Preparing a custom block for digital-on-top integration

Prof. Adam Teman

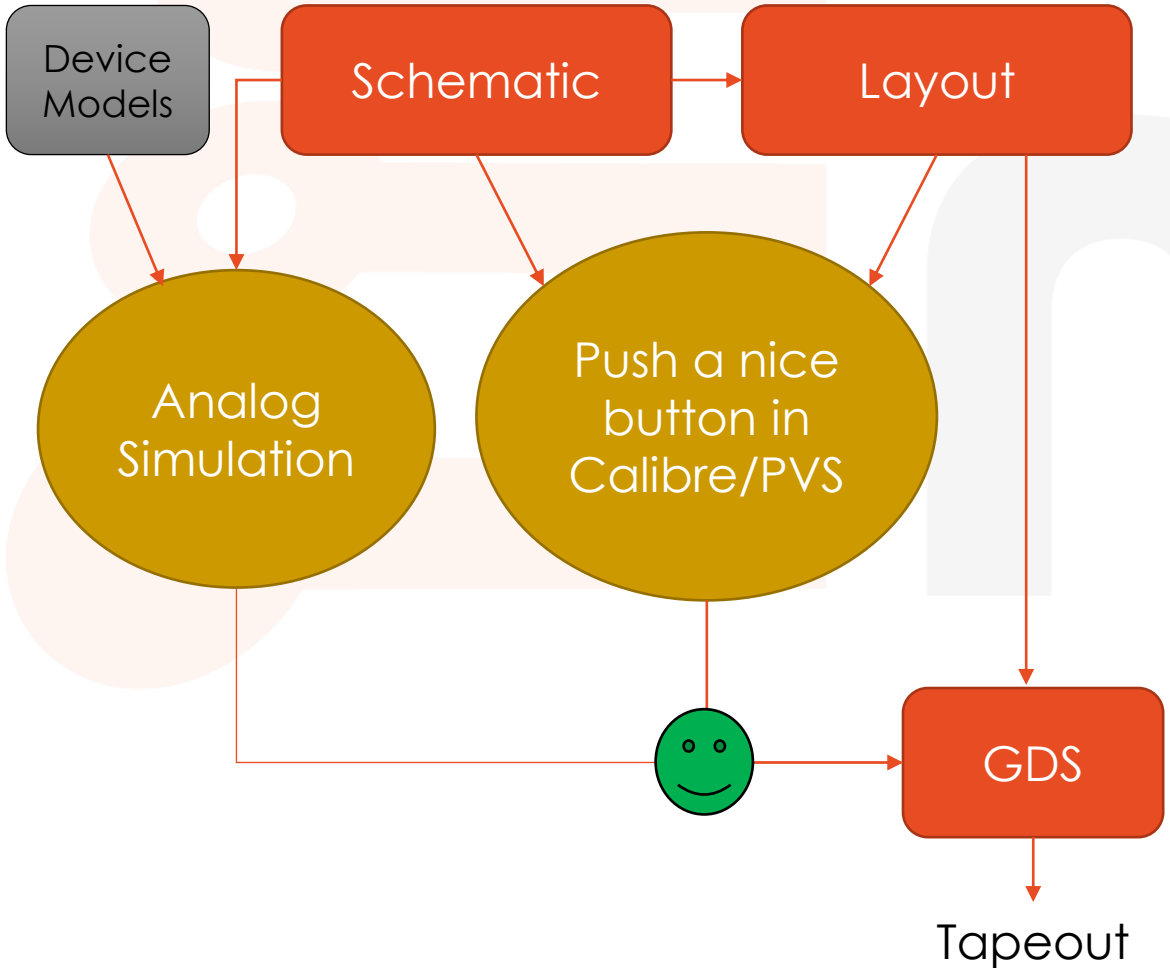
25 December 2021

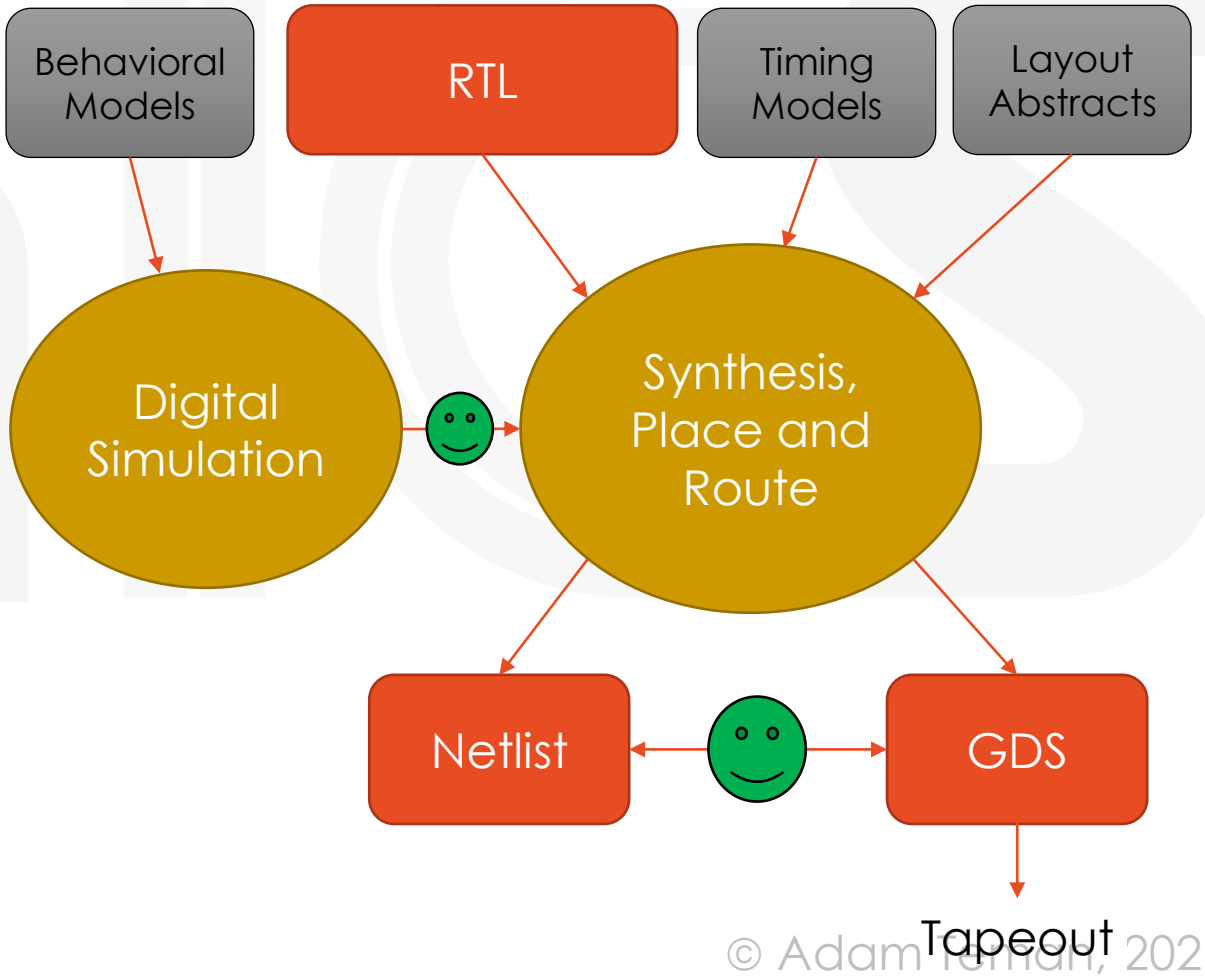
Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Introduction

