RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.1: SoC Overview

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

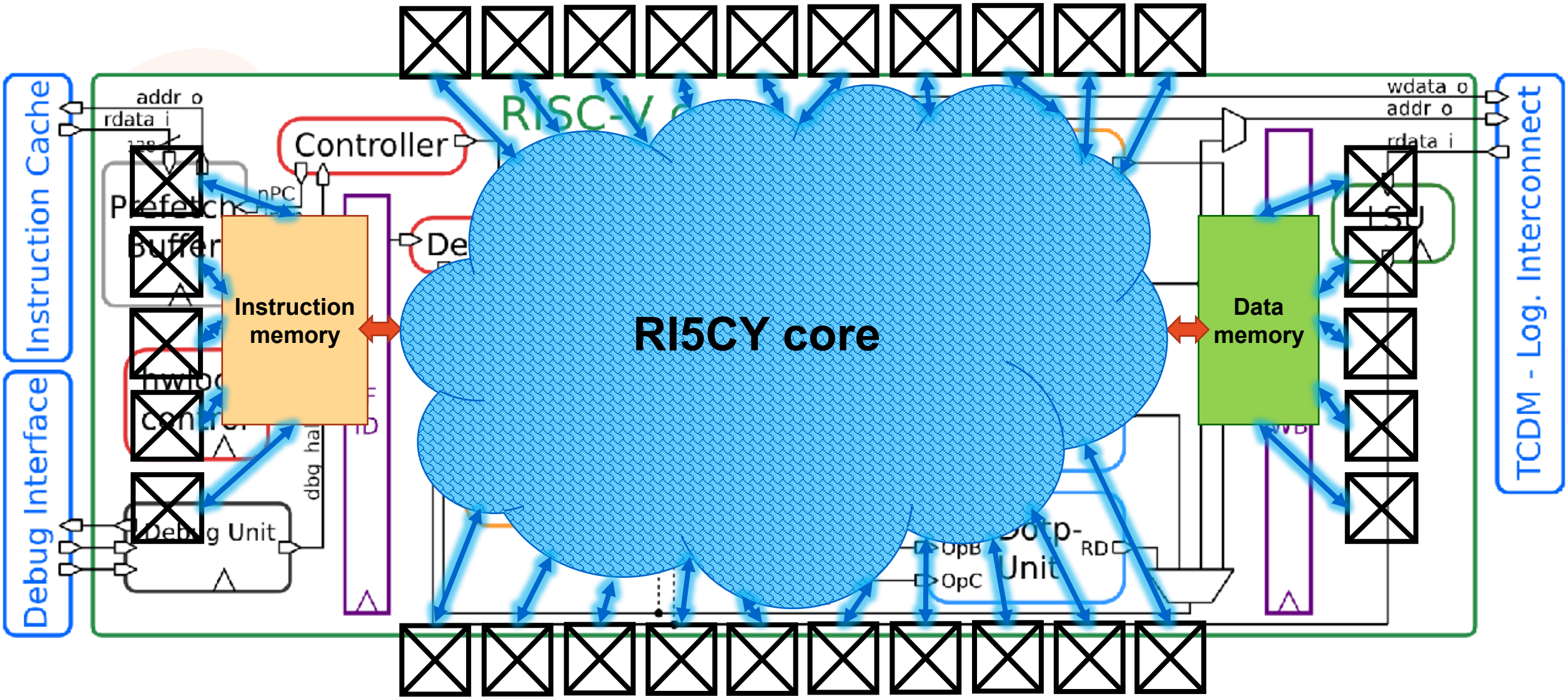
Introduction

- **The previous parts of the series, covered ran RTL2GDS on a simple block:**
  - We understood the RTL.
  - We ran directed tests to verify functionality.
  - We ran synthesis to get a gatelevel netlist.
  - We ran gatelevel simulation with timing backannotation.
  - We ran vector-based power estimation.
  - And we ran place and route.
- **However, our place and route flow was a bit "too easy":**
  - We didn't have any macros to deal with.
  - We didn't have to create an I/O ring.
  - Everything was small and simple.
- **Now we will go for a slightly more complex design… an SoC.**

Our SoC overview

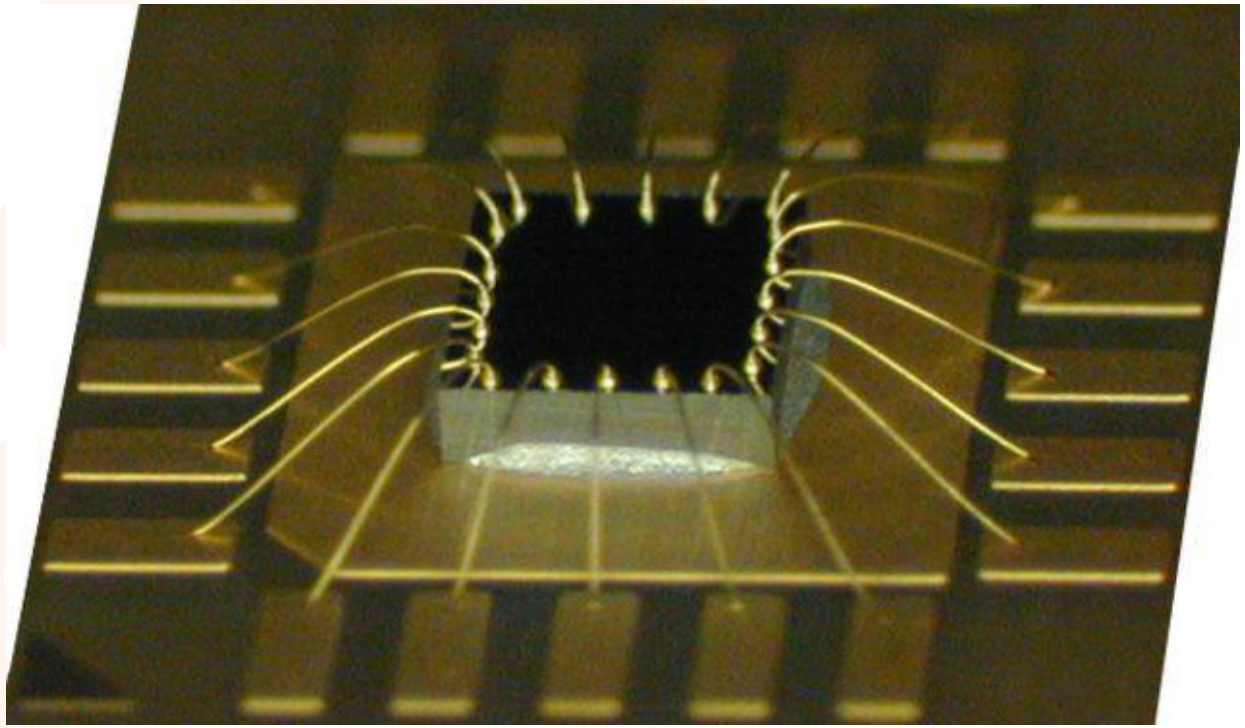
- **An SoC is a "system-on-chip", which is basically a chip with various different components, including control and memory.**
- **In this case, we are implementing a tiny SoC, consisting of:**
  - A CPU – in this case a small RISC-V core.
  - Memories – instruction and data memories that are tightly coupled to the CPU.
  - Inputs and Outputs – I/Os that let our chip communicate with the outside world.
- **This is a very minimal and basic SoC, but it is a wonderful example:**
  - It is a real working system that can run compiled C code.
  - It has RTL files made up of many modules and pieces.
  - It has SRAM blocks that need to be placed in the floorplan.
  - It has I/Os that need to be constructed into a functional I/O ring.

