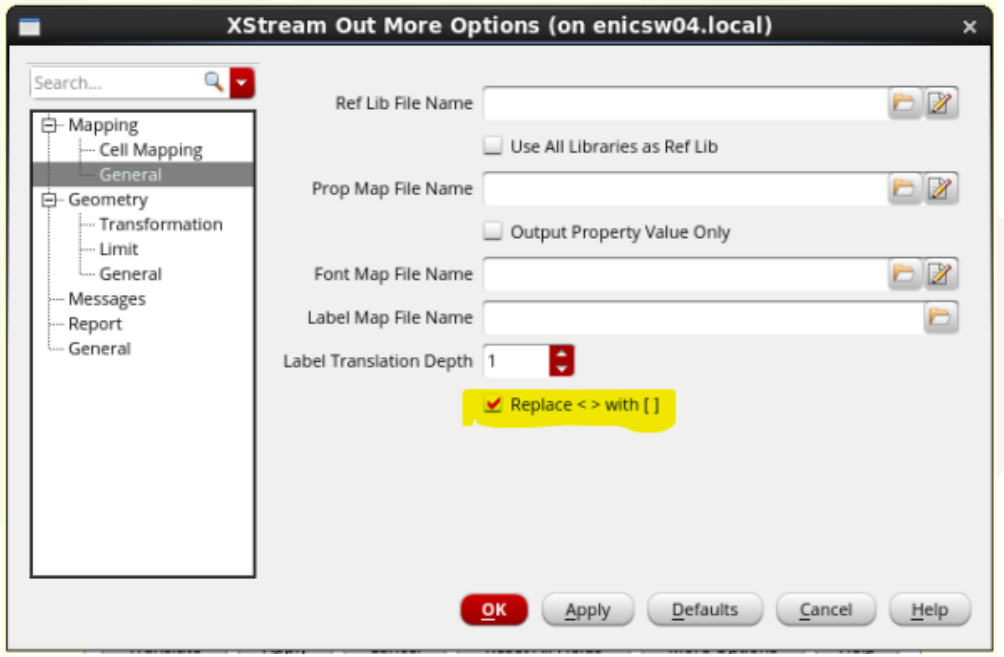
Problem #1: Duplicate Instances

- **Issue:**
  - Merged GDS files have cells with the same name as other cells.
- **Solution for digital blocks:**
  - Use `set_db write_stream_cell_name_prefix` to add a prefix to streamed cells.
- **Solution for custom blocks:**
  - Add a Cell Name Prefix on the **XStream Out More Options** form.
  - Make sure you also check the "Ignore Cell Name Prefix and Suffix for Top Cell" option



Problem #2: Bus Notation

- **Issue:**
  - Custom cell busses use triangular brackets < >, which are streamed out in the GDS.
- **Solution #1:**
  - Choose "Replace < > with [ ]" on the **XStream Out More Options** form.
- **Solution #2:**
  - Change the < > labels to [ ] in the **extraction runset file**




```
LAYOUT RENAME TEXT
                "/>/]/"
                "/</[/"
```

Problem #3: Missing Ports

- **Issue:**
  - Sometimes you are missing a label in your GDS
  - This will immediately cause a “port mismatch” in the LVS report
- **Solution #1:**
  - Add it in Innovus. Not that easy, but the better solution.
- **Solution #2:**
  - In the **extraction runset** file, use the **`LAYOUT TEXT`** command:

```
LAYOUT TEXT VDDPST   1037 1916    137
LAYOUT TEXT POC       188 1360    133
```

Label (Port) Name

Label Coordinate (X,Y)

Layer Number

The GDS Layer number of the PIN layer.

Problem #4: Multiple Labels

- **Issue:**
  - Multiple labels of the same name appear on the layout.
  - For example: VDD is on every VDD pad.
  - This is called a "Stamping Conflict" and will appear as a "Short Circuit" warning or error in the log file.
- **Solution:**
  - Use `Virtual Connect Colon` to connect labels with colons (e.g., `VDD:`)
  - Use `Virtual Connect Name` to connect labels with the same name (The option `?` connects all nets with the same name)

```
VIRTUAL CONNECT COLON YES
VIRTUAL CONNECT NAME  ?
```

Problem #5: Special Power Net Names

- **Issue:**
  - You use a non-standard name (i.e., not VDD, VSS, GND…) to tap your bulks.
  - Calibre will warn you that this the bulk is not connected to Power or Ground.
- **Solution:**
  - Set the `LVS POWER NAME` and `LVS GROUND NAME` commands in the **extraction runset file**.

```
LVS POWER  NAME “VDD” “VDD1” “VDDIO”
LVS GROUND NAME “VSS”
```

Problem #6: Blocks that aren't Ready

- **Issue:**
  - You want to start to setup and debug LVS, but you don't have all your IPs.
  - Calibre will error out during extraction.
- **Solution:**
  - Set the following command in the runset file:

```
LAYOUT INPUT EXCEPTION SEVERITY MISSING_REFERENCE 1
```

  - Now, extraction will run, but you can't pass LVS, of course.
  - We will see how to exclude, filter, or box to try to get around this problem later.