Analog-on-Top vs. Digital-on-Top



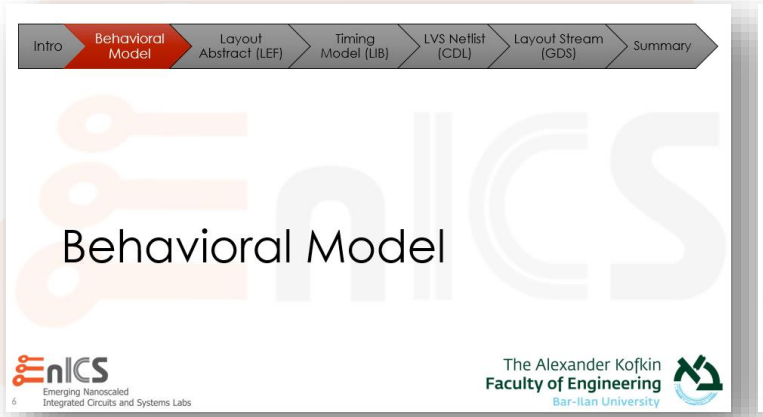


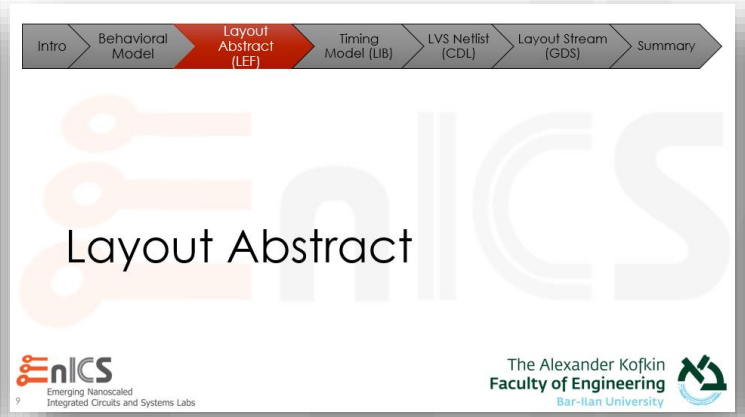
What do we need to deliver

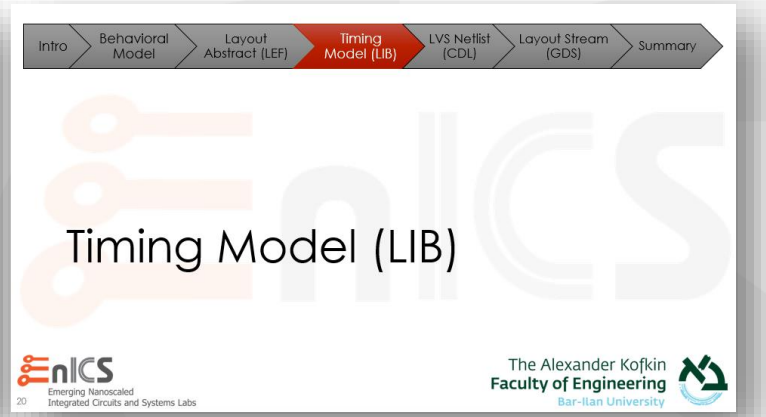
- **Early on in the project:**
  - Behavioral (RTL) model for RTL simulation
- **At an intermediate stage**
  - Layout abstract (`.lef`) for floorplanning and power planning
  - Timing model (`.lib`) for synthesis
- **For signoff**
  - Spice netlist (`.cdl`) for LVS
  - Full layout (`.gds`) for LVS and DRC

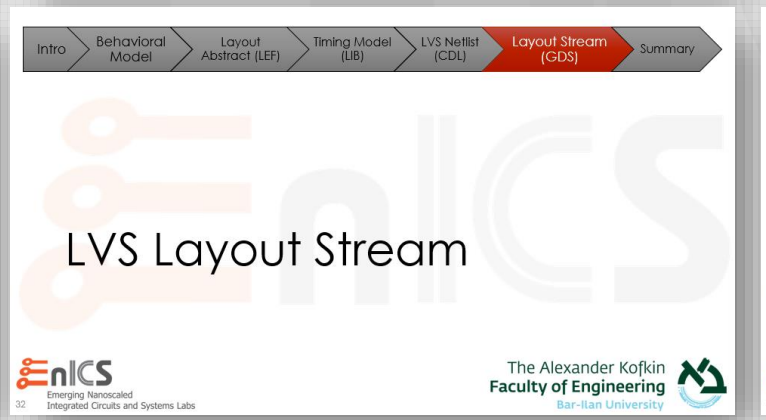
- **In the following slides, we will overview how to prepare your custom block and create these files.**

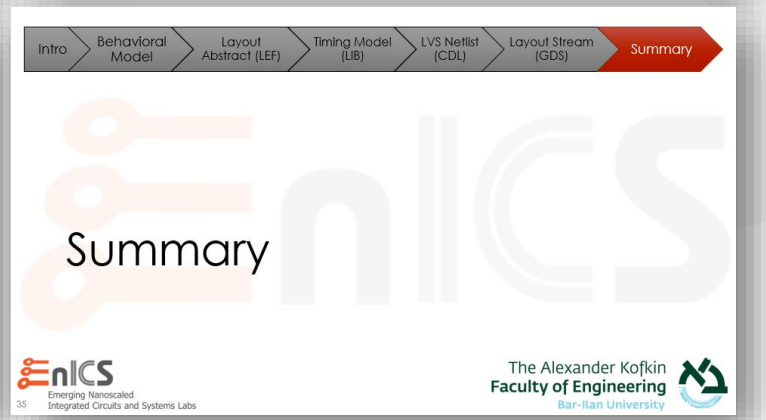
Lecture Overview











Behavioral Model

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

Behavioral Description

- **Assuming the custom block's top hierarchy is called "`topcell`", your behavioral description is a Verilog file with the following properties:**
  - **File name**: `topcell.v`
  - **Module name**: `topcell`
  - **Interface**: Input and output ports with the same names as the circuit port names.
    No power/ground inout ports.
  - **Body of code**: A description of how you want your block to behave in digital simulation. If possible, it should behave equivalent to the actual behavior of the custom block.
    This does not have to be synthesizeable.

Example: Memory Macro


```
`timescale 1ns/1ps

module edram_wPS(
   // Outputs
   DOxDO,
   // Inputs
   ClkxSI, WAddrxDI, RAddrxDI,
          DIxDI, WExSI, RExSI
   );

parameter WIDTH = 64;
parameter DEPTH = 4096;
parameter RT_IN_NS = 1000000;
localparam DEPTH_W = $clog2(DEPTH);

input ClkxSI;
input [DEPTH_W-1:0] WAddrxDI; // write address
input [DEPTH_W-1:0] RAddrxDI; // read address
input [WIDTH-1:0] DIxDI; // input data
input WExSI; // write enable
input RExSI; // read enable
output reg [WIDTH-1:0] DOxDO; // output data
```

```
reg [WIDTH-1:0] MEM [0:DEPTH-1];
integer timestamp [0:DEPTH-1];
initial timestamp = '{DEPTH{0}};

always @(posedge ClkxSI) begin
    if (WExSI) begin
        MEM[WAddrxDI] DIxDI;
        timestamp[WAddrxDI] <= $time;
    end

    if (RExSI) begin
        if (timestamp[RAddrxDI] + RT_IN_NS > $time)
            DOxDO <= MEM[RAddrxDI];
        else begin
            DOxDO <= {WIDTH/16{16'haa55}};
        end
    end
end

endmodule
```

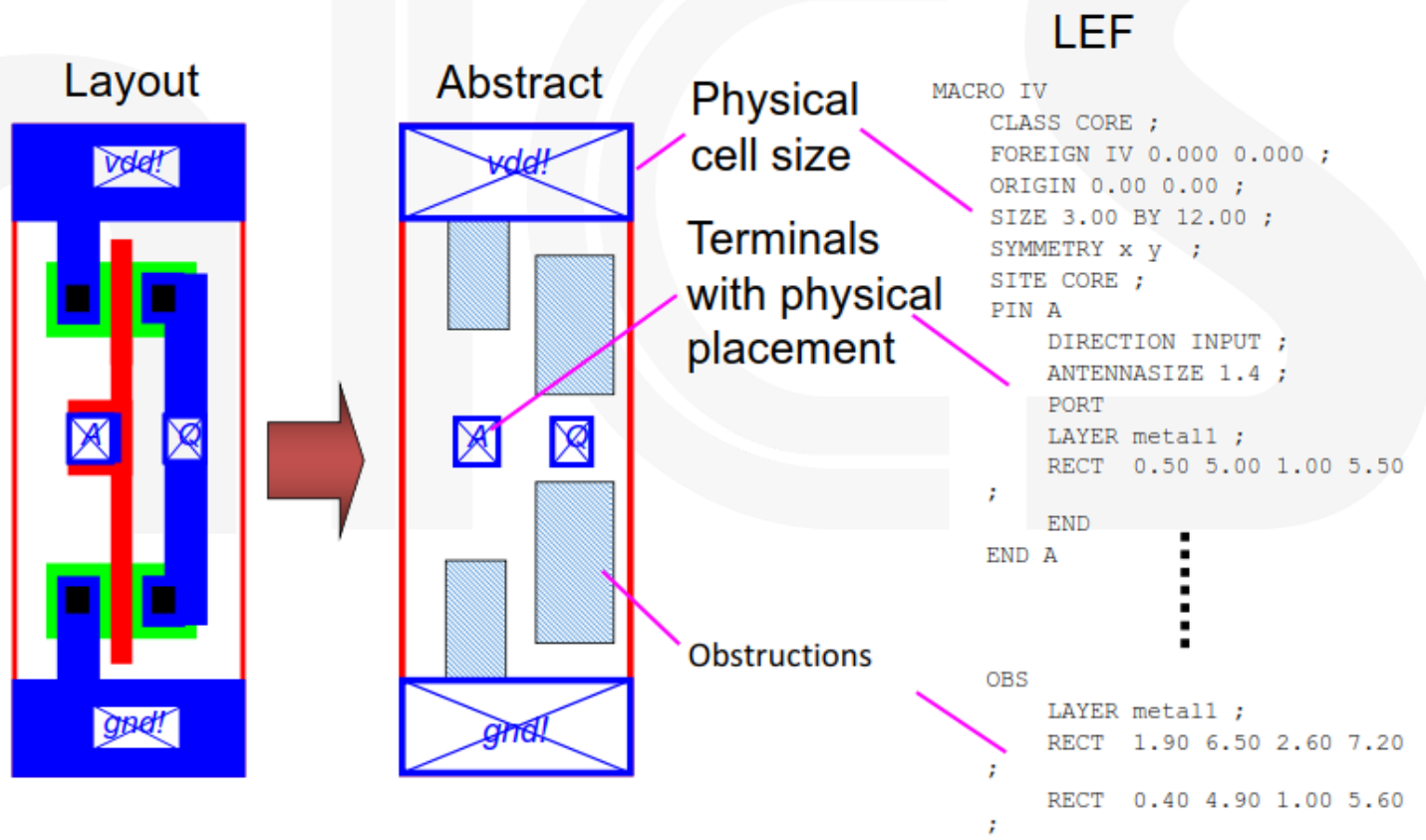
Layout Abstract

Emerging Nanoscaled Integrated Circuits and Systems Labs

The Alexander Kofkin Faculty of Engineering
Bar-Ilan University

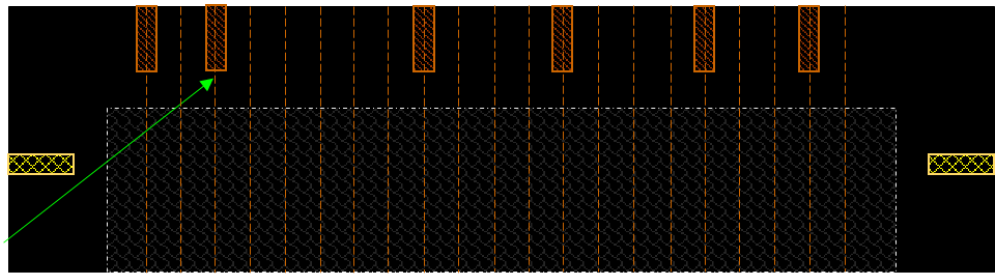
Layout Abstract (.lef)