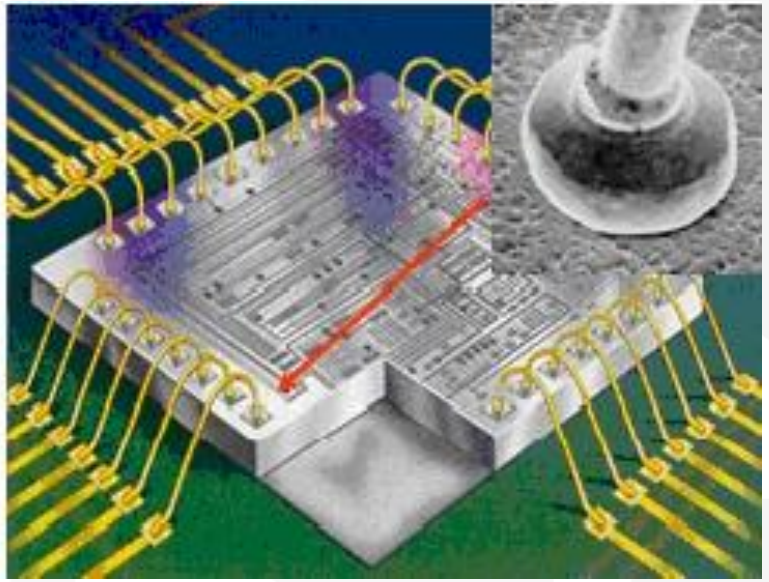
RISC-V core from PULP group = RI5CY



sourcecode/rtl/sos/pulpino/riscv_core.sv

Wire bonding to package



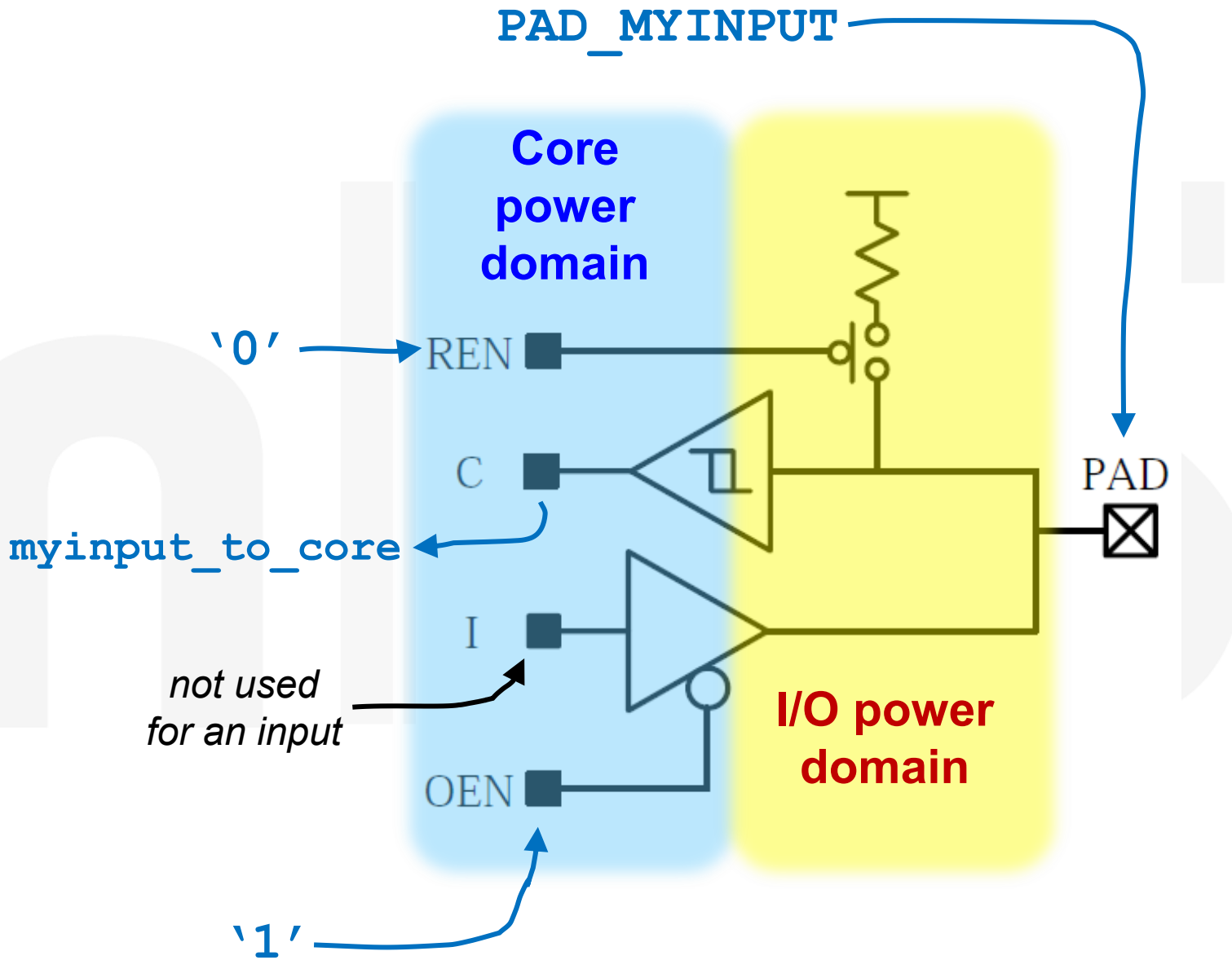


The I/O Ring

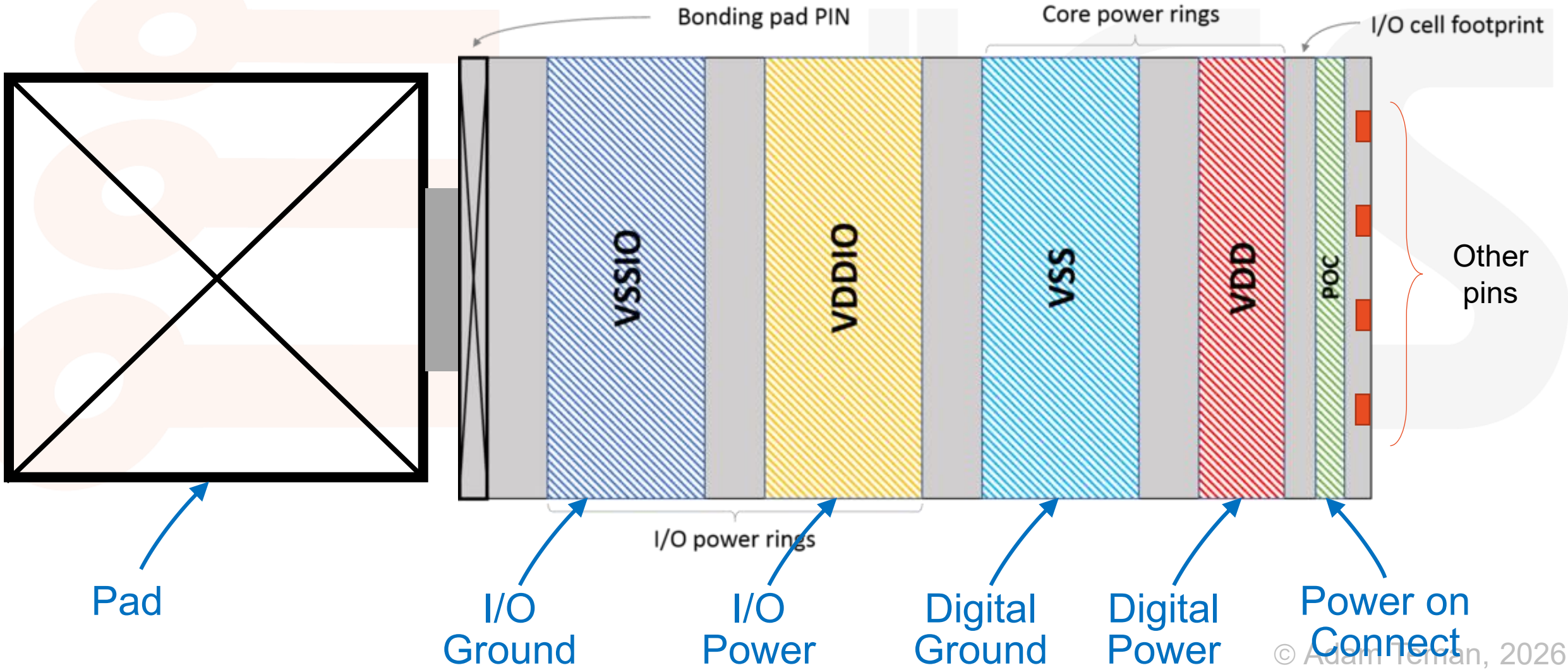
- **The I/O Ring connects the outside word (off-chip) to the SoC (on-chip)**
- **For this we need:**
  - Pads: big pieces of metal that we can connect wire bonds to.
  - Digital I/Os: Buffers that contain level shifters and ESD protection.
  - Analog I/Os: Wires with ESD protection that connect to the outside world.
  - Power Supply cells: I/Os for connecting bias voltages to the chip.
- **The Digital I/Os just propagate signals. They don't usually carry out logic.**
  - The external signals connect to the Pad.
    Therefore, we call the `inputs/outputs` "`PAD_xxx`" and use CAPITAL letters.
  - The internal signal connects to the core.
    Therefore, we call the internal wires "`xxx_to_core`".
  - The `ioring.v` file instantiates I/Os from the library and is not synthesized.

Digital I/O cell

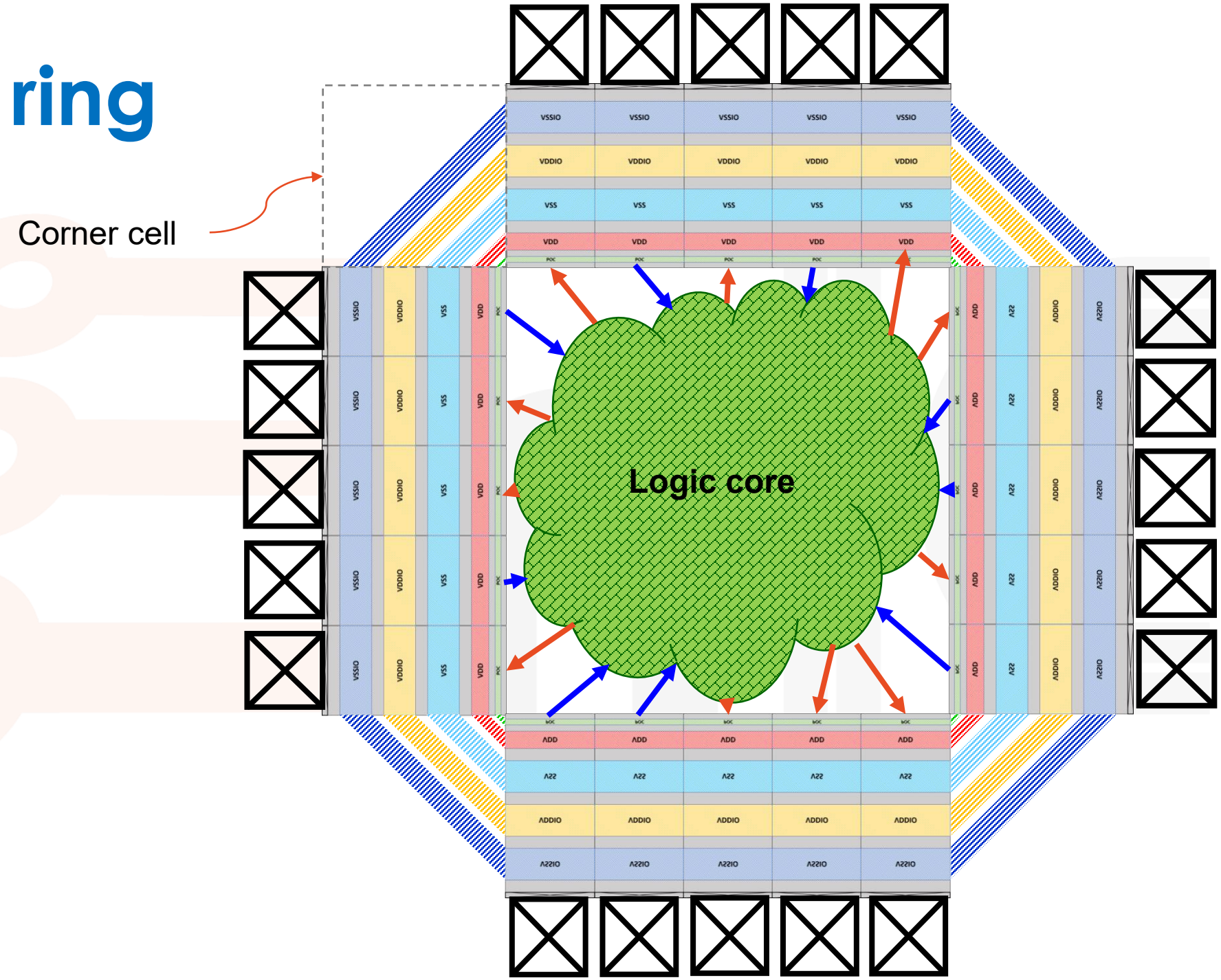
PORT NAME	DESCRITPION
REN	Pull-resistor enable
OEN	Output enable
C	Output signal to core
I	Input signal from core
PAD	Signal pin on pad side