Problem #7: Adding additional structures

- **Issue:**
  - You often need to add additional physical structures to the final GDS, such as a LOGO, Seal Ring, Dummy Fill, etc.
- **Solution #1:**
  - Create a LEF for your physical structure and add it in Innovus.
- **Solution #2:**
  - Merge the GDS of the physical structure with the toplevel GDS.
  - For example, using Calibre DRV:

```
calibredrv -a layout filemerge -append -createtop "my_toplevel" \
                    -in my_logo.gds -out my_logo_for_merging.gds
calibredrv -a layout filemerge -append -topcell "my_toplevel" \
                    -in my_toplevel.gds -in my_logo_for_merging.gds \
                    -out my_toplevel_with_logo.gds
```

Running LVS

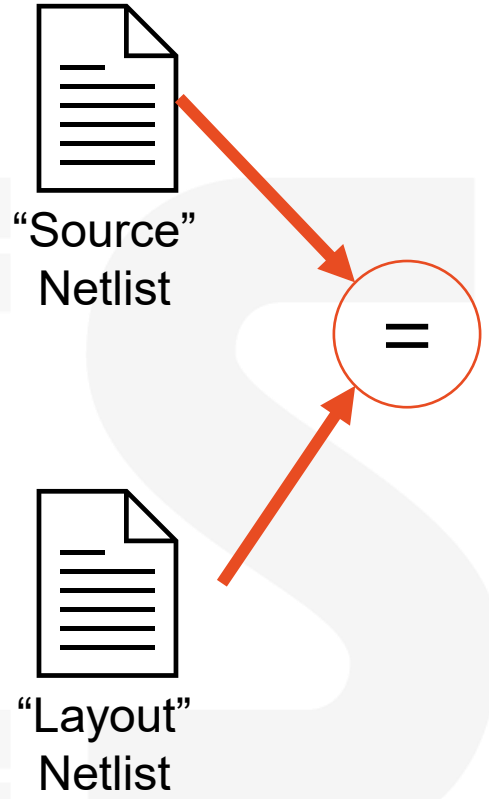
Emerging Nanoscaled Integrated Circuits and Systems Labs
The Alexander Kofkin Faculty of Engineering
Bar-Ilan University

Running LVS

- So now we have the both the Source and the Layout netlists.
- We can use Calibre to run the comparison:

```
calibre -hier -64 -hyper -turbo \
                    /path/to/runset.compare
```

- Here, too, we have a runset file tells the tool how to run the comparison.
- First and foremost, this includes the paths to the two netlists:




```
LAYOUT PATH "$MY_LAYOUT_NETLIST"
LAYOUT PRIMARY "my_toplevel"
LAYOUT SYSTEM SPICE

SOURCE PATH "$MY_SOURCE_NETLIST"
SOURCE PRIMARY "my_toplevel"
SOURCE SYSTEM SPICE
```