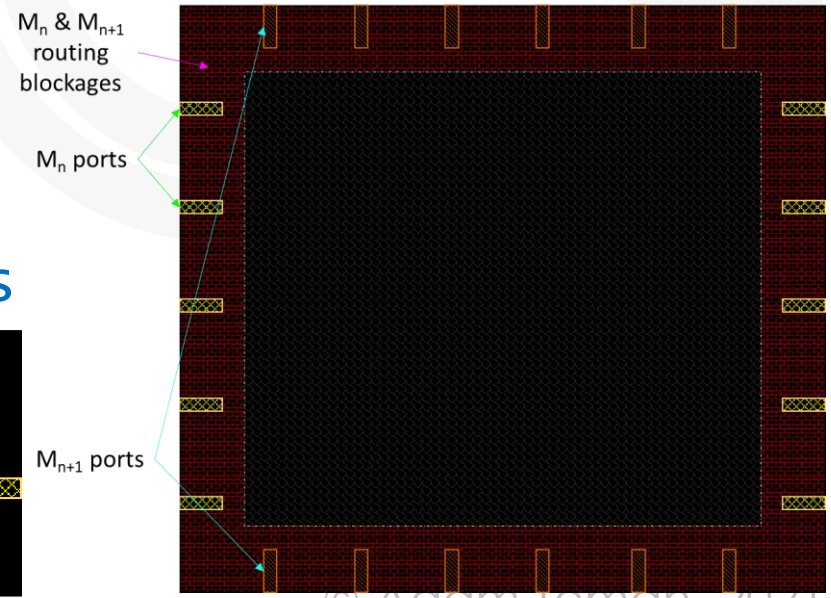
- **Well before you have finished your detailed layout, you will need to provide the backend team with a physical abstract of your block, including:**
  - Size (Width x Height)
  - Pin locations and metal layers
  - Power connections
  - Blockages
- **This is provided in as a `.lef` (Library Exchange Format) file, which can be:**
  - Written by hand
  - Created with the "Abstract Generator"



Preparing your abstract: Pins (Ports)

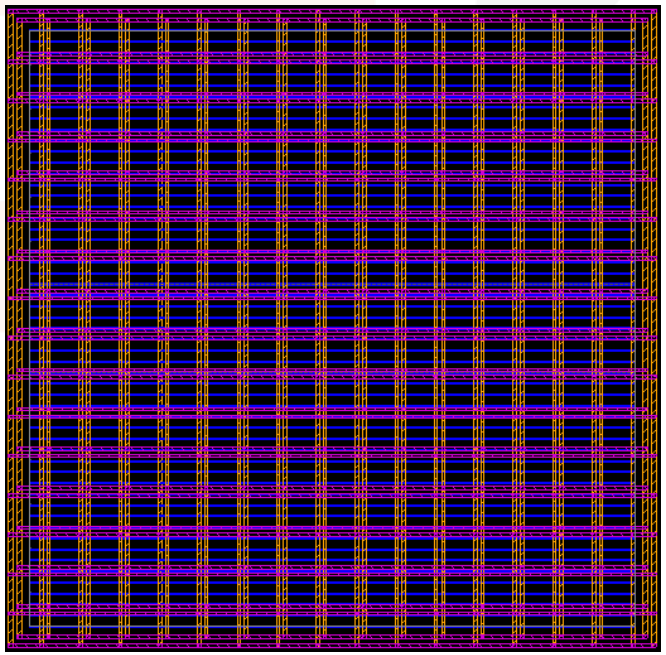
- **There are many guidelines that can improve your *routability*, but in general:**
  - Put pins on layers according to process preferred routing direction.
  - Try to align your pins to a periodic grid, preferably the routing tracks.
  - Try to separate ports by at least one empty track.
  - Pins should reach the edge of the macro and be completely covered by routing blockage ("`cover`" blockage with no "`pin cutouts`")
  - Power (PG) Pins should be fully detailed and wide enough for a via to be dropped. This will enable multiple connections to power.
  - On a top layer with only power, provide `pin cutouts` or entirely remove blockage



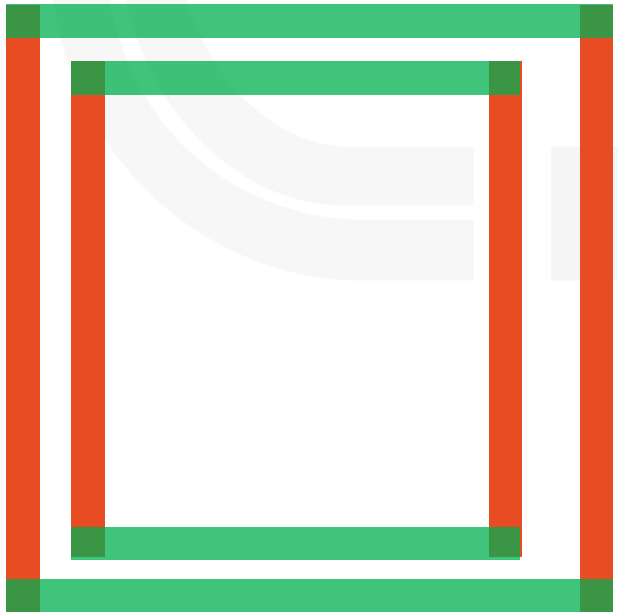


Preparing your abstract: Power Grid

- **Need a robust power delivery network**
  - For preventing IR Drop and EM
  - Rule of thumb: 10% of routing resources for power
- **Two main structures:**
  - Mesh