I/O cell with internal power stripes



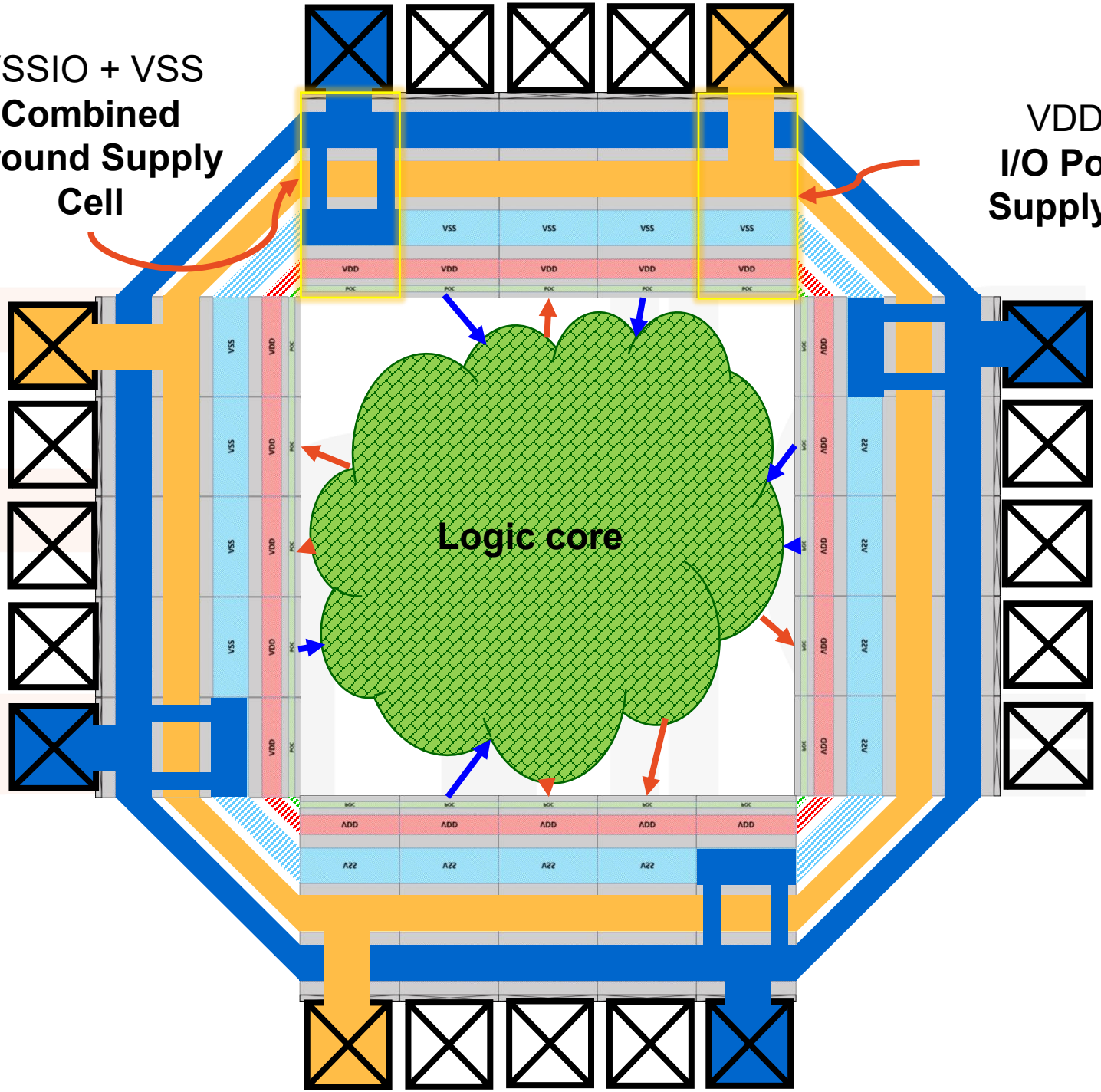
I/O ring

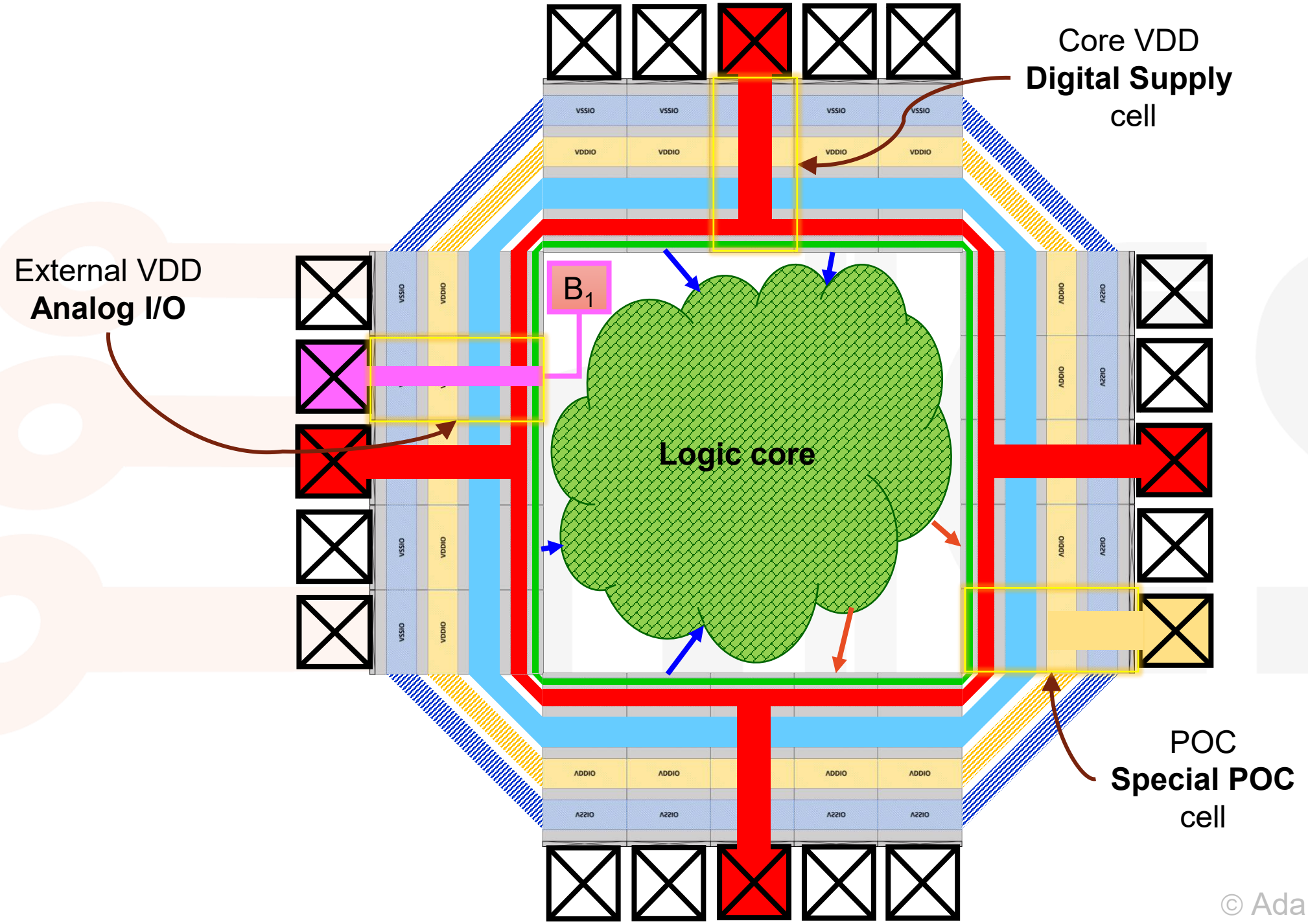


I/O ring

VSSIO + VSS
Combined Ground Supply Cell

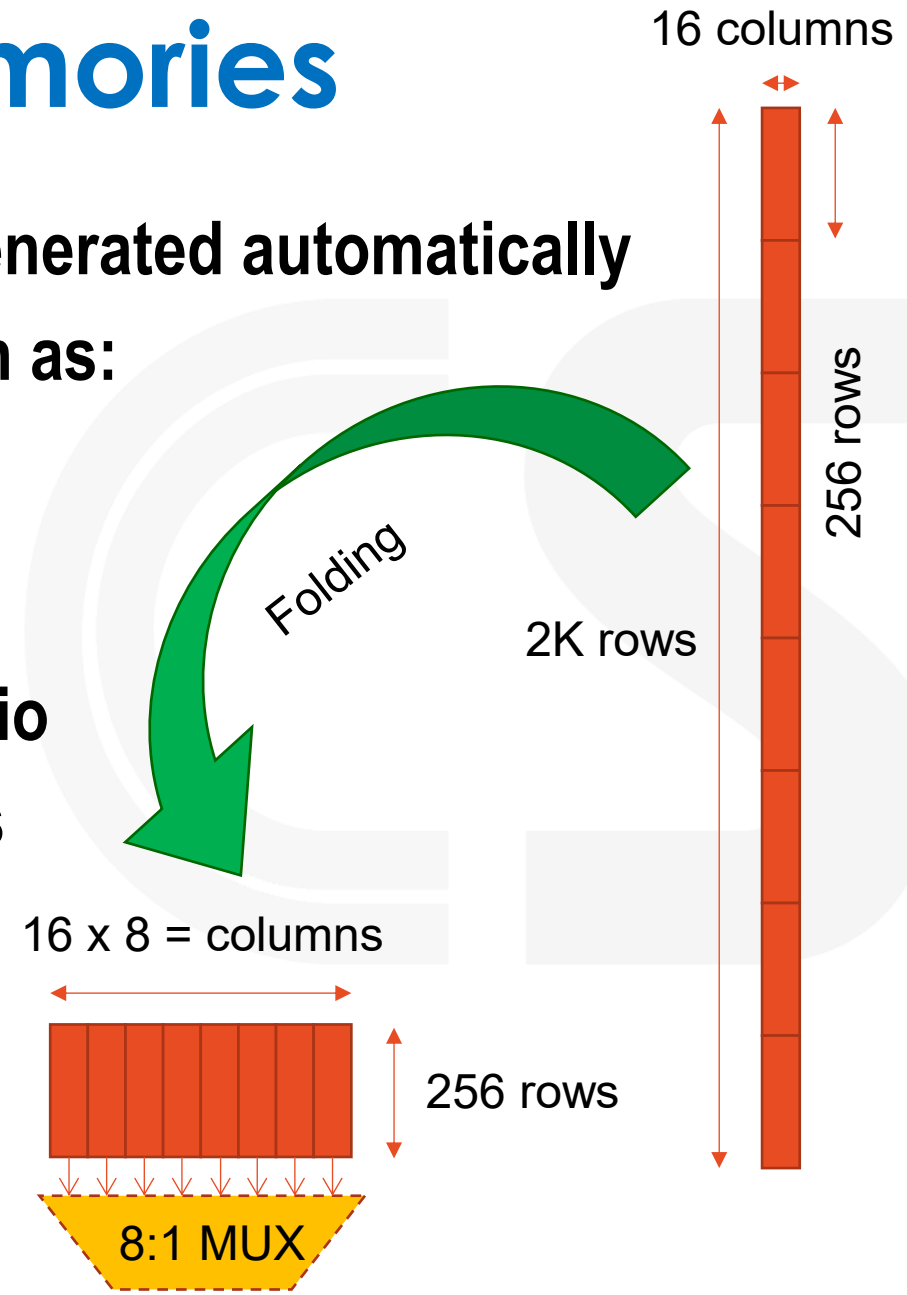
VDDIO
I/O Power Supply cell





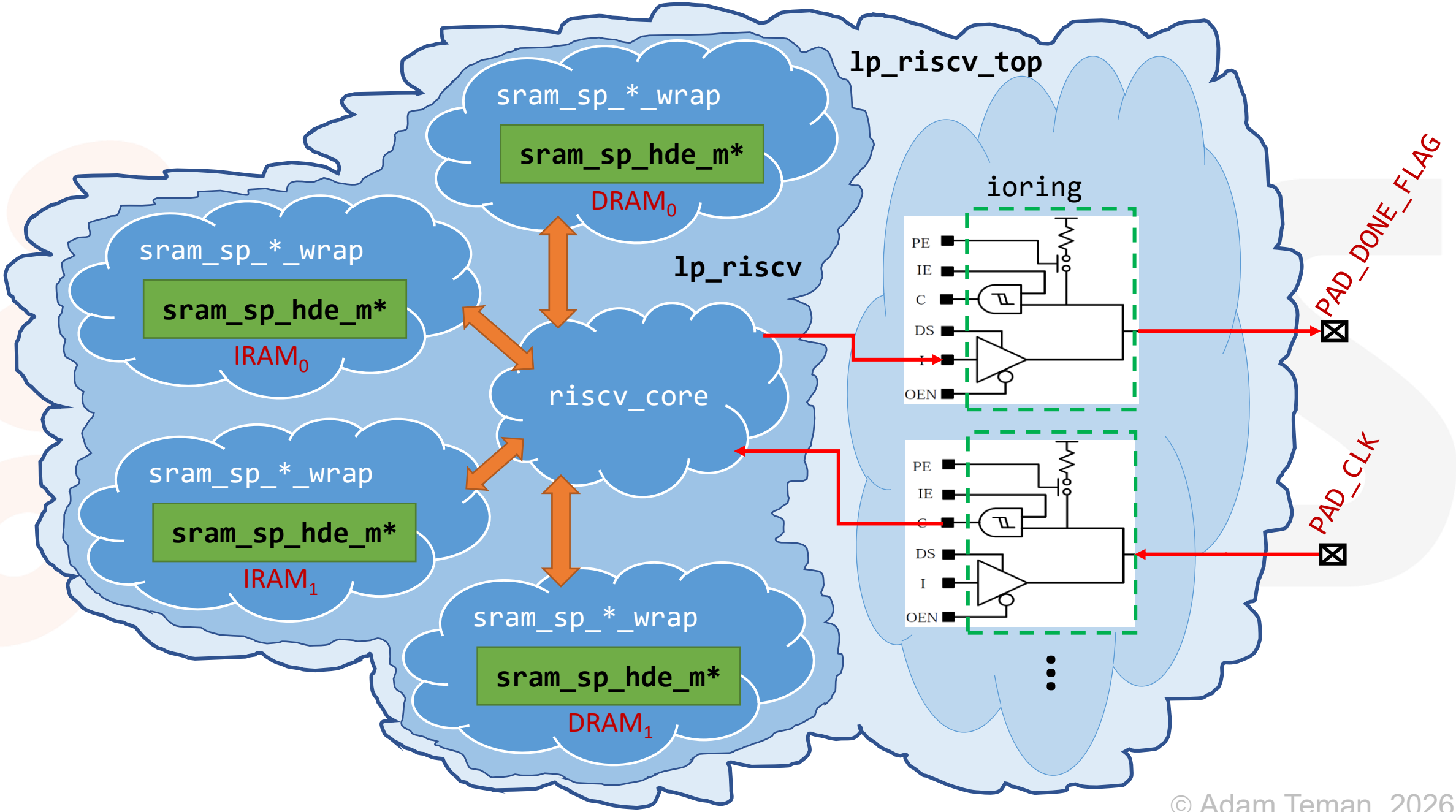
Compiled vs. STD cells memories

- **A compiled memory is a large array of bitcells – generated automatically**
- **Compiled SRAM arrays have various options, such as:**
  - BIST
  - ECC
  - Redundancy
- **Memory array might be folded for better aspect ratio**
- **Example: 2K word x 16 → 256 rows x 128 columns**
  - Select 16 output bits from the 128 columns
  - Requires 8:1 column 16 bits wide multiplexer
- **Memory compilers supply all needed views:**
  - `.libs`, `.lefs`, etc.



Wrappers

- **Compiled memories (and other hard IPs) have specific features and interfaces.**
- **To abstract away the specific implementation, we often use a "wrapper"**
- **In the example:**
  - `lp_riscv_top.v` instantiates wrappers for instruction and data memories:
    `sram_sp_instr_wrap.v` `sram_sp_data_wrap.v`
    These wrappers have generic memory interfaces (clk, address, data…)
  - A second level of wrapper abstracts away the SRAM cut size.
    For example, a 32k word data memory is composed of two 16k word cuts:
    `sram_sp_16384x32_m16_be_wrap.v`
    This wrapper muxes the two cuts according to the address.
  - The actual technology-specific memory cut is instantiated in a third wrapper:
    `sram_sp_16384x32.v`
    This wrapper contains the specific interface with all the extra signals.



RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.2: Running Compilation

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Running Compilation

- **To demonstrate that we are using a real functional design, we will now compile a simple C program, load it into the SoC and run it.**
- **To run a program, we need a *software toolchain***
  - We will use GCC to compile, assemble, and link our program.
  - We provide a set of bare-metal libraries for functions like `printf`.
  - We provide some scripts and make files to run the compilation flow.
- **In our workspace, we use the `apps/` folder for running our toolchain**
  - `sw_utils/` `libs/` and `ref/` folder contain scripts, libraries, link maps, etc.
  - We'll make a new folder for our program and write our C code in a file.
- **To compile the program, go to the folder where the program is stored and:**

```
…/sw_utils/comp_app_local.sh <program name>
```

Simulating the (large) memories

- **The memories are provided as hard macros, i.e., custom-designed blocks.**
- **For simulation, we are provided with a behavioral model (`.v` file)**
  - In the testbench RTL source list, we read in this file.
  - In the synthesis source list, we provide a `.lib` of the memory.
- **For simulation, we "preload" the memories:**
  - During compilation, we create images of the instruction and data memory content needed to run the program.
  - Writing data to all the memory content would take a long time in simulation.
  - Instead, we provide a Verilog "`task`" that reads the memory content from an initial state file.
  - This is done by adding `+MEMLOAD=PRELOAD` to the `xrun` command.

RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.3: Synthesis

Prof. Adam Teman

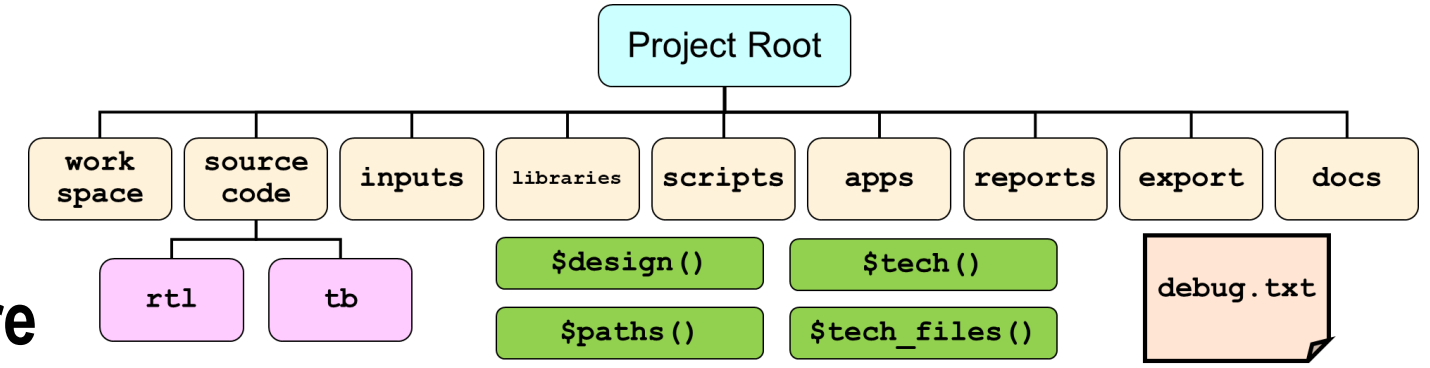
EnICS Labs, Bar-Ilan University

www.enicslabs.com

EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Synthesis



- **Before starting, we need to prepare our workspace, as we did with the simple example.**
- `libraries/` **folder:**
  - We already setup technology and standard cells in the previous example.
  - We should now provide definitions for our I/O library in `libraries.<IO>.tcl`
  - We also need to define the SRAMs we compiled in `libraries.<SRAM>.tcl`
- `inputs/` **folder:**
  - `<top>.defines.tcl`: As previously, set up target period, ports, etc.
  - Also provide instance names of hard macros (SRAMs) in `<top>.defines.tcl`
  - Setup SDC (`<top>.sdc`) and if any necessary change to MMMC (`<top>.mmmc`)
- **And now you can go ahead to synthesis.**

Some important things to check

- **During synthesis, many things are reported to the log file and/or written out as saved reports.**
- **Here are a few things that you should make sure to check carefully:**
  - Warnings during the `init_design` stage about things like `lib`s, `lef`s, RTL.
  - Unresolved instances following elaboration:
    ```
    check_design -unresolved
    ```
  - General design checks after elaboration:
    ```
    check_design
    ```
  - Timing linter on SDC after `init_design`:
    ```
    check_timing_intent
    ```
  - And finally, you should check your timing very carefully after synthesis:
    ```
    report_timing
    ```

RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.4: Moving on to Innovus

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

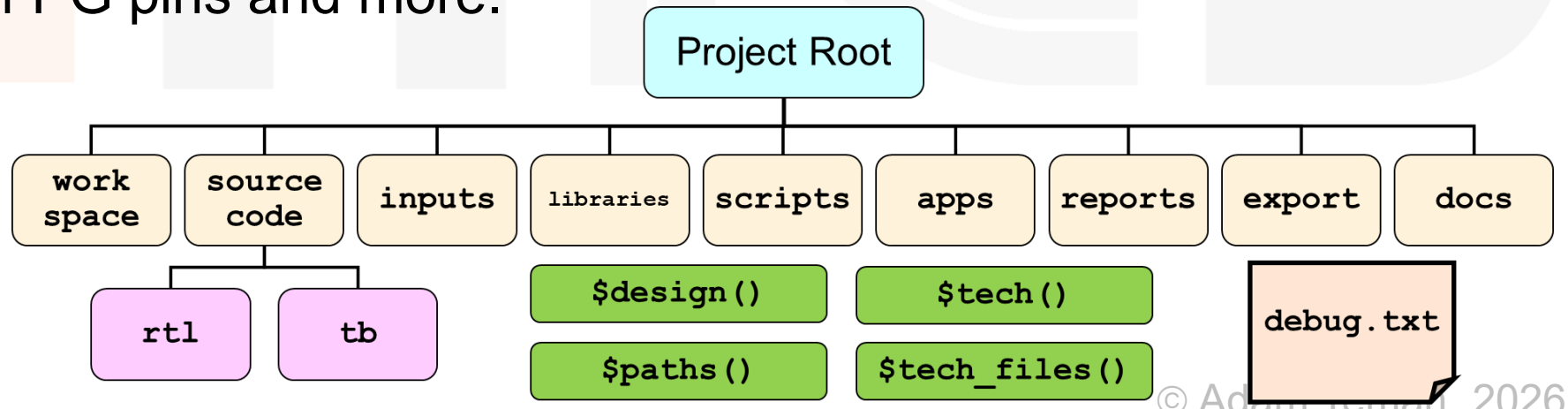
The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Moving on to Innovus

- **Having finished Synthesis, it's time to move to the physical domain:**

```
%> innovus -stylus
```

- **And of course, we start with setting up our project definitions:**
  - Read our project definitions: `<top>.defines.tcl`
  - Read in our library definitions for technology, standard cells, SRAMs, IOs and any other IPs that we are integrating into our design.
  - SRAM and I/O defines include paths to libs, lefs, etc., as well as names of PG pins and more.



The Innovus `init_design` stage

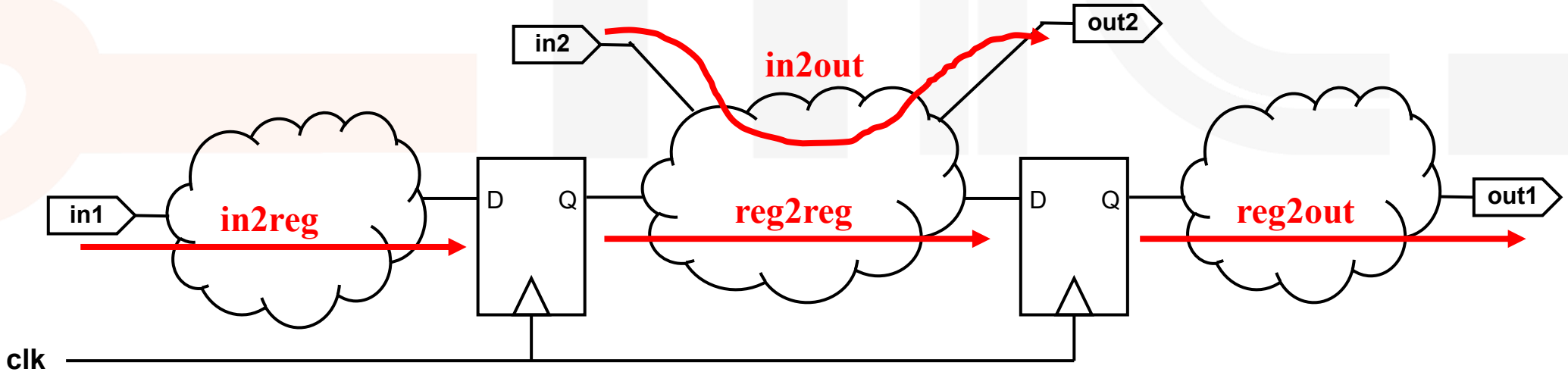
- **Our defines file set up variables that contain all the data we need:**
  - Global net names: `$design(all_power_nets)`, `$design(all_ground_nets)`
  - Library definitions: `$tech_files(ALL_LIBS_WC/BC/TC)`
  - LEF abstracts: `$tech_files(ALL_LEFS)`
  - A pointer to our post-synthesis netlist: `$design(postsyn_netlist)`
- **So we can move on to the `init_design` stage**
  - `init_power_nets`/`init_ground_nets`: define the global nets
  - `read_mmmc`: create analysis views
  - `read_physical`: read in techlef and LEF abstracts
  - `read_netlist`: read in the post synthesis netlist
- **And everything gets processed by running `init_design`.**

The initialized design hierarchy

- **After `init_design`, we can see the design structure with the Design Browser.**
- **The various levels of the design are divided into several categories:**
  - Terms: The ports/pins (terminals) of this hierarchy.
  - Nets: The nets (wires) at this hierarchy.
  - StdCells: Standard cells that are instantiated within this hierarchy.
  - Hinsts: Hierarchical instances that were not flattened during synthesis.
  - Blocks: Hard IPs, such as SRAMs, that are instantiated within this hierarchy.
  - Pads: I/O cells that are instantiated within this hierarchy.
  - PG Pins: Power/Ground pins of this hierarchy.
- **We can traverse connectivity using the design browser:**
  - Each Term and StdCell/Block/Pad pin is connected to a single net.
  - Each net is connected to one or more Terms and/or StdCell/Block/Pad pins

Path Groups

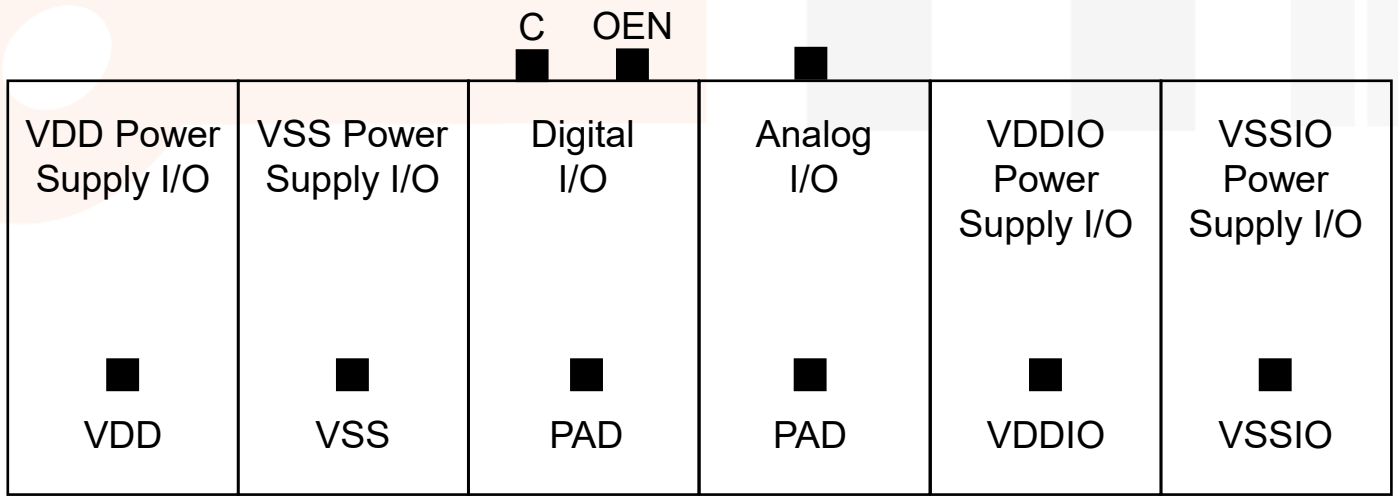
- **It is common practice to categorize timing paths into four groups.**
  - These can be defined with a single command: `create_basic_path_groups`
- **Sometimes we want to separate other paths into their own group.**
  - This is done with the `group_path` command.
- **Each path group is treated independently during optimization and reporting.**

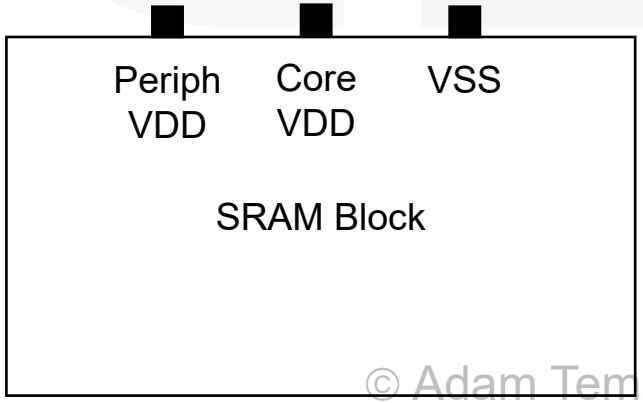


Connecting Global Nets

Remember to use the **Design Browser** to check that your **Global Net Connections** worked!