LVS Principles

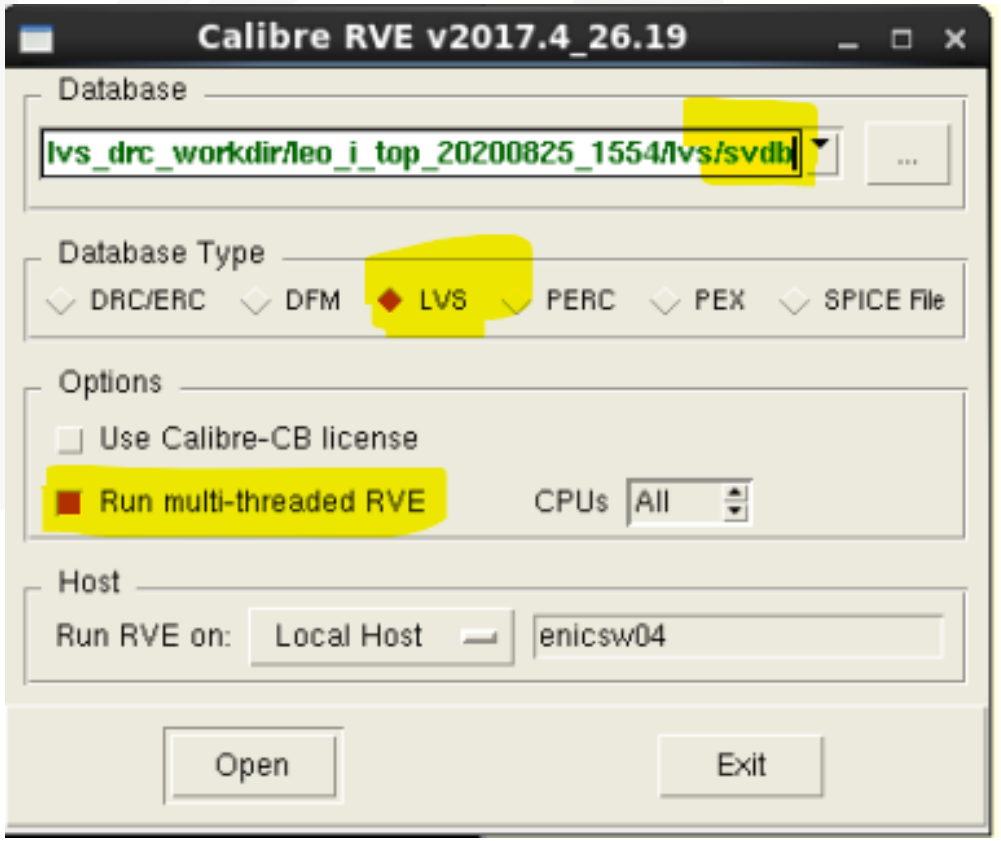
- **Before trying to understand what's wrong, let's understand the basic mechanics of Source vs. Layout comparison:**
  - LVS is a formal verification. In other words, it's black or white – the two netlists are entirely equivalent or not. There is no "middle ground", and this makes it hard to debug.
  - The anchors are the toplevel ports. The comparison is told (by us!) that port X in the Source is port X in the layout. This is the "ground truth". Therefore, it can start searching for equalities starting at the ports.
  - If the design is not equivalent, the tool tries to give a "best guess". Unfortunately, this guess is not always that great… However, it usually includes many different hints that can point us to fundamental problems.

The LVS Results Database

- The LVS results database is usually dumped into a folder called "svdb"
- You can open it from Calibre RVE:

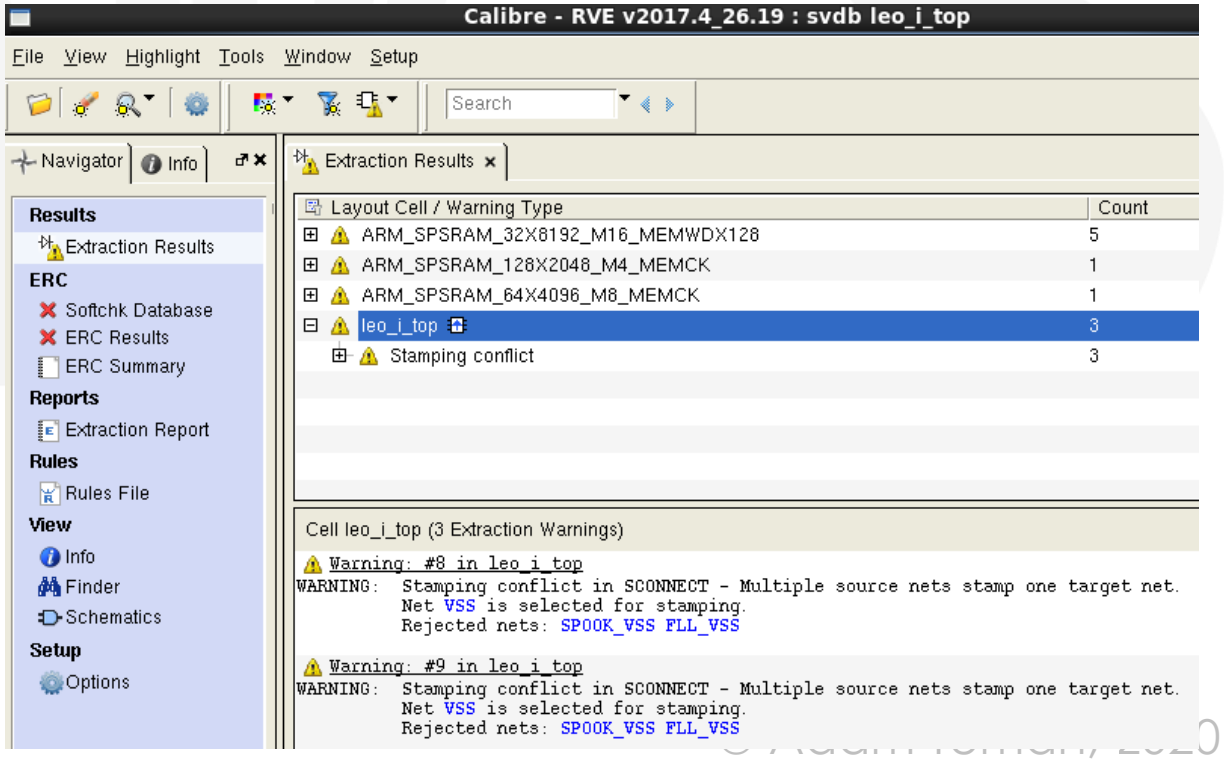
```
calibre -rve svdb/
```

- But to get connectivity to layout, open it from the Virtuoso plugin
- This will also display the ERC results that were created during extraction.



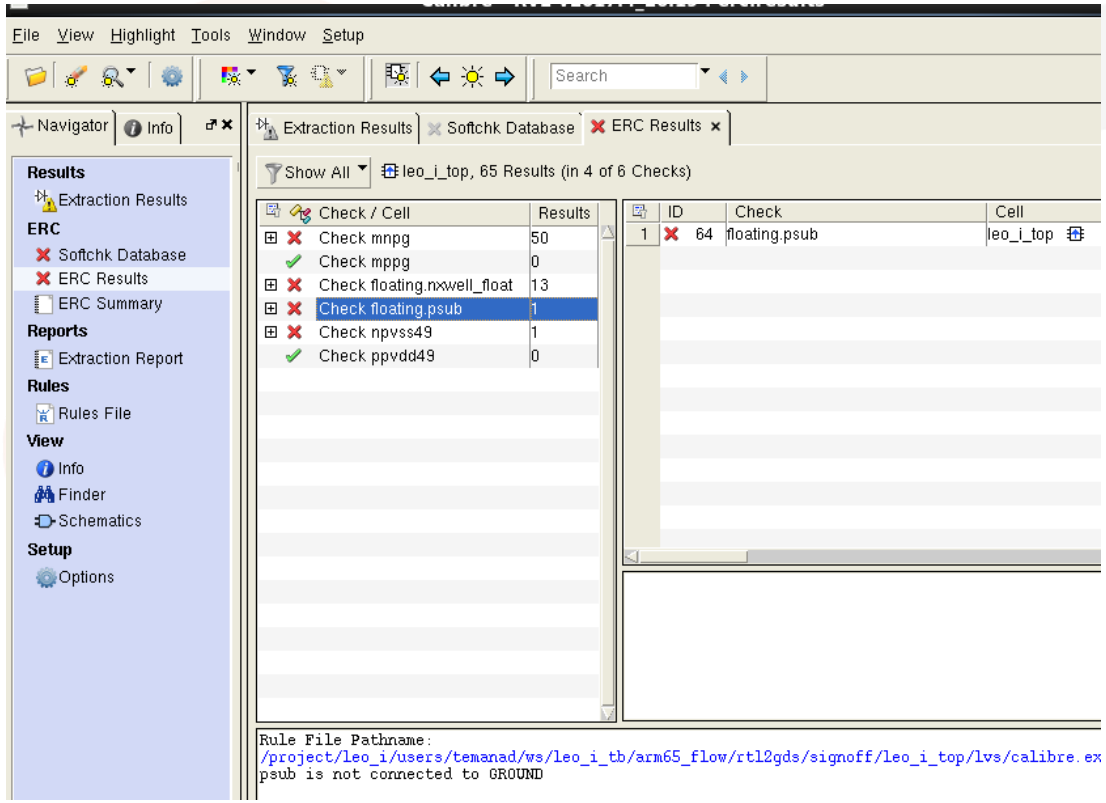
Electronic Rules Check (ERC)