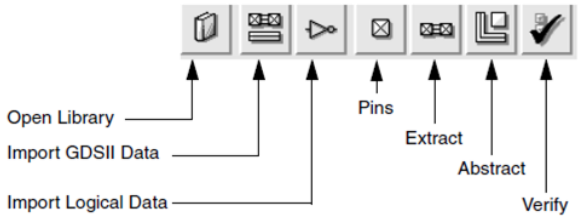
  - Ring



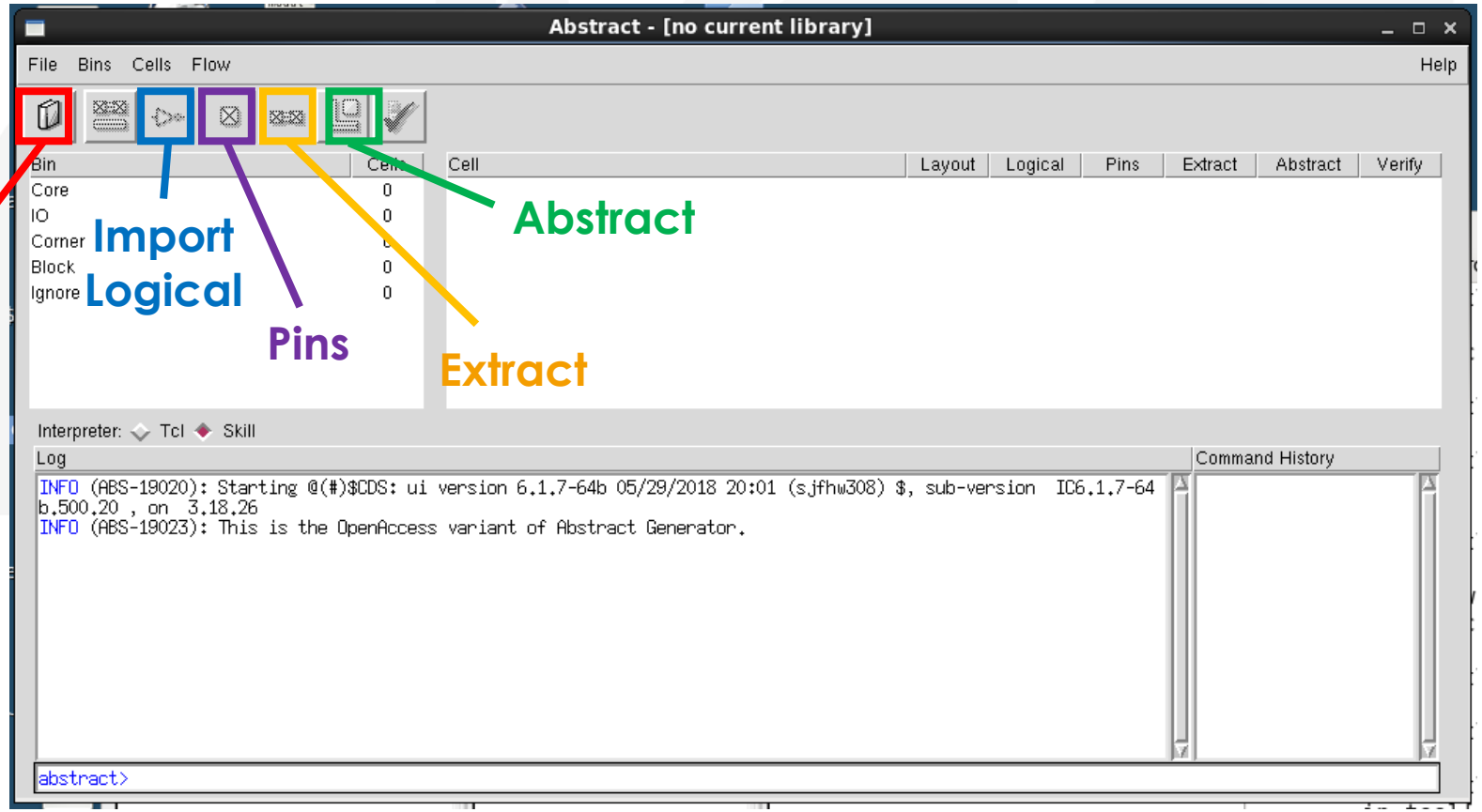
Preparing your abstract: Others

- **Make sure you provide Antenna information in your LEF**
  - This will enable the router to solve antenna problems
  - Otherwise, during signoff DRC, you will have “surprises”
- **Prepare yourself for Metal Fill**
  - All layers have to have density between 30%-70% or so.
  - This will be added during signoff using an automated flow.
  - If you have sensitive areas, add metal fill blockage to prevent this.
  - If you are sensitive to timing, add the metal fill at the macro level.
- **High Voltage Markers**
  - If you are using non-standard voltages, add Marker CAD layers to employ the correct DRC rules on these nets.
- **Make sure your Origin is at `(0,0)` and your `PR Boundary` is correct!**

Using the Abstract Generator

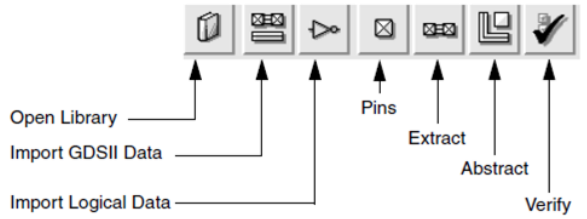


- The **Abstract Generator** is a Cadence tool, started by running `abstract`
- The **Abstract Generator** has a five-step flow
  1. Import Library (Layout)
  2. Import Logical (Interface)
  3. Pin Derivation
  4. Shape Extraction
  5. Abstract Generation

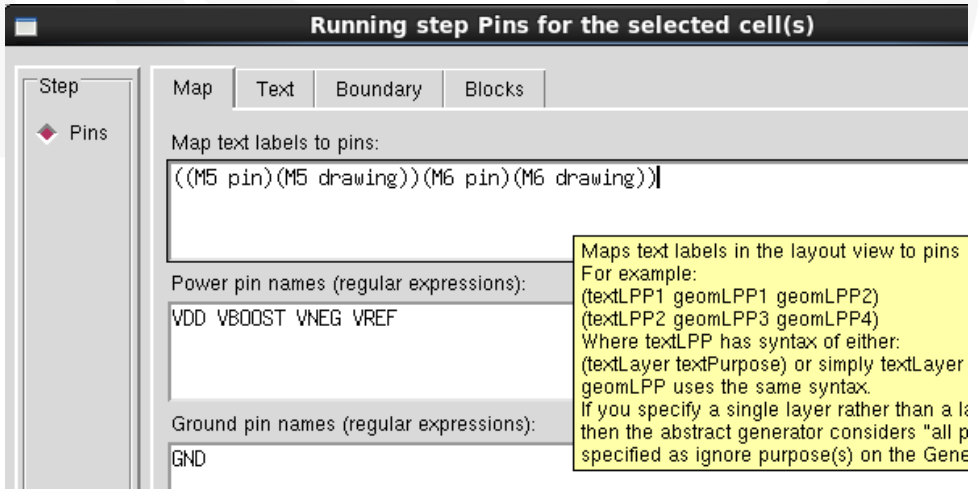


- When this is done you can export the `.lef` file.

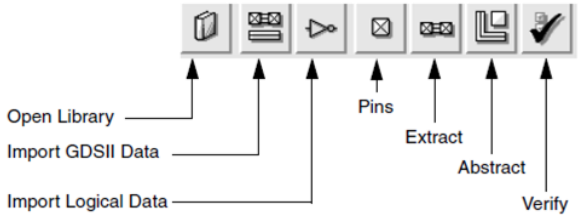
Using the Abstract Generator (2)



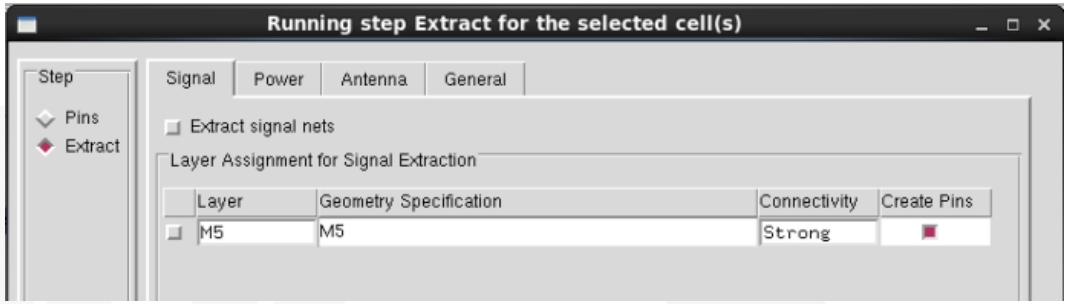
- **Import Library Stage:**
  - Choose a library from Virtuoso (`cds.lib`)
- **Import Logical Stage:**
  - Upload a Verilog (`.v`) or Liberty (`.lib`) file to provide the pin interface.
- **Pins:**
  - *Correct way*: define pins (shapes defined with "`Create Pin`") in your layout.
  - You can also have the tool extract pin shapes based on Labels.
  - In the **MAP** tab define pin layers and make sure all power/gnd label names are specified.
- **After running, check the `abstract.pin` cellview in your library**

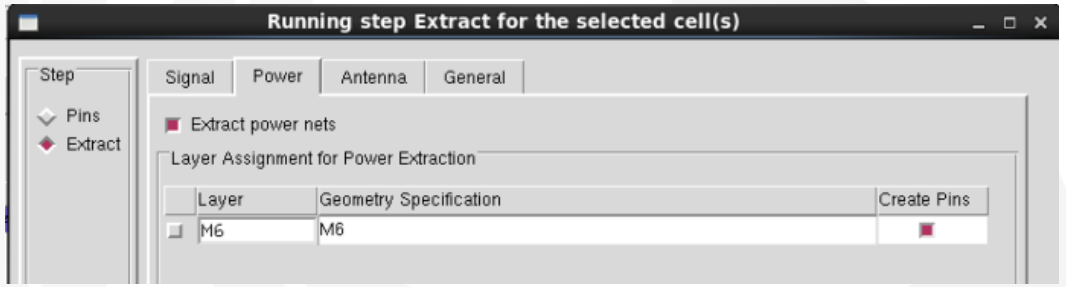


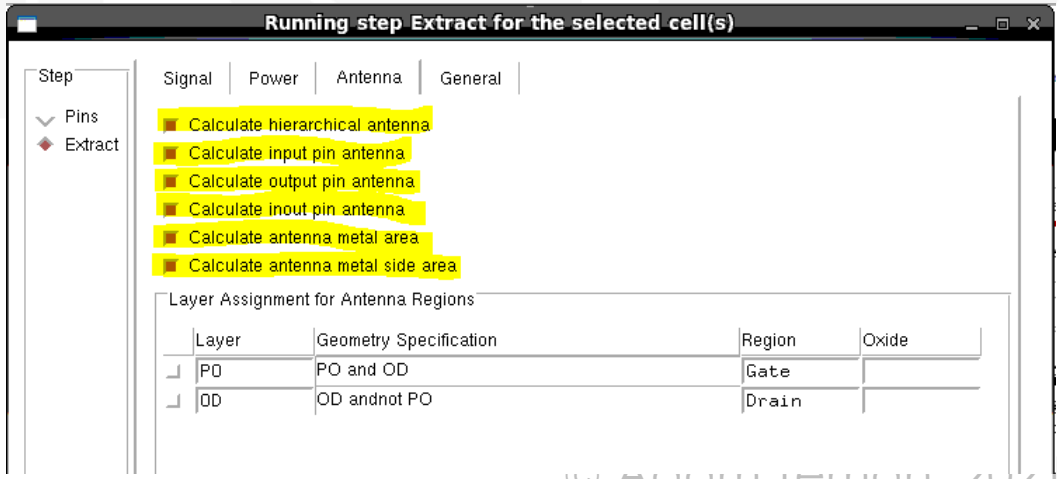
Using the Abstract Generator (3)