- **In the previous video, we discussed global net connections**
  - We defined global power and ground nets
  - We only had one of each and they connected to all standard cells.
- **We now have additional global nets connected to the I/Os and SRAMs:**
  - The I/O ring usually has an additional higher voltage (VDDIO). Global nets connect to Power Supply cells and are abutted to other I/Os
  - SRAMs often have separate voltages for memory core and periphery.





RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.5: Floorplanning

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

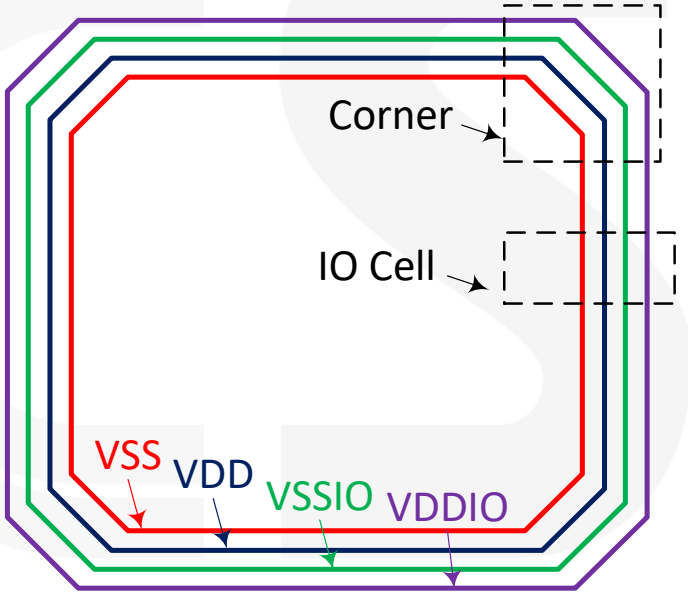
www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

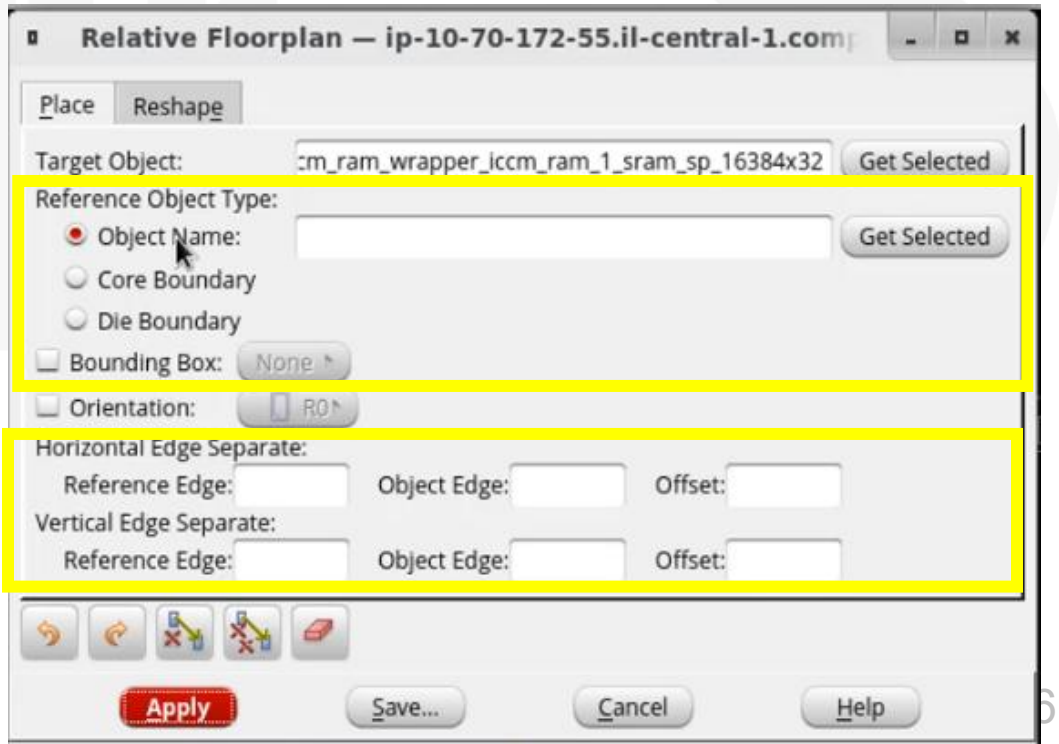
Full Chip Floorplanning

- **At the block level, our design connected to the outside world through pins**
  - We used the `edit_pins` command to locate them around the block.
- **At the full chip level, we have an I/O ring**
  - All of our ports are connected to instantiated I/Os.
  - The I/Os usually have a particular `site` definition and are placed around the perimeter of our chip.
  - We define the location of each I/O with a special I/O file.
- **The I/Os have power/ground rings running through them**
  - The rings are connected through abutment.
  - We need to use I/O fillers to continue these connections in the empty space.
  - We need special Corner cells to connect the rings around corners.



Placing Hard Macros

- **The two primary ways of placing hard macros are:**
  - Manually placing them by using the "move" tool (and copying the resulting `update_floorplan_obj` command)
  - "Relative Floorplanning: Floorplan→Relative Floorplan→Edit Constraint (or `create_relative_floorplan`)
- **Place relative to a "reference object"**
  - Core/Die boundary
  - Other object
- **Place vertically/horizontally relative to an edge with a given offset.**

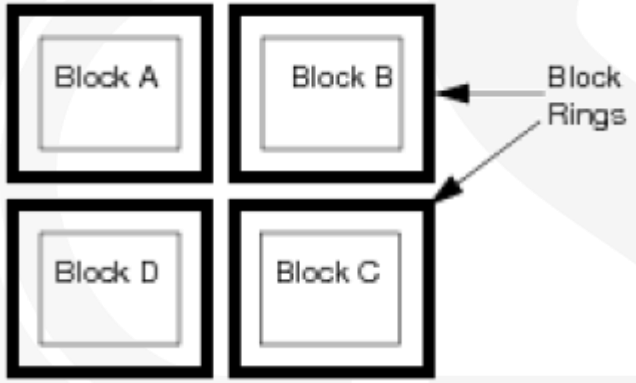


Adding Rings and Halos to Macros

- **Adding a ring to a hard macro (e.g., SRAM) has the following benefits:**
  - Easy to connect macro PG pins to ring.
  - Easy to terminate power mesh near macro.
- **But this comes at the expense of wasted area**
  - Therefore, often SRAMs are connected by dropping vias from above.
- **To add rings to a macro:**
  - Select “Ring Type” = “Block ring(s) around”

    ```
    add_rings -around selected -type block_rings ...
    ```
  - Optionally add one set of rings around many macros.
- **To ensure no standard cells close to macro edges, create a placement halo:**

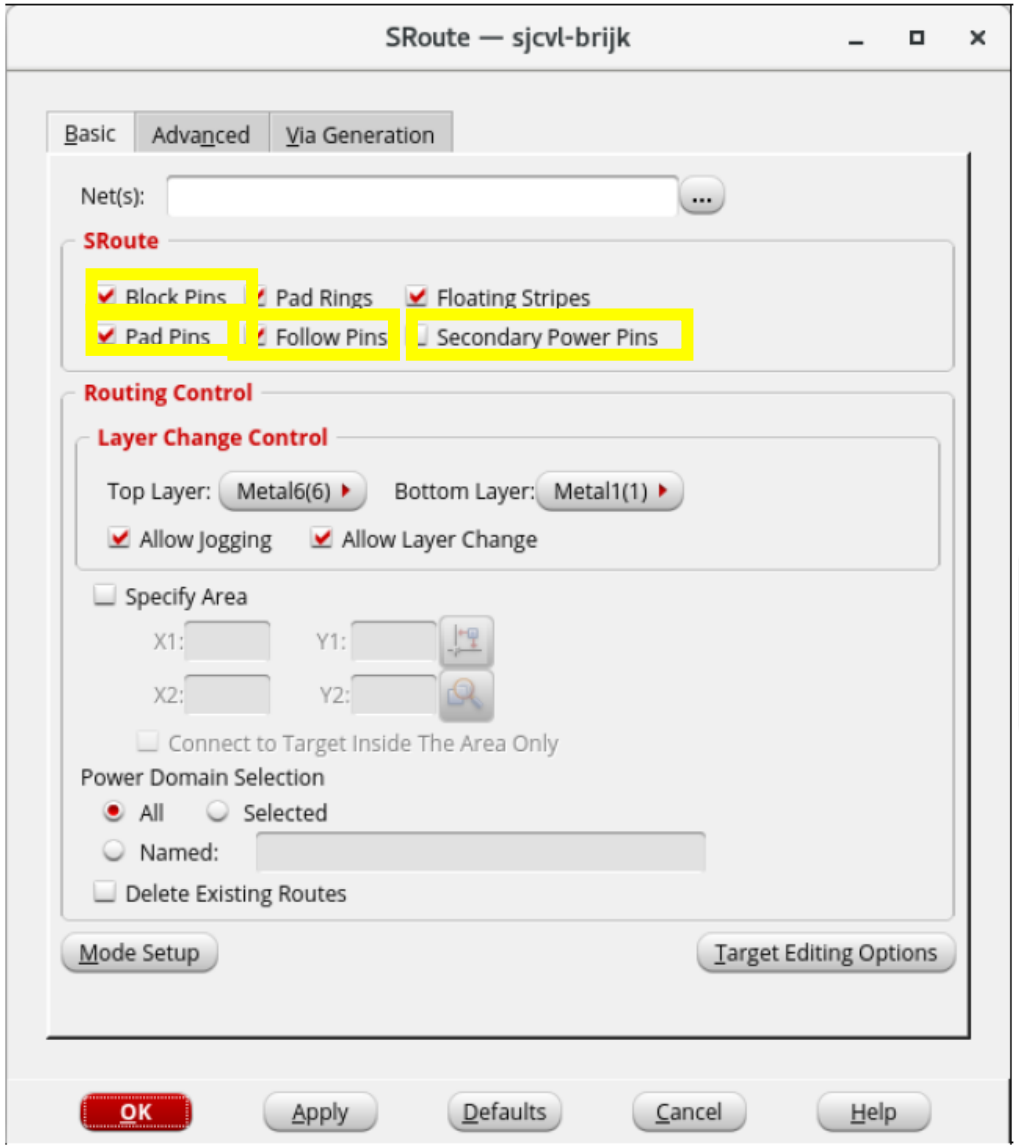
  ```
  create_place_halo -insts ...
  ```



Special Route Revisited

- **SRoute is a router for making PG connections to pins and stripes**
  - Follow Pins: Routes Standard Cell rails
    `route_special -connect core_pin`
  - Block Pins: Connects macro PG pins
    e.g., SRAM VDD/GND pins to rings.
    `route_special -connect block_pin`
  - Pad Pins: Connects I/Os to core ring
    `route_special -connect pad_pin`
  - Secondary Power Pins: For connecting pins of level shifters that are not in the PG rails.
- **Check connectivity of special route only:**