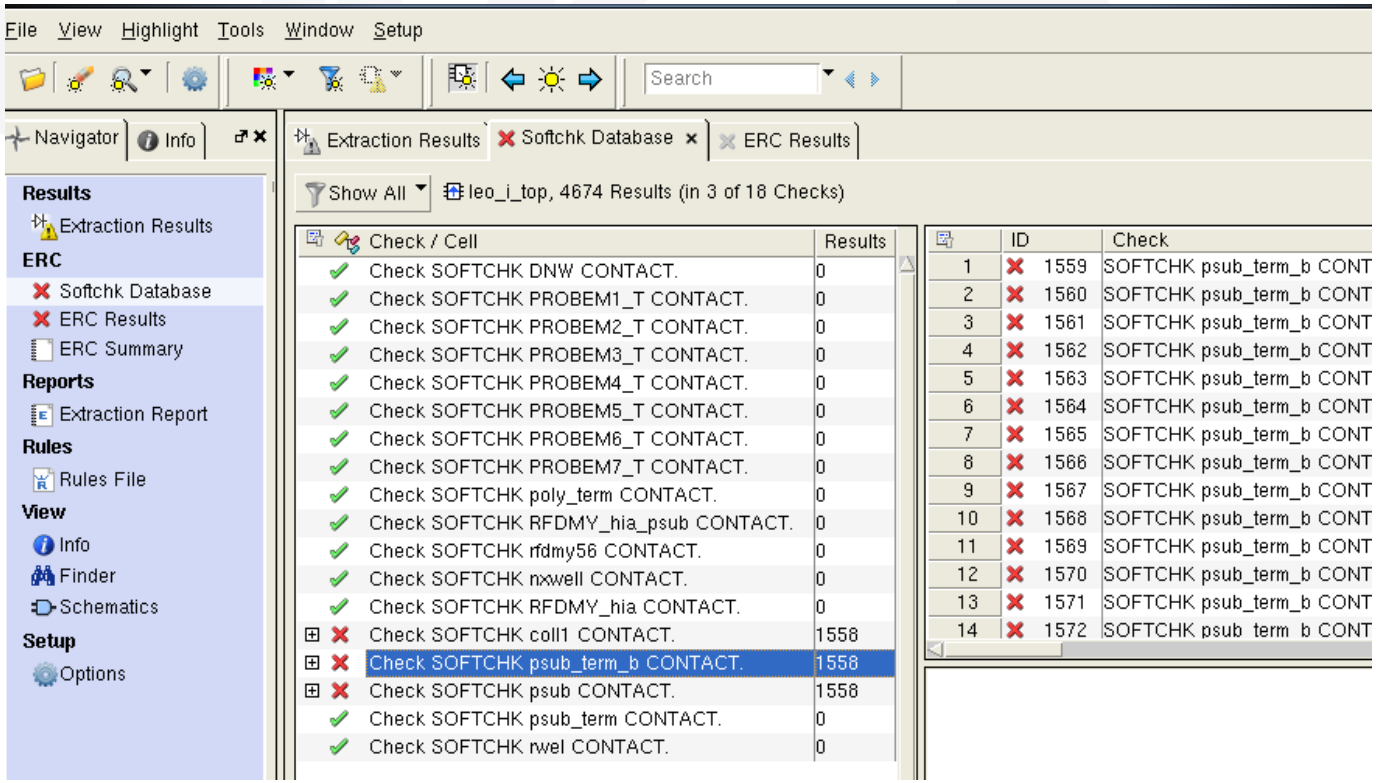
- **ERC is run during the extraction stage, but you can view the results along with the comparison database results.**
- **ERC checks tell you about things like short circuits, open circuits and problems with bulk connections (often called "Stamping Conflicts")**
- **First, check the extraction log file for warnings and try to fix them.**
  - For example, "Short Circuits" will be reported if two ports (e.g., VDD and VSS) are connected somewhere.
  - Sometimes, warnings in hierarchical blocks can be ignored…



Electronic Rules Check (ERC)

- **Next, check the ERC report in RVE.**
  - Highlight errors to understand where problems (such as missing bulk connections) occur.





LVS Debugging Principles

- **Port Comparison**
  - If your ports are not equivalent, don't look any further!
- **Hierarchy Comparison**
  - Make sure lower level hierarchies pass LVS and ERC.
  - Make sure the connections to these hierarchies are correct.
- **Bulk Connections**
  - The bulks are pretty clear – usually VSS for NMOS, VDD for PMOS.
  - If some transistor in one of the netlists is connected to something else, that is probably a good place to start looking for problems…
- **Device Connections**
  - "Best guesses" usually try to match devices by the most equivalent pin connections. If 3 out of 4 pins are connected correctly, try to look at the 4th pin.

Tip #2: HCells

(*Tip #1 was to get your ports right!*)

- **As previously mentioned, *Hierarchical Comparison* is really helpful.**
- **To get this to run, provide a file called "hcells" that details all hierarchical blocks**
  - e.g., soft hierarchies, hard macros
- **Really stupid file, it's just is the name of the block… twice!**