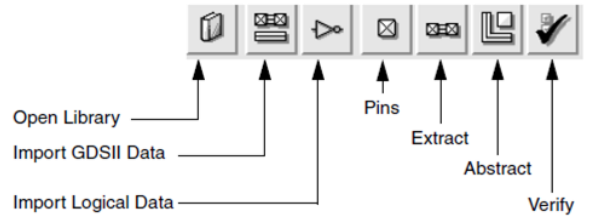
- **Extract Stage**
  - **SIGNAL** tab: Only leave the layers that you want to be written as signals to the LEF.
  - **POWER** tab: Only leave the layers that you want to be written as power pins to the LEF.
  - Select "**EXTRACT POWER NETS**" so the whole power wire will be extracted
  - **ANTENNA** tab: Enable all options. This will provide antenna data in the LEF, otherwise, you will probably have antenna violations in full chip DRC.
- **After running, check the `abstract.ext` cellview in your library**



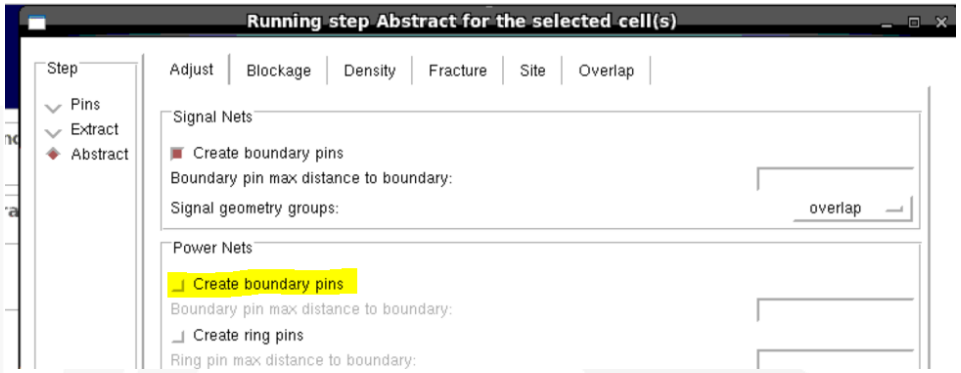


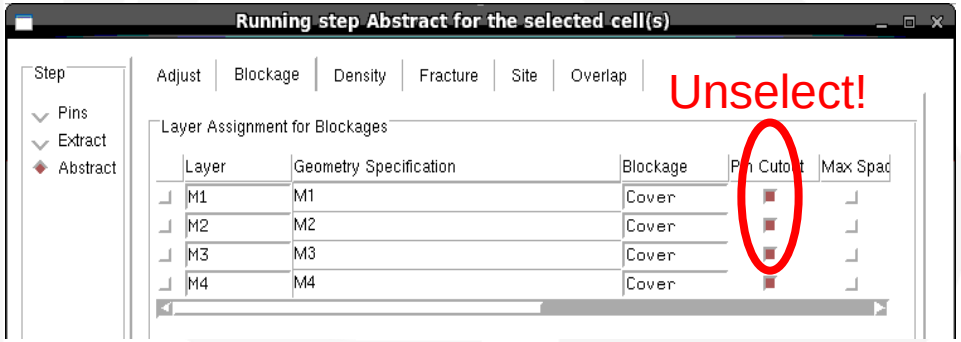


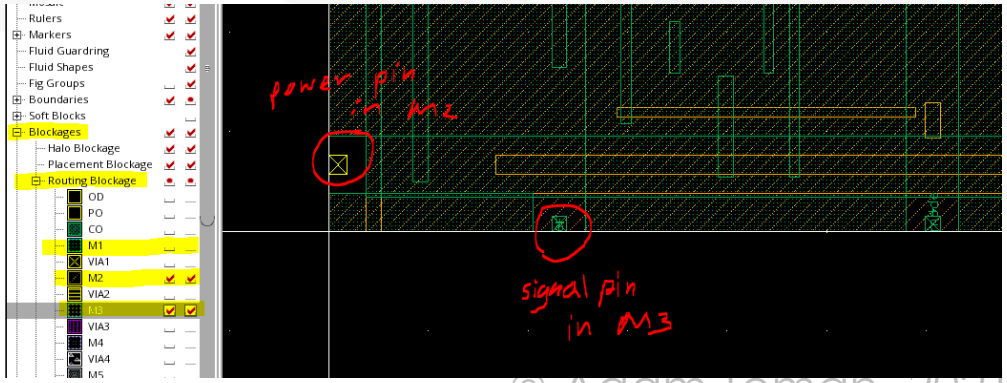
Using the Abstract Generator (4)



- **Abstract stage:**
  - **ADJUST** tab: Select "**CREATE BOUNDARY PINS**" for signals only! (*uncheck* for power!)
  - **BLOCKAGE** tab: Define "**COVER**" blockage on all layers to block inside the macro.
  - *DO NOT* select "**PIN CUTOUT**" for pin layers. This can lead to DRC errors during routing!
  - For Power Metals, select "**PIN CUTOUT**"!
- **After running, check the `abstract` cellview**
  - Check that pins touch the edges
  - Check that blockages are created correctly (**OBJECTS → BLOCKAGES → ROUTING BLOCKAGES**).

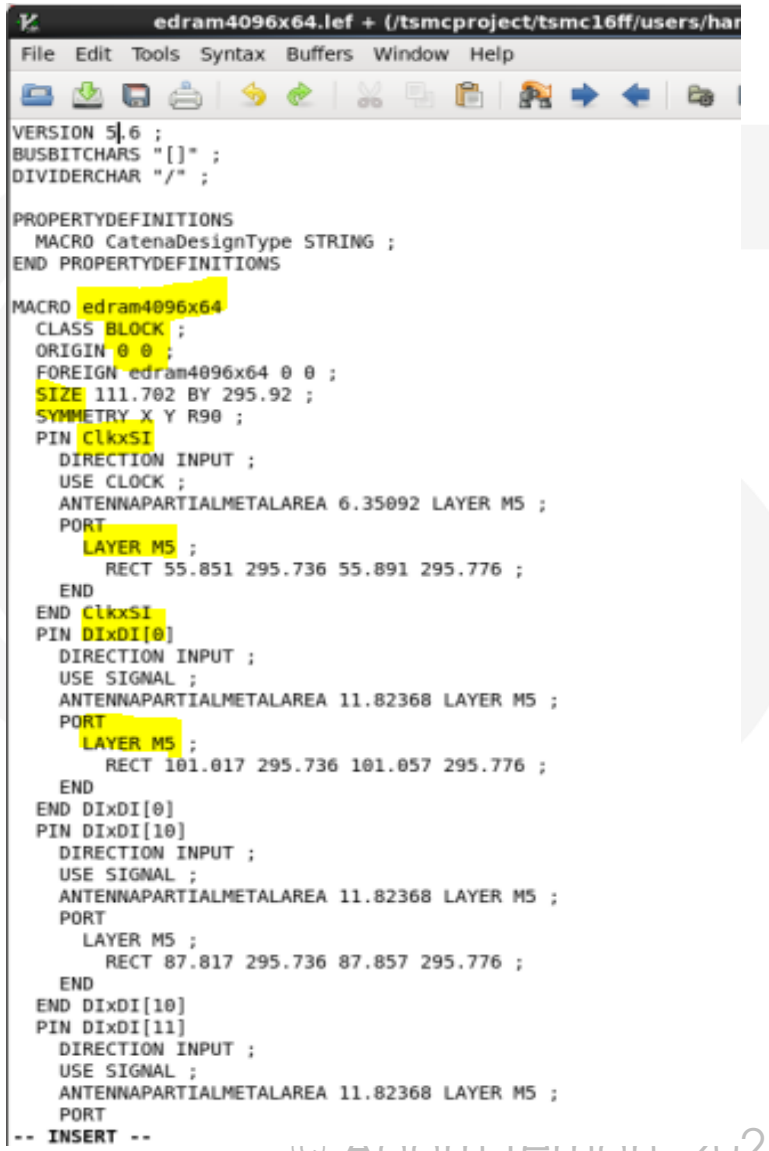






Using the Abstract Generator (5)

- **To export the `.lef` file, select FILE→EXPORT→LEF**
- **Open the file to reassure it was generated as expected:**
  - You can see the `MACRO` name and the `BLOCK` class
  - The `SIZE` is calculated
  - The `ORIGIN` should be at `(0,0)`
  - All the `PIN`s are created and have coordinates in the correct layers.
  - The antenna information is described.
  - "Cover" blockage should be defined, i.e., very few rectangles of type `OBS`.