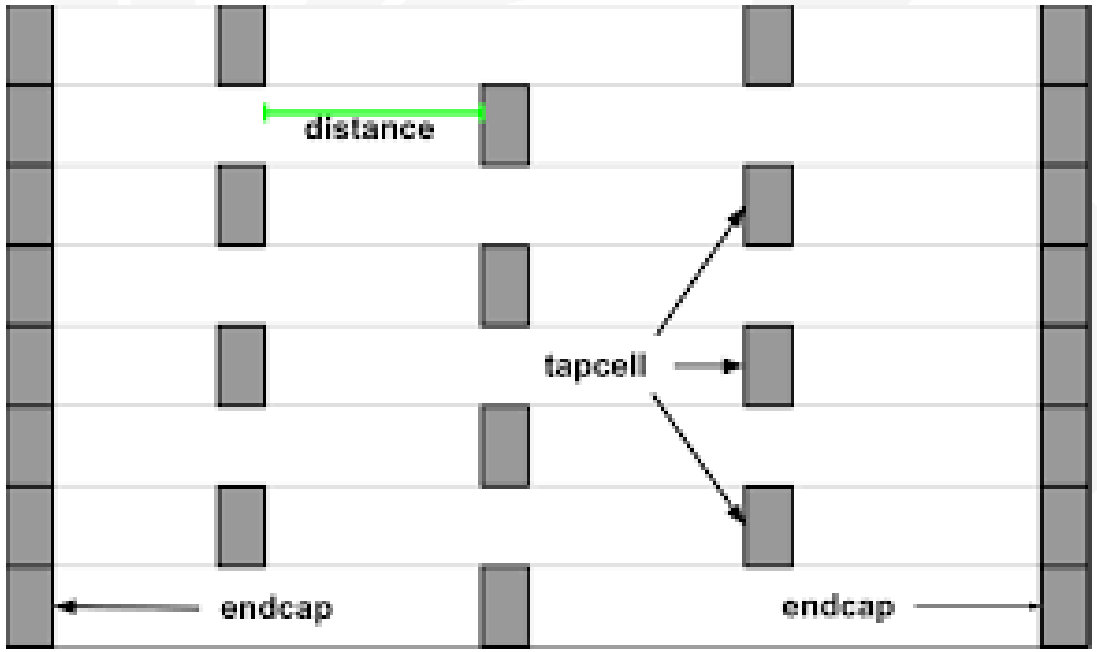
```
check_connectivity -type special
```



Reminder: End Caps and Well Taps

- **End Caps (or boundary cells):**
  - Are cells required at the end of standard cell rows (and in advanced processes, at the top, bottom, corners, etc.)
  - Include POLY, OD, NWELL and other layers for DRC and density.
- **Well Taps (or Tap Cells):**
  - Are cells that connect VDD/GND to NWELL/PWELL to prevent latch up.
  - DRC rules require tap cells every several microns.

```
set_db add_endcaps_left_edge $ENDCAP_LEFT
set_db add_endcaps_right_edge $ENDCAP_RIGHT
add_endcaps -prefix ENDCAP
```



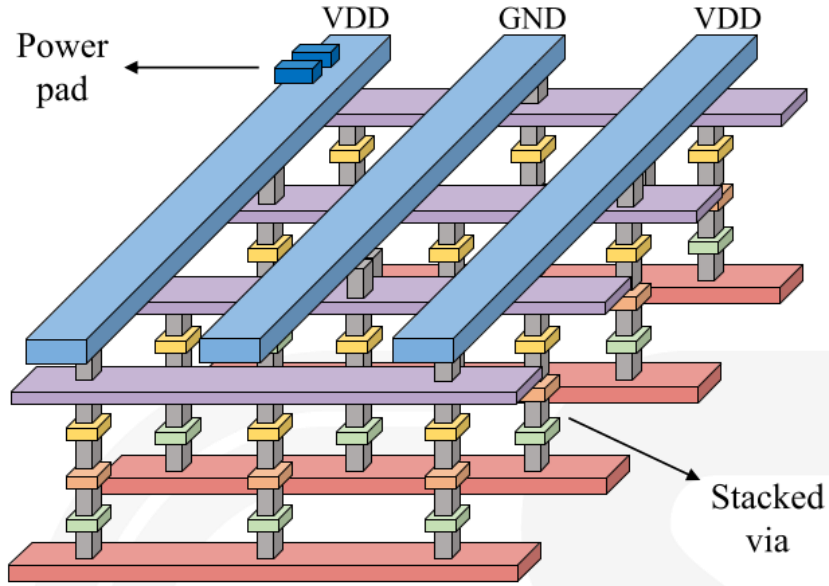
Source: OpenRoad

```
add_well_taps -cell $WELLTAPCELL \
        -skip_row 1 -prefix WELLTAP \
        -in_row_offset 3 -cell_interval 10
```

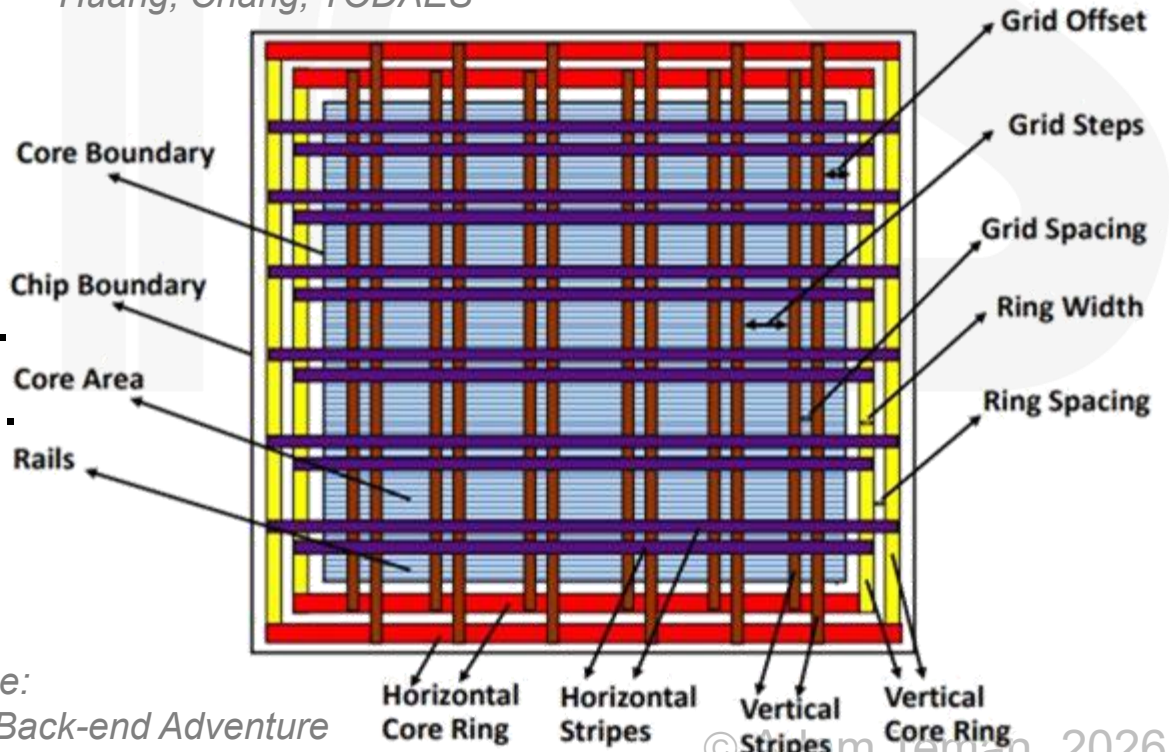
```
check_well_taps -max_distance 20
```

Power Planning

- ~~One of the most important~~
  The most important part of floorplanning is creating the power grid.
- The goal of the power grid is to provide VDD/GND to all circuits, while:
  - Minimizing (static) **IR Drop**.
  - Minimizing (dynamic) **dI/dt drop**.
  - Meeting **Electromigration** requirements.
  - Leaving enough room for **signal routing**.
- This is done with power rail analysis tools (e.g., **Voltus**, **RedHawk**), but that is beyond the scope of this demo.



Source: Huang, Chang, TODAES



Source: VLSI Back-end Adventure

RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.6: Placement

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Placement

- **After the floorplan is ready, it is time to move on to standard cell placement.**
- **This is done with the timing-aware GigaPlace tool:**

```
place_opt_design
```

  - Runs pre-placement optimization
  - Runs standard cell placement
  - Performs scan tracing and reordering
- **The `place_opt_design` command replaces the legacy two-staged placement:**

```
place_design
opt_design -pre_cts
```

Congestion and Blockages

- **Following placement, we can evaluate congestion based on trial route:**
- **To fix congestion try:**
  - Running congestion-driven placement
  - Adding padding to modules
  - Creating placement blockages and module constraints

```
report_density_map
```

```
set_db place_global_cong_effort high
```

```
set_db place_global_module_padding XXX
```

- **There are three types of placement blockages:**
  - **Hard blockages**:
    Areas that cannot be used for standard cell placement.
  - **Soft blockages**: Blocked during placement, but can be used during optimization, CTS, ECO, legalization
  - **Partial blockages**:
    Limits the placement density (utilization) in the area

```
create_place_blockage \
 -polygon|rects XXX   \
 -type hard|soft|partial
```

Module Constraints

- **Beyond placement blockages, modules can be constrained to be placed within a certain area using the `create_boundary_constraint` command.**

```
create_boundary_constraint -hinst <hinst_name> \
      -polygon|rects XXX -type fence|region|guide|cluster
```

- **There are four types of module constraints:**
  - **Guide**: Guides the placer to locate the module in the specified area. This is a "soft constraint".
  - **Fence**: Ensures that a module is placed inside the specified area. This is a "hard constraint". No cells from other modules can be placed here.
  - **Region**: Like a Fence, ensures that the module is placed inside the specified area, but allows placement of instances from other modules within the region.
  - **Soft Guide/Cluster**: Similar to a guide, but there are no fixed locations.

Post placement reporting

- **Following placement, we should analyze the results.**
- **Innovus provides several reports and tools for this, including:**
  - `check_place`: Checks for overlaps, off-grid, and unplaced cells.
  - `report_area`: Standard cell area in each hierarchical module.
  - `report_density_map`: Generates a density map and report.
  - `report_pin_density_map`: Generates a pin density map and report.
  - `report_congestion`: Reports average congestion and hotspot score.
- **And, of course, we should check timing.**
  At this point, setup timing with an ideal clock is usually sufficient.

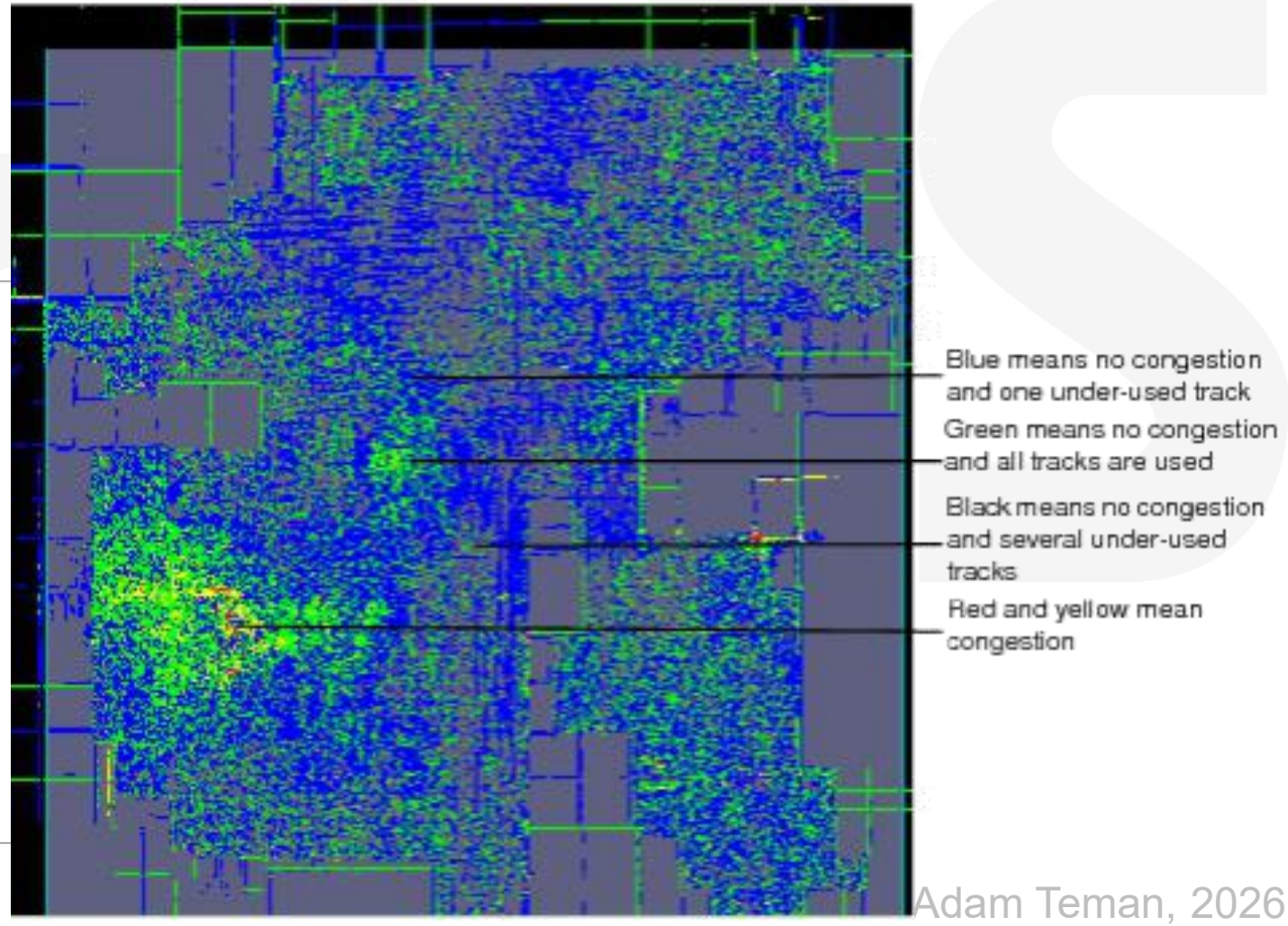
```
report_timing
```

```
time_design -pre_cts -ideal_clock
```