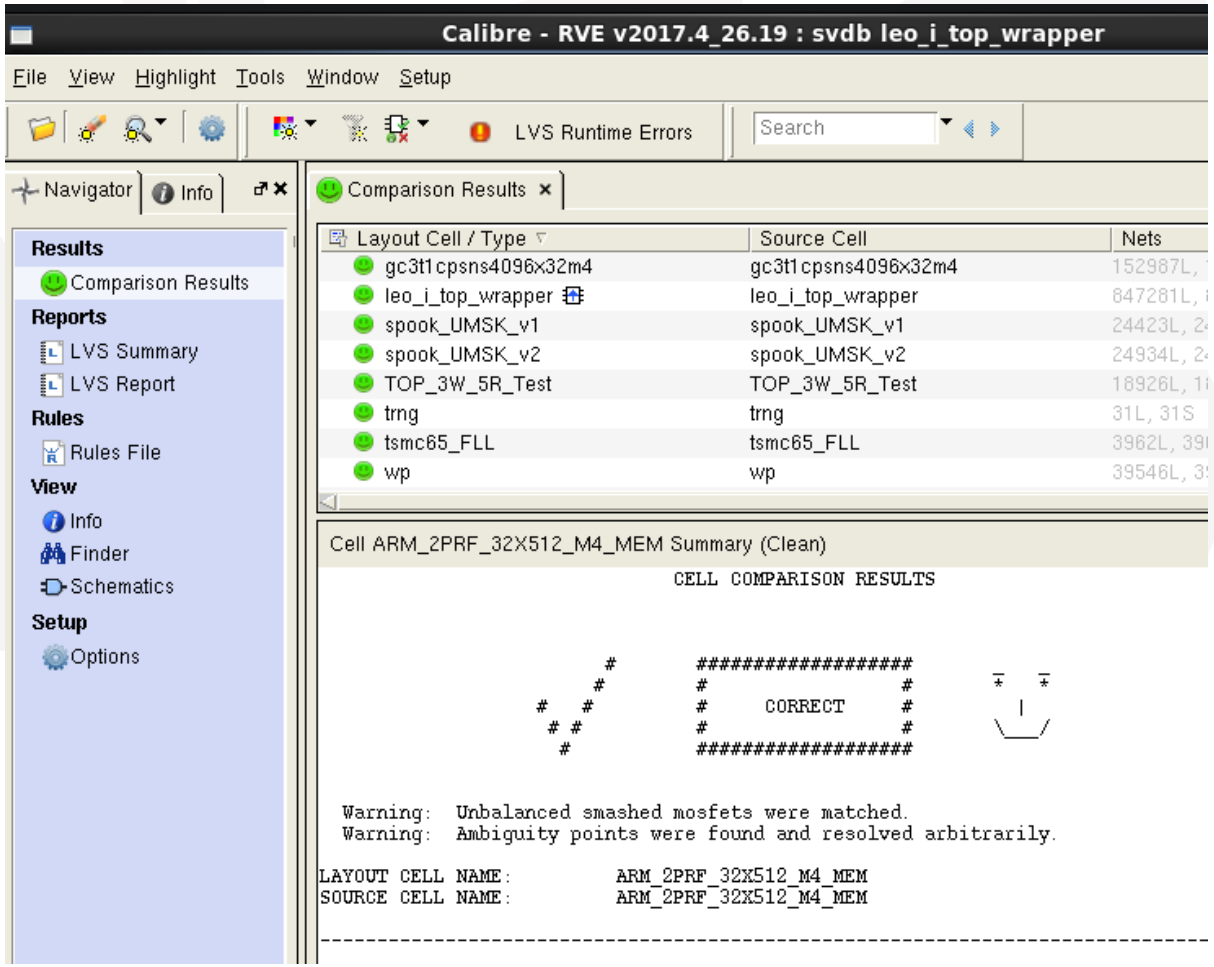
```
my_block1 my_block1
my_block2 my_block2
my_block3 my_block3
```

- **Then run LVS with the `-hcells` option:**

```
calibre -hier -64 -hyper -turbo –hcells hcells.txt \
                    /path/to/runset.extract
```

Tip #2: HCells (ctnd.)

- **If you run LVS with the hcells option, each hierarchical block will be compared and reported separately.**
- **Obviously, all of the blocks must pass standalone LVS before integration!**
- **So if there is an error here, it probably is due to connections to the block**
  - Often VDD/GND connections, since signal connections shouldn't cause block level LVS to fail…
  - In any case, these can help pinpoint the problem…



Tip #3: Excludes, Filters, Blackboxes

- **We can use the `EXCLUDE`, `FILTER`, or `BOX` commands to remove a block from our comparison for:**
  - When a block is not quite ready
  - To help debug, especially when we get hcell LVS errors
  - When we don't have the GDS or CDL of a block (such as libraries without backend views)
- **To understand the differences, read the Calibre manual, but the syntax is:**

```
LVS SPICE EXCLUDE CELL LAYOUT MY_MISSING_BLOCK
```

```
LVS FILTER MY_PROBLEMATIC_BLOCK OPEN BOTH
```

```
LVS BOX MY_PROBLEMATIC_BLOCK
```