Shortcut for Initial LEF

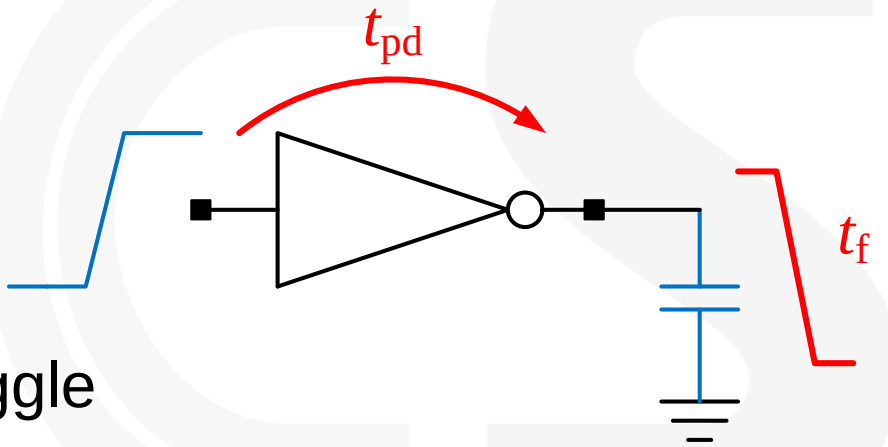
- **You will need to provide an initial LEF early on in the project for floorplanning.**
  - After defining footprint, power connections and pin placement, you will need to freeze this and only change the internals.
- **A quick way to do this is to use Innovus:**
  - Create "empty" Verilog netlist (module) only including interface.
  - Load this netlist into Innovus according to technology backend flow.
  - Define floorplan size with `create_floorplan` command
  - Define power rails (rings/stripes) and pins (`edit_pin`)
  - Export LEF with the `write_lef_abstract` command

```
write_lef_abstract my_macro.lef -stripe_pins -pg_pin_layers {M5} \
      -port_for_each_stripe_pin -extract_block_pg_pin_layers {M5} -top_layer M6
```

Timing Model (LIB)

Timing Model (.lib)

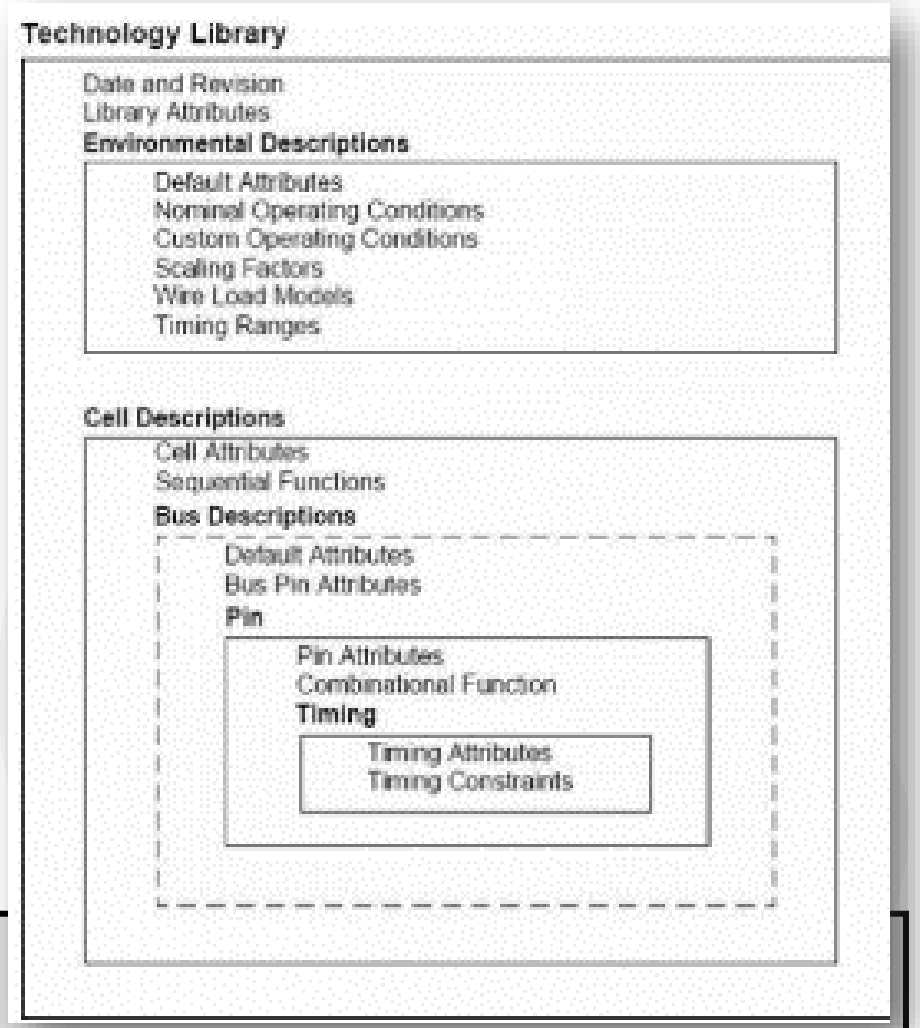
- **The digital-on-top flow is based on static timing analysis (STA) verification**
  - Throughout the flow, max-delay, min-delay and DRV constraints are checked.
- **For each instance in the design, STA requires:**
  - Propagation delays through timing arcs
  - Capacitances (loads) on pins (mainly inputs)
  - Drive capabilities (affecting transition) of outputs
  - Type of transition (rising/falling) due to an input toggle
  - Dynamic and Static Power consumption



- **This data is provided by a Liberty (`.lib`) file, based on:**
  - Input net transition (t_{rise}, t_{fall})
  - Output Load Capacitance (C_{load})
- **A separate `.lib` file should be provided for each corner (PVT+RCX)**

Liberty Format

- **A `.lib` file has the following hierarchy:**
  - `Library`: a collection of cells at a certain corner.
  - `Cell`: a specific cell (e.g., macro)
  - `Pin`: Timing/Power information for each pin
- **Each `pin` has:**
  - General info, e.g., capacitance, functionality
  - `Timing`: Propagation delay and output transition
  - `Power`: Power consumption of arc
  - `Constraint`: Setup/Hold constraints




```
lu_table_template(delay_template_5x5) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
  index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
}
```

```
cell (INVX1) {
  pin(Y) {
    timing() {
     cell_rise(delay_template_5x5) {
       values ( \
         "0.1479, 0.2180, 0.3598, 0.922, 1.766", \
         "0.2243, 0.2929, 0.4303, 0.991, 1.831", \
         "0.3653, 0.4487, 0.5842, 1.135, 1.970", \
         "0.4620, 0.5515, 0.7010, 1.244, 2.081", \
         "0.7564, 0.8742, 1.0570, 1.628, 2.449"); }
```

Library Characterization

- **The best way to create a `.lib` file is with a library characterization tool, such as Cadence Liberate**
  - For each timing arc, a SPICE testbench is generated and run.
  - Propagation delay, output transition, power consumption, setup/hold constraints, etc. are extracted and written to a `.lib` file
- **Cadence Liberate comes in several "flavors"**
  - Liberate: Used for Standard Cell characterization
  - Liberate-MX: Used for Memory Macro characterization
  - Liberate-AMS: Used for Analog Mixed-Signal (custom) blocks

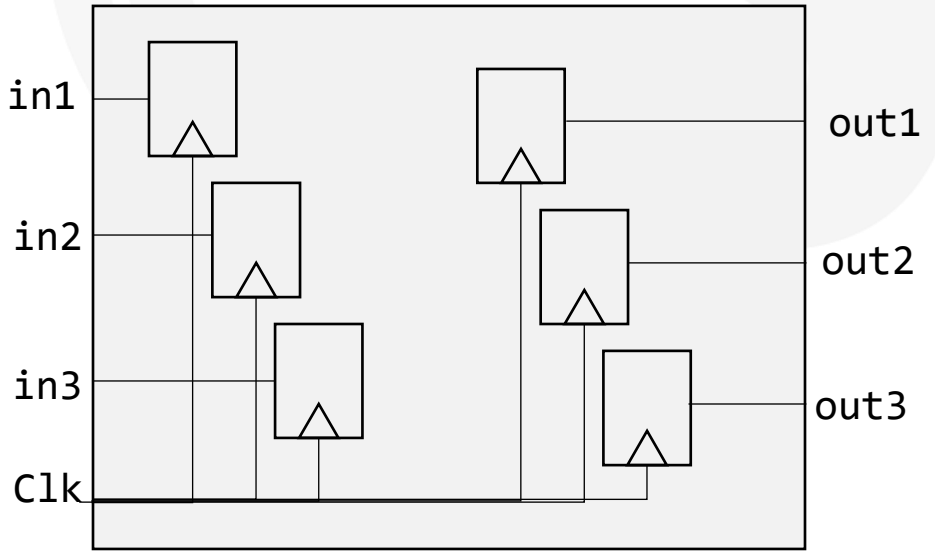
- **However, setting up Liberate-AMS is often beyond our scope, and therefore, we will generate our `.lib`s manually or semi-automatically.**

Manually Creating a .lib File

- `Library`:
  - Copy general information (only required parts) from standard cell library
- `Cell`:
  - Cell name should be equivalent to cell name in Virtuoso
  - Manually fill in area and copy other fields from standard cell library
  - Provide power information in `pg_pin` fields
- **Input `Pins`**
  - **For each input pin**, provide (measured) input capacitance
  - If input pin has setup/hold constraints, measure and provide table
- **Output `Pins`**
  - Provide `timing` field for each **timing arc** of each output pin
  - For each of these generate tables (measured) for delay, transition, and power

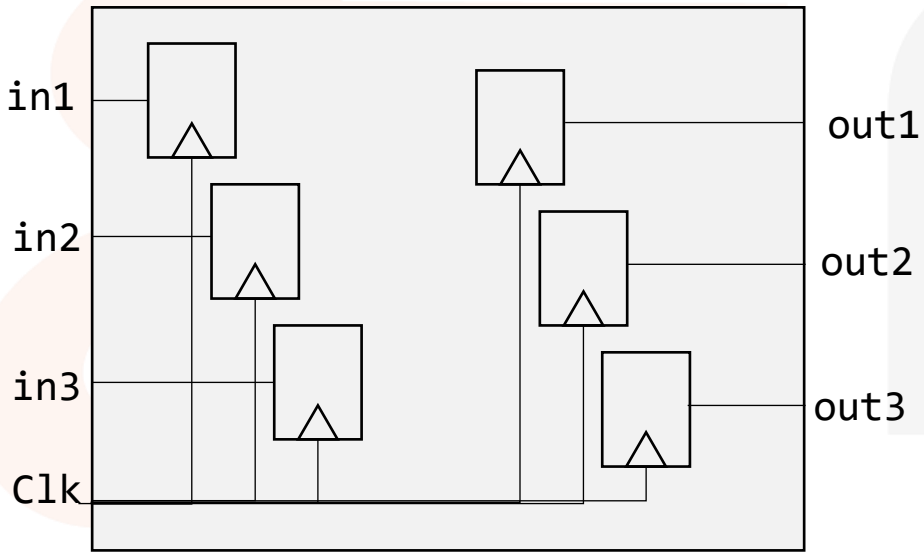
Wrapping with Registers

- **To get around the tough measurement, surround your block with registers**
  - Inputs to the design are just `D` pins of flip flops.
  - Setup/Hold constraints are between the `clock` and `D` of these flip flops
  - Output arcs are just `clock-to-Q` arcs of flip flops
- **So, you can copy the `cell` definition of the flip flop from the standard cell library for each pin**
- **Or better yet, you can semi-automate this by:**
  - Writing an interface-only block in Verilog with input/output sampling.
  - Loading into Innovus and writing out a .lib.