Check congestion

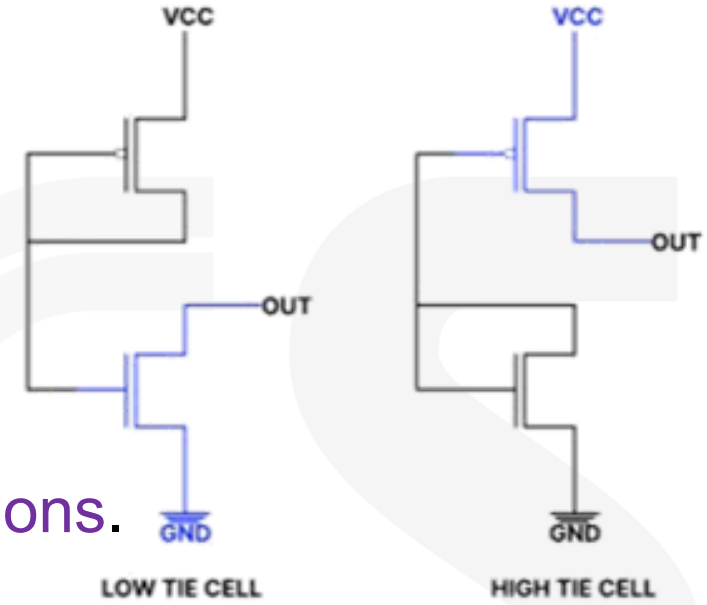
- **After global routing (trial route) check the congestion report or visualize with:**
  VIEWS→PHYSICAL VIEW→
  →ALL COLORS→VIEW ONLY→
  →CONGESTION

```
Congestion Analysis:
                OverCon       OverCon       OverCon       OverCon
                 #Gcell        #Gcell        #Gcell        #Gcell     %Gcell
Layer              (1-2)         (3-4)         (5-6)        (7-12)   OverCon
--------------------------------------------------------------------------------
Metal 1        22(0.01%)     10(0.00%)      0(0.00%)      0(0.00%)   (0.01%)
Metal 2      5531(2.39%)   1680(0.73%)    370(0.16%)    123(0.05%)   (3.33%)
Metal 3      4114(1.78%)     19(0.01%)      0(0.00%)      0(0.00%)   (1.79%)
Metal 4      1333(0.58%)    137(0.06%)      0(0.00%)      0(0.00%)   (0.64%)
Metal 5      5852(2.53%)      4(0.00%)      0(0.00%)      0(0.00%)   (2.53%)
Metal 6        27(0.01%)      0(0.00%)      0(0.00%)      0(0.00%)   (0.01%)
--------------------------------------------------------------------------------
  Total     16879(1.22%)   1850(0.13%)    370(0.03%)    123(0.01%)   (1.39%)

#Max overcon = 12 tracks.
#Total overcon = 1.39%
#Worst layer Gcell overcon rate = 2.53%
```

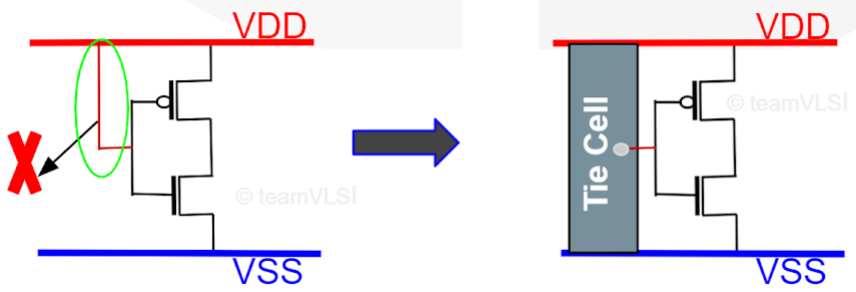


Tie-Off Cells

- **Following synthesis, there are various gates or macro pins connected to constant values (‘0’ or ‘1’).**
- **However, in nanoscaled technologies, we are not allowed to directly connect VDD/GND to gates of transistors.**
  - Thin oxides sensitive to surges, as well as antenna violations.
- **Therefore, after placement, we need connect constants with “tie cells” using the `add_tieoffs` command**
  - Tie-Hi – connect VDD (`1'b1`)
  - Tie-Lo – connect GND (`1'b0`)
- **Design considerations**
  - Maximum distance of tie cell from gate.
  - Maximum fanout for a tie cell



Source: Tomodachi Kushagra



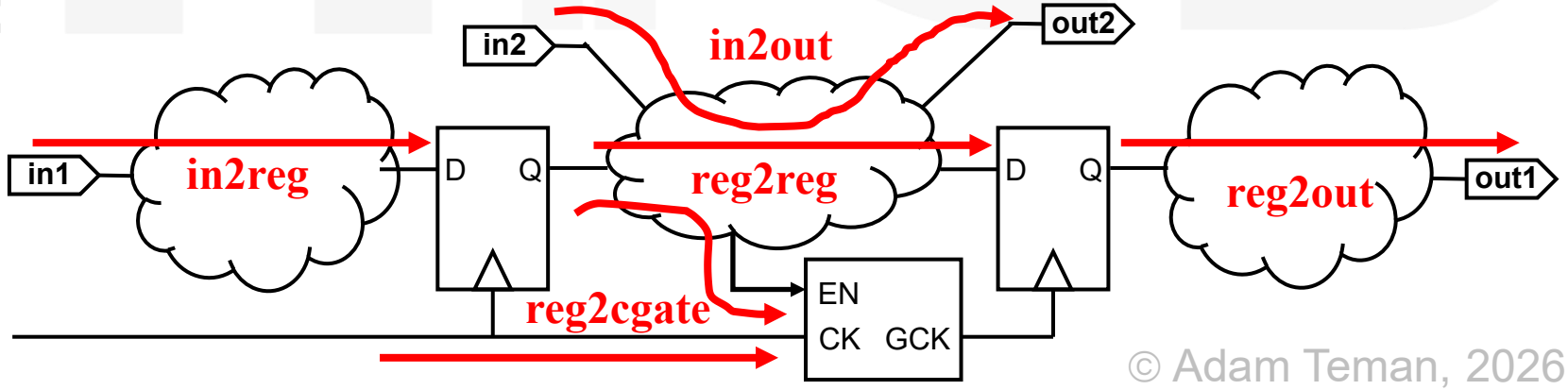
Source: Team VLSI

Timing Optimization

- **Timing optimization is applied throughout the flow after every major step:**
  - After Placement (a.k.a., Pre-CTS)
  - After Clock Tree Synthesis
  - After Route

```
opt_design -pre_cts
opt_design -post_cts
opt_design -post_route
```

- **Depending on the stage, timing optimization includes:**
  - Adding buffers
  - Resizing gates
  - Restructuring the netlist
  - Remapping logic
  - Swapping pins
  - Deleting buffers
  - Moving instances
  - Applying useful skew
- **Default timing categories:**
  - reg2reg
  - in2reg, reg2out, in2out
  - reg2cgate



Timing Optimization

- **After running optimization, a final summary will be printed out:**

- **Instances and nets that are added during optimization are named according to a convention, starting with "`FE`"*.**
- **For example:**
  - `FE_OCP_RBC`: Instance added by rebuffering
  - `FE_RC`: Instance added by netlist reconstruction
  - `FE_RN`: Net added by netlist reconstruction
  - `FE_USKC`: Instance added for useful skew

Note that these names can be customized using `set_db` commands

```
------------------------------------------------------------
  opt_design Final Summary
------------------------------------------------------------

Setup views included:
 Setup_SSG_m40c_0p81v_RCworst_ccworst_func
 Setup_SSG_m40c_0p81v_Cworst_ccworst_func

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):| -0.675  | -0.204  | -0.226  | -0.675  |
|           TNS (ns):| -1453.1 |-532.662 | -5.301  | -1078.5 |
|    Violating Paths:|  7106   |  5320   |   72    |  3548   |
|          All Paths:|  45995  |  43625  |   740   |  5404   |
+--------------------+---------+---------+---------+---------+

+----------------+-----------------------------------+------------------+
|                |              Real                 |       Total      |
|    DRVs        +------------------+----------------+------------------|
|                |  Nr nets(terms)  |  Worst Vio     |  Nr nets(terms)  |
+----------------+------------------+----------------+------------------+
|   max_cap      |      0 (0)       |    0.000       |     297 (297)    |
|   max_tran     |      0 (0)       |    0.000       |      0 (0)       |
|   max_fanout   |      0 (0)       |      0         |      0 (0)       |
|   max_length   |      0 (0)       |      0         |      0 (0)       |
+----------------+------------------+----------------+------------------+

Density: 62.345%
Routing Overflow: 0.12% H and 0.04% V
------------------------------------------------------------
```

* This is legacy from one of the early versions of Innovus that was then called "**First Encounter**"

RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.7: Clock Tree Synthesis

Prof. Adam Teman

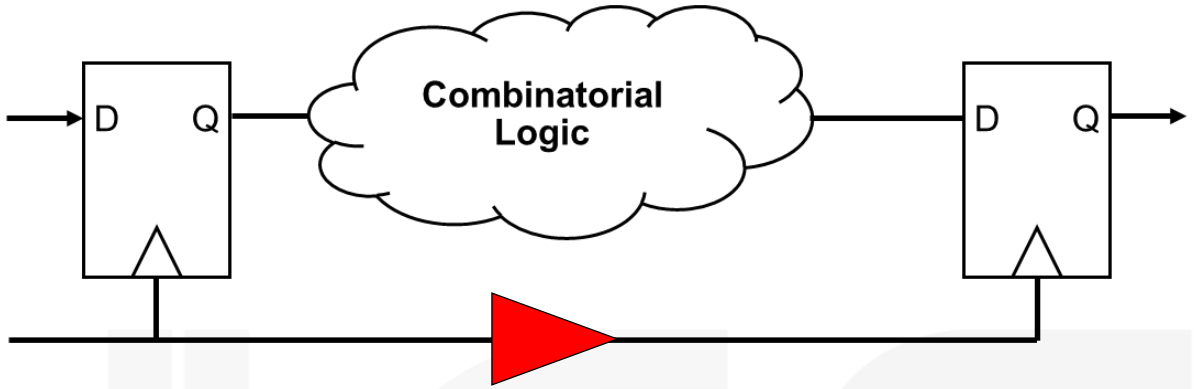
EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Clock Tree Synthesis