Tip #4: Bulk Connections

- **A good place to search for problems is in the bulk connections.**
- **Start by running DRC**
  - DRC can highlight a lot of unexpected errors, such as NWELL overlaps, missing well taps, etc.
- **Then go to the ERC report**
  - Are there warnings about short circuits or stamping conflicts with VDD/GND?
  - Check “soft connect” errors and highlight to see what’s connected wrong.
- **Finally, look at the device connections in the LVS report**
  - Your bulks should be connected to VDD (PMOS) or GND (NMOS)
  - If this is not the case (e.g., bulk is connected to net 1234), try to understand why.

Tip #4: Bulk Connections (ctnd.)

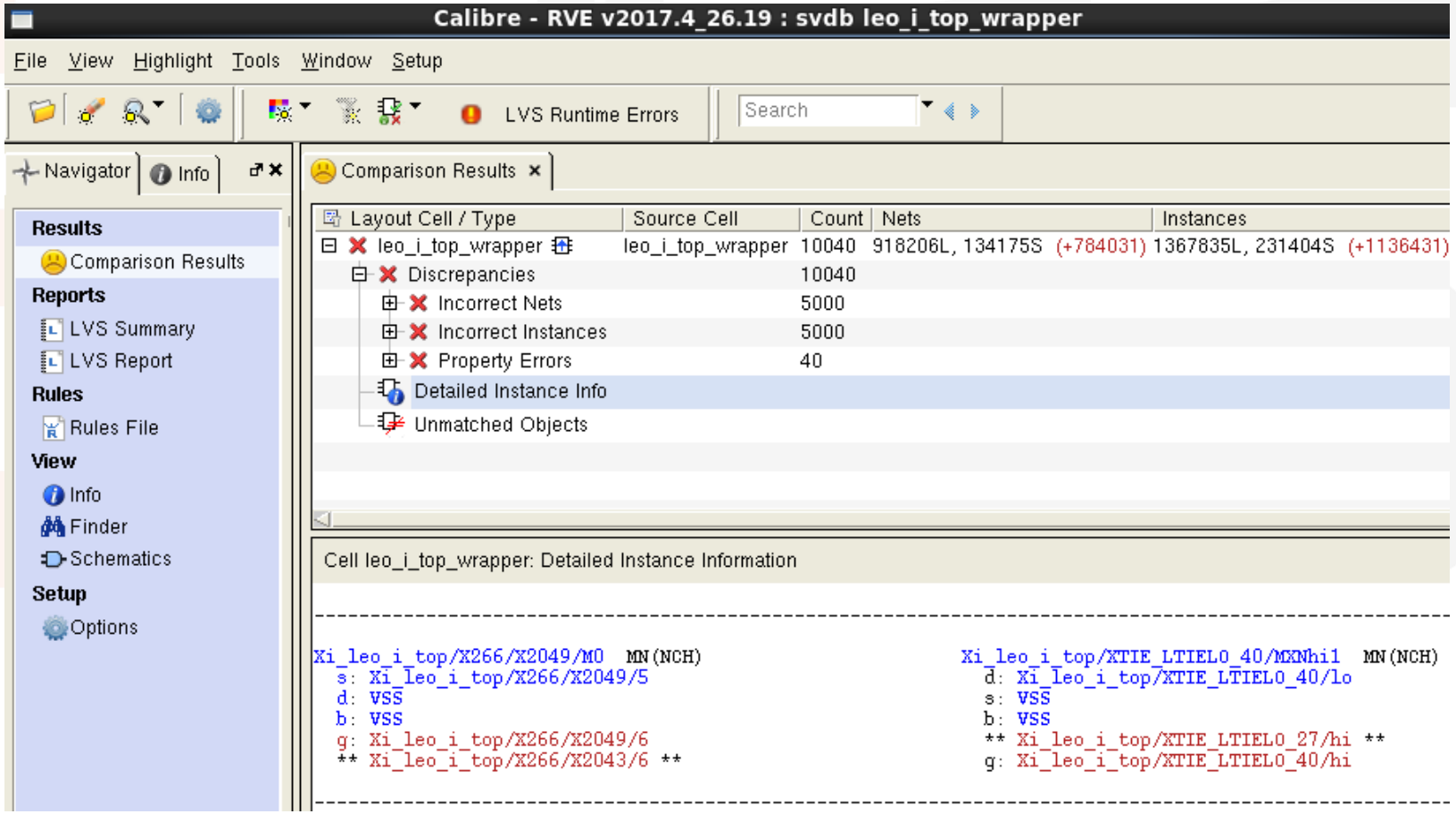
Examples of things that can cause problems with bulk connections:

(*note, all of these occurred in LEO-I*)

- **Forgot to add fillers**
- **Fillers added where they weren't supposed to be**
- **Forgot to add Well Taps or wrong Well Tap cell used**
- **Wrongly connected Well Taps**
- **Macro overlaps**

Tip #5: Device Connections

- The last (very weak) general tip I can give is to look at the "detailed instance info" section of RVE.
- Sometimes you can tell by the names of the nets that something is connected wrong, such as a flipped bus.
- But in general, this is when you really start to scratch your head…



Fullchip DRC and Chip Finishing

Emerging Nanoscaled Integrated Circuits and Systems Labs

The Alexander Kofkin Faculty of Engineering
Bar-Ilan University

The Chip Finishing Flow

- Adding required structures to your Layout
- Design Rule Check (DRC)
- Density Fill
- Antenna Violation Check
- LVS after density fill and DRC fixes
- Post-fill signoff timing in Tempus

Adding required structures to your Layout

- **Sometimes you need to add special structures to your layout, e.g.:**
  - Bond pads or bumps
  - Seal Ring
  - Logo
  - Fiducial
- **In general, I recommend making a LEF of these and adding them in Innovus.**
- **But if you need to add a GDS:**
  - **Method 1**: Create a layout wrapper in Virtuoso and instantiate the toplevel and the added layers. Then stream it out (or run LVS from the GUI)
  - **Method 2**: Merge the GDS files with Calibre DRV

Example: Adding a Logo with Calibre DRV

- **First, create your logo with a layout editor**
  - For example, use the AP layer (or LOGO layer)
- **Next, create a toplevel that instantiates your logo:**
  - In Virtuoso, create a cellview called <my_toplevel>, instantiate your logo at the coordinates according to the real toplevel origin, and streamout.
  - Or do this to the logo GDS using Calibre DRV:

```
calibredrv -a layout filemerge -append -createtop "my_toplevel" \
           -in my_logo.gds -out my_logo_for_merging.gds
```

- **Finally, merge the logo into the GDS with Calibre DRV:**

```
calibredrv -a layout filemerge -append -topcell "my_toplevel" \
           -in my_toplevel.gds -in my_logo_for_merging.gds \
           -out my_toplevel_with_logo.gds
```

Design Rule Check (DRC)

- In general, this is the same as running DRC from the Calibre Virtuoso Plugin, but you can run from the command line to automate the process:

```
calibre -drc -hier -turbo -turbo_litho -hyper –nowait \
                                    /path/to/runset.nmDRC
```

- Here, too, the runfile that points to the GDS file:

```
LAYOUT PATH "$MY_GDS"
LAYOUT PRIMARY "my_toplevel"
LAYOUT SYSTEM GDSII
```

- Also, there are DEFINES that are usually described in the *readme* comments at the top of the rulefile, e.g.:

```
#DEFINE "FULL_CHIP"
```

- Finally, you can filter out DRC errors that are irrelevant at this stage, using the `DRC UNSELECT CHECK` command:

```
DRC UNSELECT CHECK “M1.DN.1”
```

Density Fill

- **Density fill is required for all scaled process technologies to reduce variation in the CMP steps and other process steps.**
  - In older processes (e.g., 65nm), a density of around 30%-70% is required on all layers.
  - In newer processes, there may be additional rules, such as special structures to improve photolithography.
- **Density fill is added by:**
  - Using the internal functionality of the Place and Route tool.
  - Or (more commonly) by using the DRC tool with a special rulefile.

```
DRC RESULTS DATABASE “my_density_fill.gds” GDSII PREFIX "FILL_"
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX 4096
```

  - You can then use Calibre DRV to merge the results with the toplevel GDS.

Antenna Checks

- **Antenna violations occur when the ratio between manufactured interconnect layers and the connected gate oxide is so large that the charge that is built up during manufacturing will burn out the oxide.**
- **Two solutions to antenna violations:**
  - Metal bridging – should be automated by the place and route tool
  - Antenna diodes – often used to fix antenna violations in ECO.
- **How to check antenna violations:**
  - During the place and route flow.
    But this is not sufficient due to missing information in the LEFs!
  - Using the DRC tool with a special runset.

And usually a bit more…

- **There are usually many other things to do as part of backend signoff.**
  - For example, there may be a bonding rule DRC run
- **It is also very highly recommended to run final signoff STA (Tempus/PrimeTime) after adding Density Fill.**
- **And always make a "dry run tapeout" enough time before the real tapeout date so you don't run into any unexpected problems.**

- **Good luck, and most importantly, start running DRC/LVS really early!**