Semi-Automatic .lib Generation

- **Interface-only Verilog Netlist**




```
module my_block(
    out1,out2,out3,
    in1, in2, in3,
    clk, rst_n, VDD, VSS );
output out1, out2, out3;
input in1, in2, in3;
input clk, rst_n;
inout VDD, VSS;
wire in1_wire, in2_wire, in3_wire;

DFF in_ff1 (.CK(clk), .D(in1), .Q(in1_wire), .RN(rst_n));
DFF in_ff2 (.CK(clk), .D(in2), .Q(in2_wire) , .RN(rst_n));
DFF in_ff3 (.CK(clk), .D(in3), .Q(in3_wire) , .RN(rst_n));

DFF out_ff1 (.CK(clk), .D(in1_wire), .Q(out1), .RN(rst_n));
DFF out_ff2 (.CK(clk), .D(in2_wire), .Q(out2), .RN(rst_n));
DFF out_ff3 (.CK(clk), .D(in3_wire), .Q(out3), .RN(rst_n));

endmodule
```

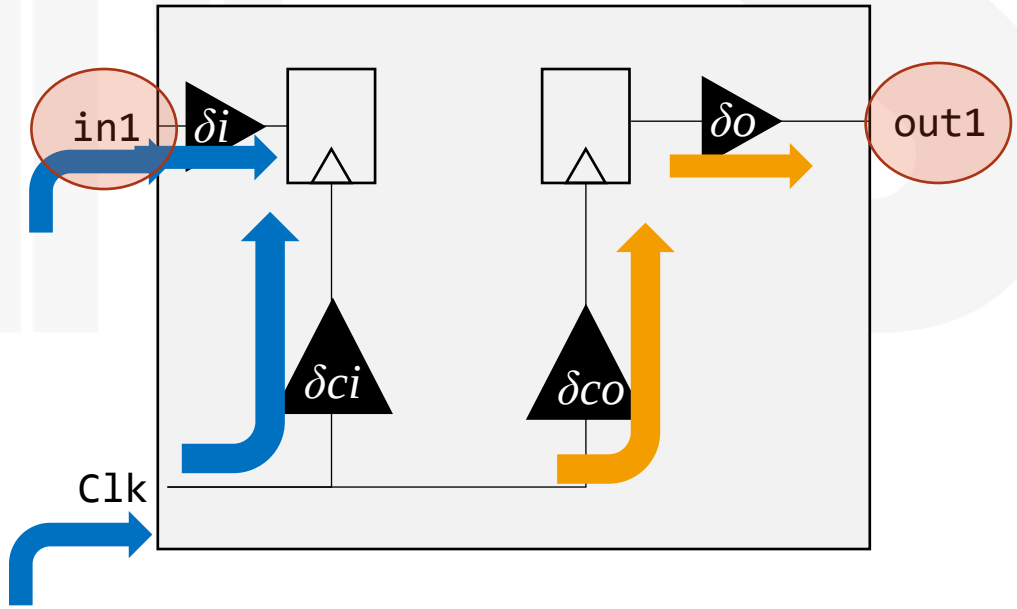
Semi-Automatic .lib Generation

- **Load into Innovus and write out .lib file**

```
# Invoke by: innovus -stylus -no_gui -file gen_initial_lib.tcl

# Read in the MMMC file for this technology
read_mmmc generic_innovus_mmmc.tcl

# Read in interface-only netlist
read_netlist my_block_interface.v

init_design

# Write out .lib in one of the corners
write_timing_model -view typical_corner my_block_interface.lib
```

Final .lib adjustment

- **Even when surrounding your block with registers, you will have different timing than the "vanilla" standard cell, due to your internal wires:**
  - Measure input capacitance and update **input pins**.
  - Measure skew between inputs/outputs and clock:
    $in\_skew = \delta ci - \delta i \quad out\_delay = \delta co - \delta o$
  - Update **input pin** setup/hold constraints:
    $setup \mathrel{-}= in\_skew$ (setup got easier)
    $hold \mathrel{+}= in\_skew$ (hold got harder)
  - Update **output pin** propagation delay:
    $tcq \mathrel{+}= out\_delay$ (delay got longer)



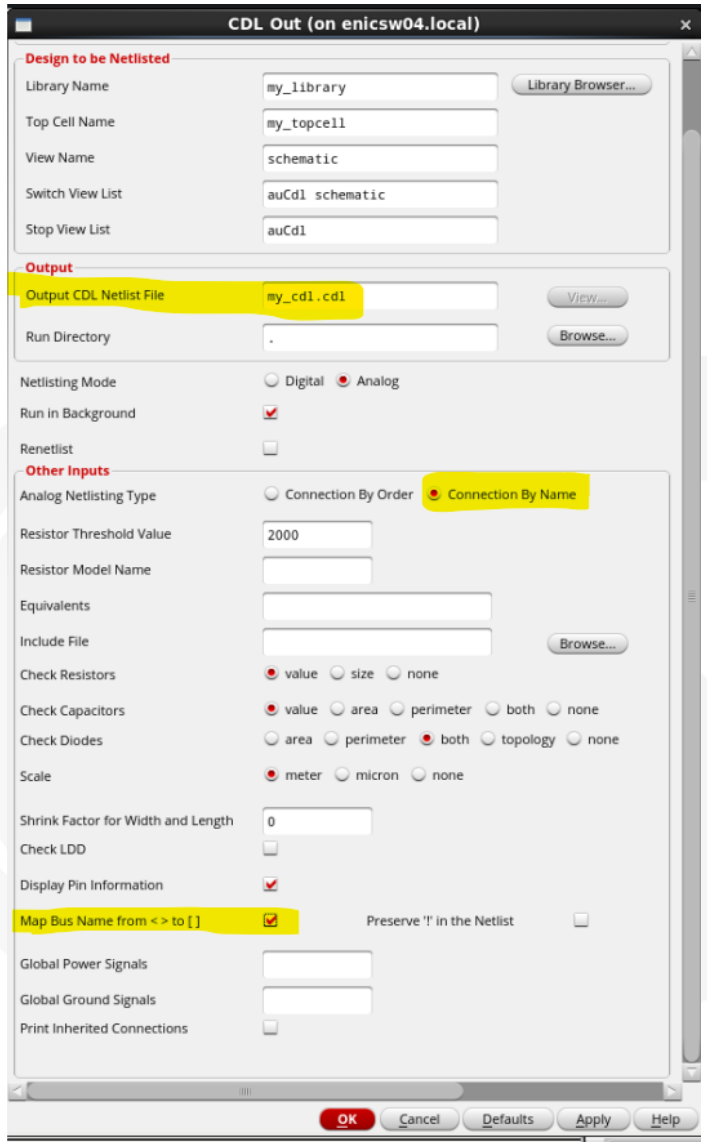
LVS Netlist

LVS Netlist (CDL)

- **LVS is "Layout vs. Schematics". Therefore, we need to provide:**
  - Layout – Usually in the GDSII format
  - Schematic – Usually in the CDL (Circuit Description Language) format
- **CDL is a subset of SPICE**
  - There are some subtle differences, but it is *basically equivalent* to SPICE.
  - You shouldn't include parasitics in your CDL file!
- **The CDL is the result of *netlisting* your schematic**
  - Netlisters are an important part of the EDA flow.
  - Virtuoso runs a netlister before running a simulation.
  - Calibre/PVS run a netlister to create an LVS run.
  - Virtuoso also has a *standalone netlister* for creating CDL files.
  - Note that these three options, don't always provide the same result!

Exporting your CDL

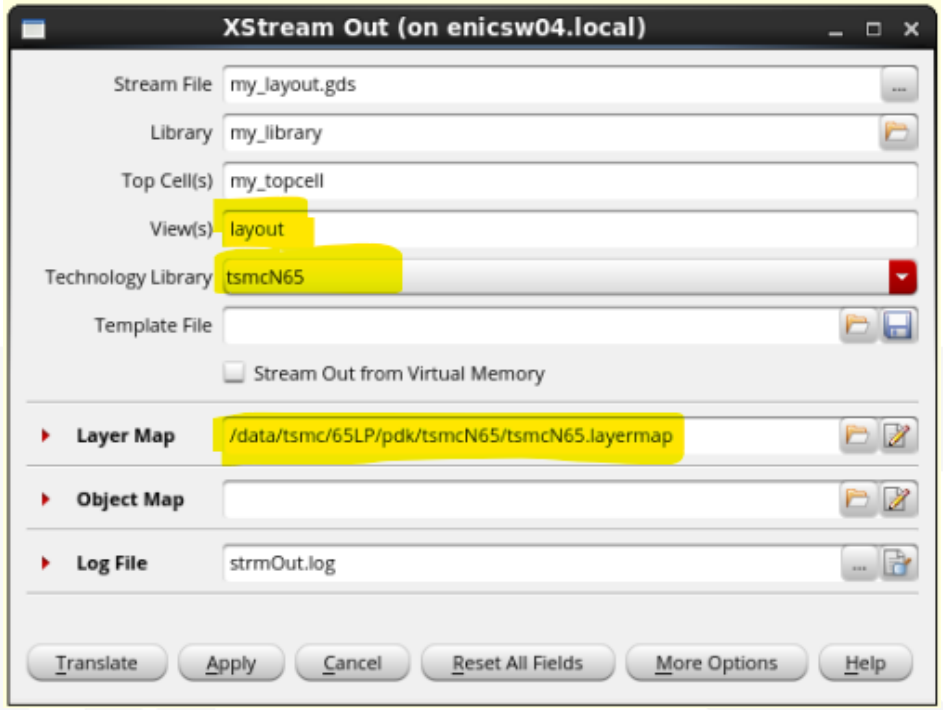
- **Ensure that your circuit:**
  - Has no GLOBAL nets (i.e., with !).
  - Uniquify your cell names
    This is *essential* for instantiated standard cells!
- **From the CIW, select FILE→EXPORT→CDL .**
  - Choose your schematic to export.
  - The exported file is defined in the OUTPUT section
  - Select MAP BUS NAMES FROM <> TO [ ] to provide Verilog compatible bus names!
  - Select CONNECTION BY NAME to use the $PINS style of connectivity, as opposed to connection by reference/position.
- ***Another option* is to take the Calibre/PVS netlist generated during LVS.**



LVS Layout Stream

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

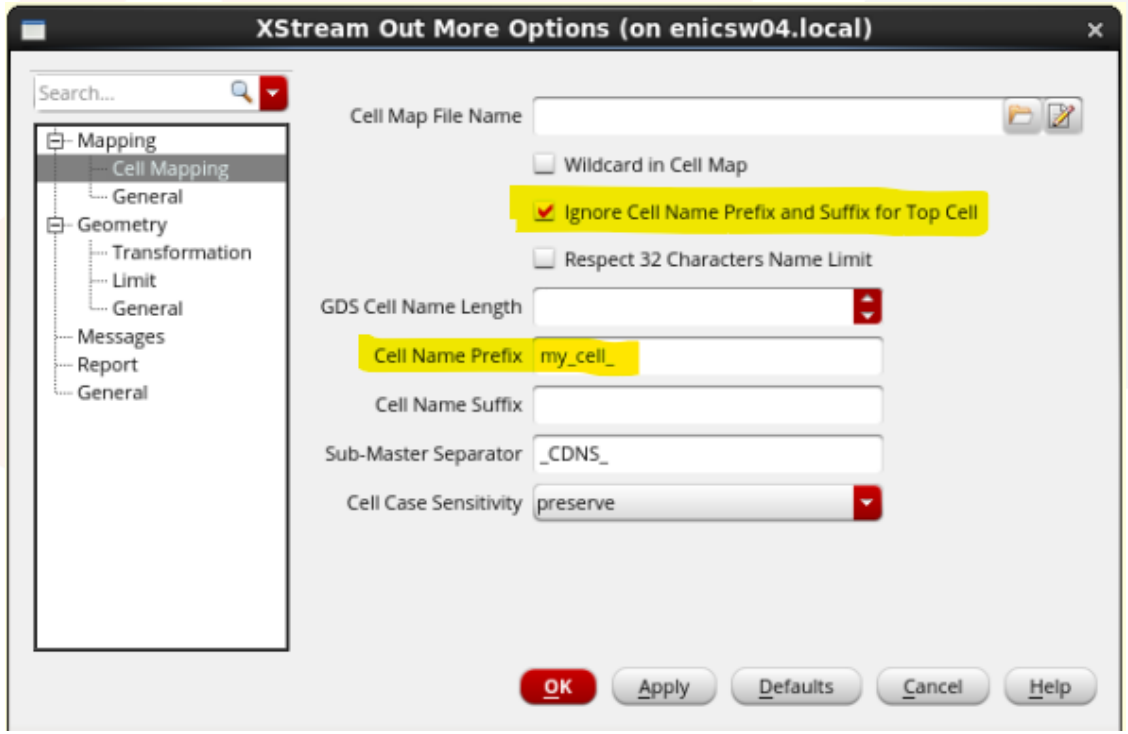
LVS Layout (Stream)

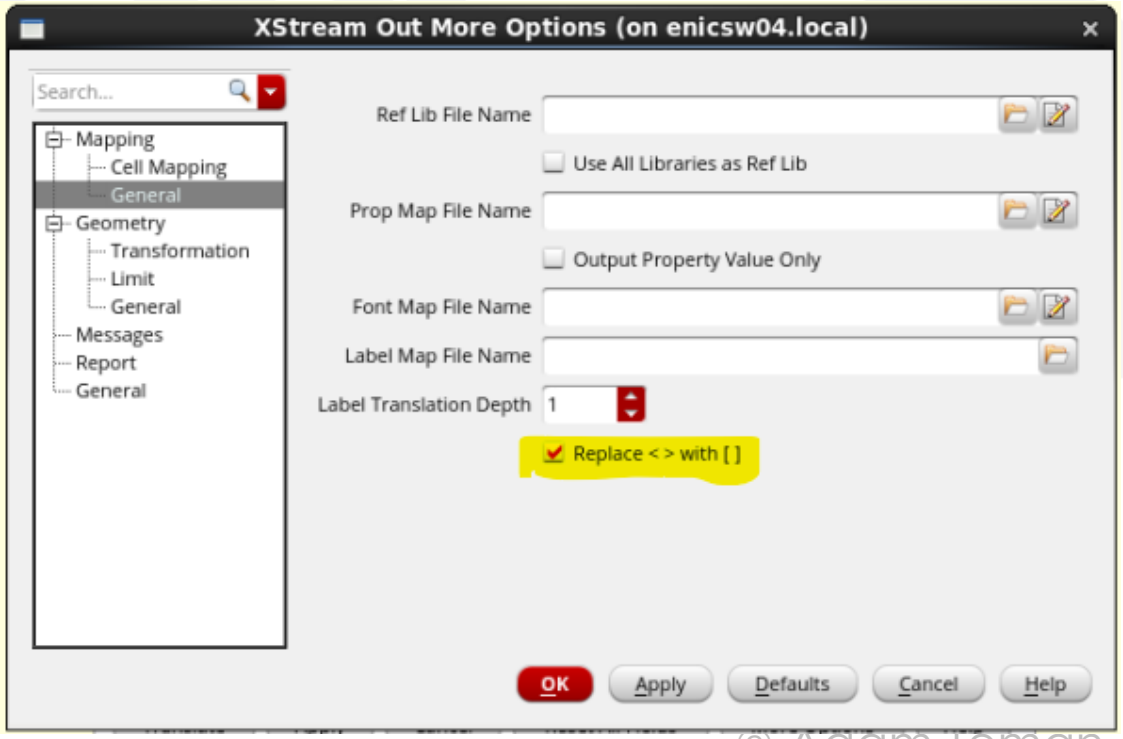


- **There are two options to extract a GDSII file from your layout:**
  - Standalone XStream Out (`strmout`) tool
  - Take it from Calibre/PVS extraction
- **From the CIW, select FILE→EXPORT→STREAM .**
  - Select the layout view (**LIBRARY/TOP CELL/VIEW**) of the cell you want to export
  - The exported GDS is defined in the **STREAM FILE** field
  - The **TECHNOLOGY LIBRARY** should be the techlib of the PDK.
  - The **LAYER MAP** is a file that translates between the layout layer and the GDS layer number. It should be provided in your PDK.
- **Select TRANSLATE to export your GDS**
  - But not before you follow the steps on the next slide!

Stream Out – Important Points

- **In the XSTREAM OUT form, choose the MORE OPTIONS button and:**
  - Under CELL MAPPING add CELL NAME PREFIX (or SUFFIX), such as "`my_cell_`" and select IGNORE CELL NAME PREFIX AND SUFFIX FOR TOP CELL.
  - Under MAPPING→GENERAL select REPLACE <> WITH [ ].





Summary

Summary

- **Following all these steps, you should provide the following early on:**
  - `my_block.v` file: behavioral model for digital simulation
  - `my_block_initial.lef`: early stage LEF file with frozen size, pins and power.
  - `my_block_initial.lib`: early stage LIB file with registers around interface

- **Your final delivery for signoff should include:**
  - `my_block.lef`: final `.lef` that should be equivalent to initial `.lef`.
  - `my_block.lib`: final `.lib` that should be equivalent to initial `.lib`, if you used the register wrapping approach.
  - `my_block.cdl`: uniquified SPICE (`.cdl`) netlist without parasitics or globals.
  - `my_block.gds`: uniquified, DRC/LVS clean `GDSII` file with metal fill blocks, where required, or after local metal fill.