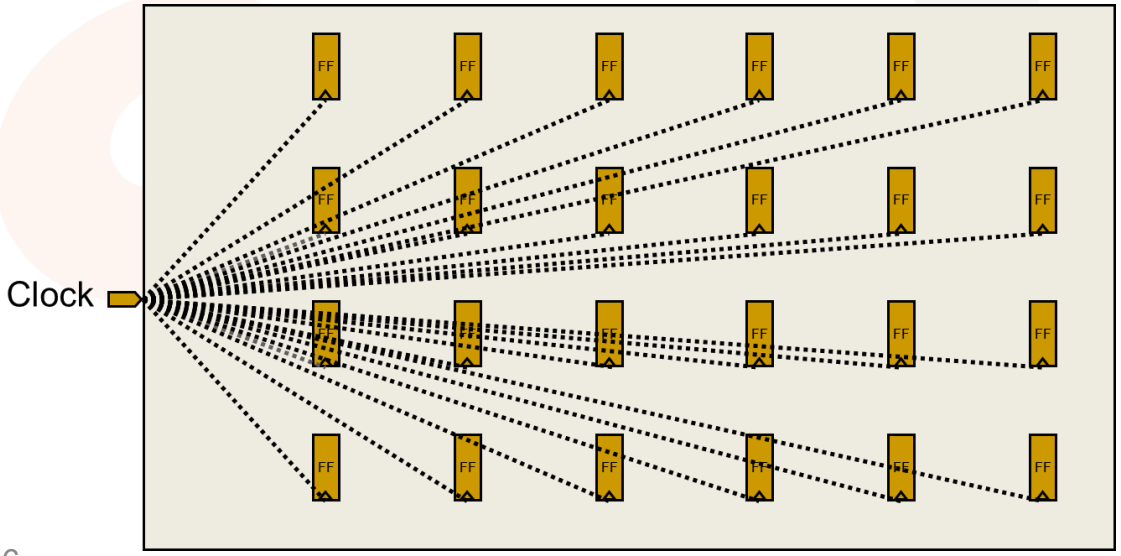


- **Until now, we had an ideal clock**
  - Max Delay:

$$T > t_{cq} + t_{logic} + t_{setup}$$

  - Min Delay:

$$t_{cq} + t_{logic} > t_{hold}$$

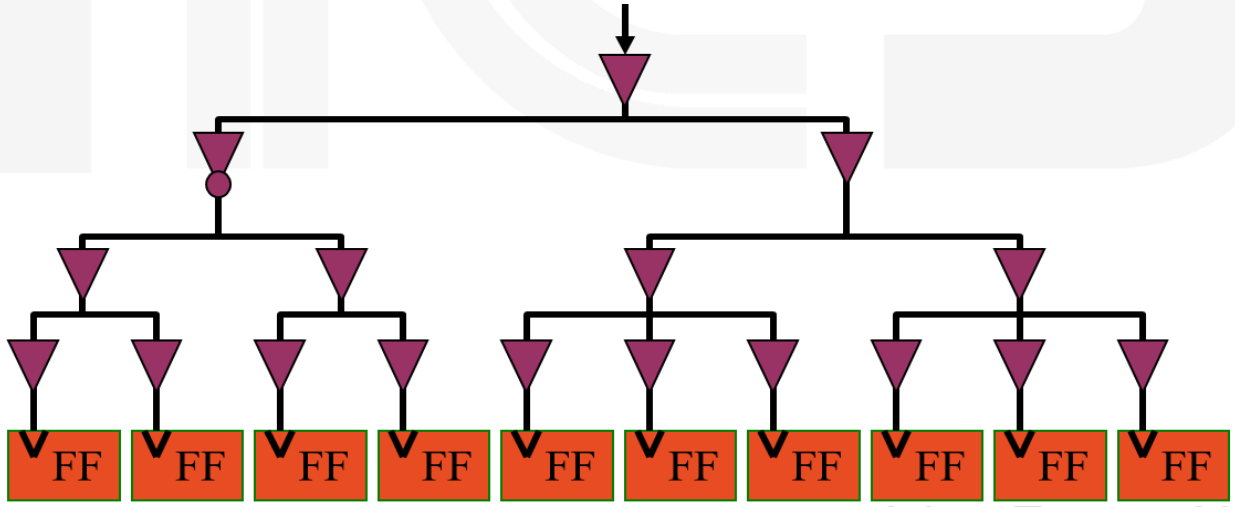


- **We now have to build a clock tree**
  - Max Delay:

$$T + \delta_{skew} - 2\delta_{jitter} > t_{CQ} + t_{logic} + t_{setup} + \delta_{margin}$$

  - Min Delay:

$$t_{CQ} + t_{logic} - \delta_{margin} > t_{hold} + \delta_{skew}$$

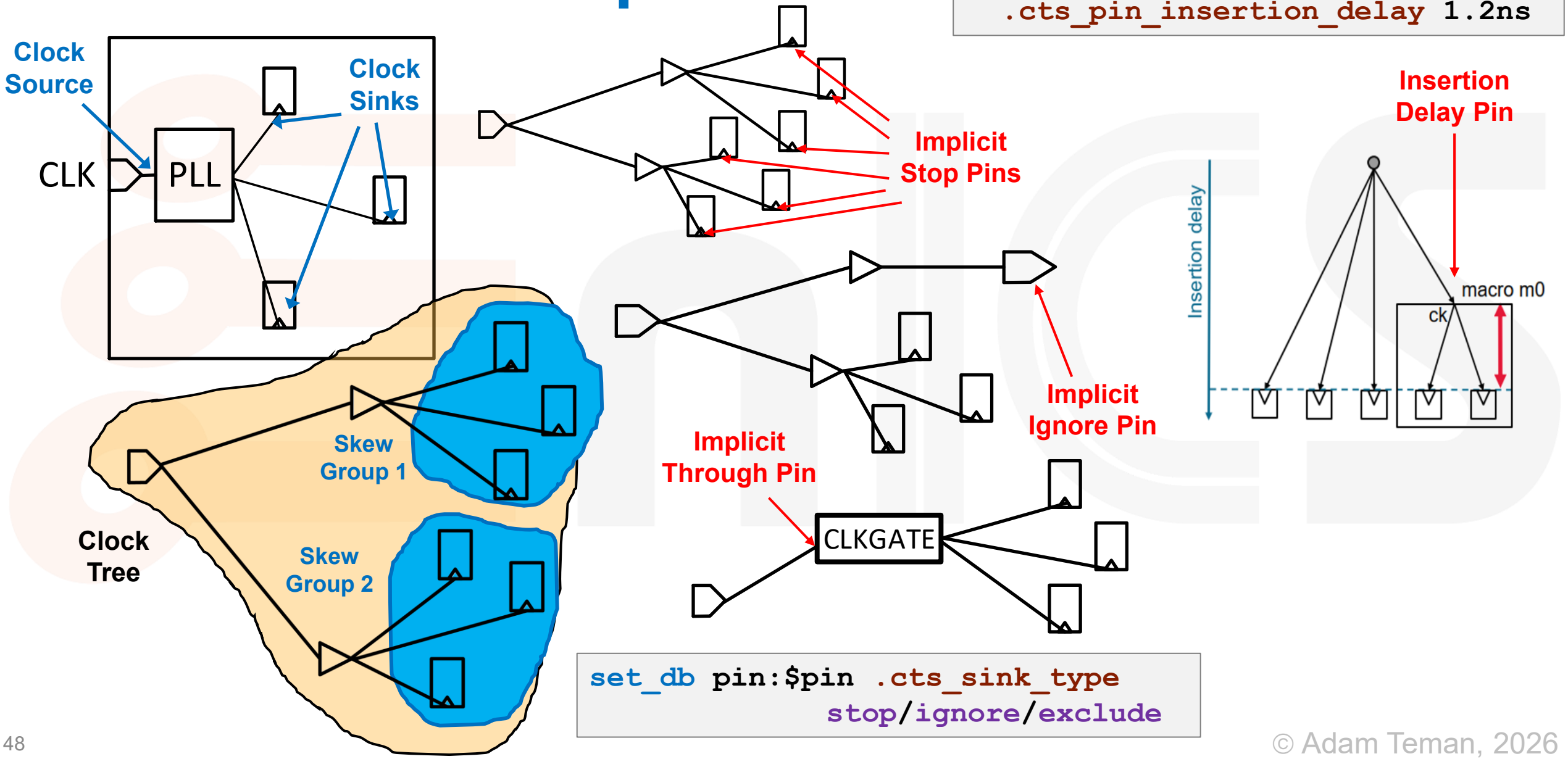


Clock Tree Spec

- There are many attributes and constraints for guiding CTS.
- They can be created automatically with: `create_clock_tree_spec`
- Example commands in clock tree spec:
  - `create_clock_tree -name clock -source CLK -no_skew_group`
  - `create_skew_group -name clock -sources CLK -auto_sinks`
  - `set_db cts_buffer_cells {BUFX12 BUFX8 BUFX6 BUFX4 BUFX2}`
  - `set_db cts_inverter_cells {INVX12 INVX8 INVX6 INVX4 INVX2}`
  - `set_db cts_clock_gating_cells {ICGX12 ICGX8 ICGX6 ICGX4}`
  - `set_db cts_use_inverters true`
  - `set_db cts_max_fanout 20`
  - `set_db cts_target_max_capacitance 0.1`
  - `set_db cts_target_max_transition_time 100ps`
  - `set_db cts_target_skew 50ps`
  - `set_db opt_useful_skew_ccopt extreme`
- In addition, various components of the clock tree can be defined.

Clock Tree Components

```
set_db pin:mem1/CK
  .cts_pin_insertion_delay 1.2ns
```




```
set_db pin:$pin .cts_sink_type
                stop/ignore/exclude
```

Shielding and Non-Default Routing Rules

Create **Route Rules**

Double Width

Double Spacing

…

```
create_route_rule -name CTS_2W2S \
      -spacing_multiplier 2   -width_multiplier 2
```

Create **Route Type**

Preferred Routing Layers

Shielding

…

```
create_route_type -name trunk_rule -non_default_rule CTS_2W2S \
      -top_preferred_layer M7 -bottom_preferred_layer M6 \
      -shield_net VSS -bottom_shield_layer M6
```

Set Route Type for:
Top
Trunk
Leaf

Leaf: Any clock net connected directly to a sink.

Trunk: Any other clock net.

Top: Clock nets with Fanout > `cts_top_fanout_threshold`

```
set_db cts_route_type_top top_rule
set_db cts_route_type_trunk trunk_rule
set_db cts_route_type_leaf leaf_rule
set_db cts_top_fanout_threshold 10000
```

Some CTS Recommendations

- **Don't forget to specify buffers, inverters, and clock gating cells.**
- **Prefer LVT cells → Lower insertion delay, better for corners/OCV.**
- **Stay away from largest and smallest clock cells:**
  - Largest ones result in higher power and can lead to EM problems.
  - Smallest ones (e.g., <= X3) are more sensitive to corners, SI, routing jogs.
  - Medium-sized cells (e.g., X4) are important so large ones aren't always used.
- **Buffers or Inverters? Often inverters result in lower insertion delay and power.**
- **Limiting the number of cells (e.g., 5 per type) can improve runtime.**
- **Use multi-cut vias to reduce electromigration.**
- **Often add padding near clock cells and flip flops to add decap.**

Running and Debugging CTS

```
reset_ccopt_config
source my_cts_spec.tcl
check_design -type cts
clock_opt_design
```

- **CTS is run with the `clock_opt_design` command*:**
  - CCOopt runs “Clock Concurrent Optimization”
  - In other words, it builds the clock tree and runs timing optimization.
- **Before CTS, run `check_design -type cts` to find possible problems.**
- **To run standalone CTS, use the `clock_design` command, which will build a balanced clock tree without optimizing timing.**
- **After CTS the clock is changed into “propagated” mode.**
- **In addition, average insertion delay is subtracted from the primary inputs.**
  - In other words, your source latency will now be negative!
  - To remove this, use: `set_db cts_update_clock_latency false`
- **After CTS, run `opt_design -post_cts -hold` to fix hold violations.**

* `clock_opt_design` replaced `ccopt_design`

Post CTS Reports

- **Some useful post CTS reports:**
  - `report_clock_trees`: Reports a summary of all defined clock trees.
  - `report_clock_tree_structure`: Reports the structure of the clock network.
  - `check`/`report_clock_tree_convergence`: Many paths leading to one output
  - `report_skew_groups`: Info about skew and insertion delays in skew groups.
  - `report_pin_insertion_delays`: Reports insertion delays at clock sinks.
  - `report_ccopt_worst_chain`: Reports clock chain with path with WNS.
  - `report_clock_tree_drv`: Reports DRVs in the clock network.
- **Of course, timing reports should be generated, including hold timing:**

```
report_timing
```

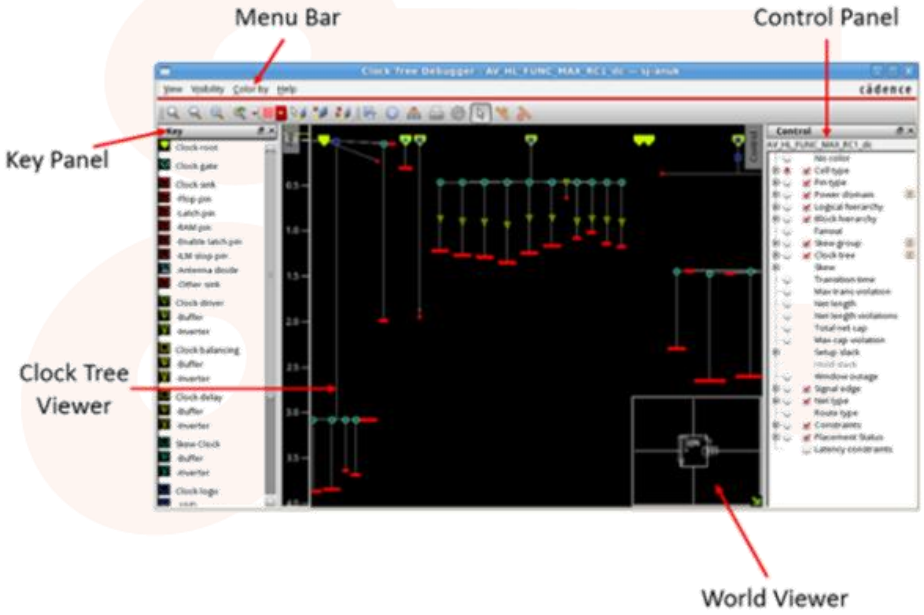
```
time_design -post_cts
```

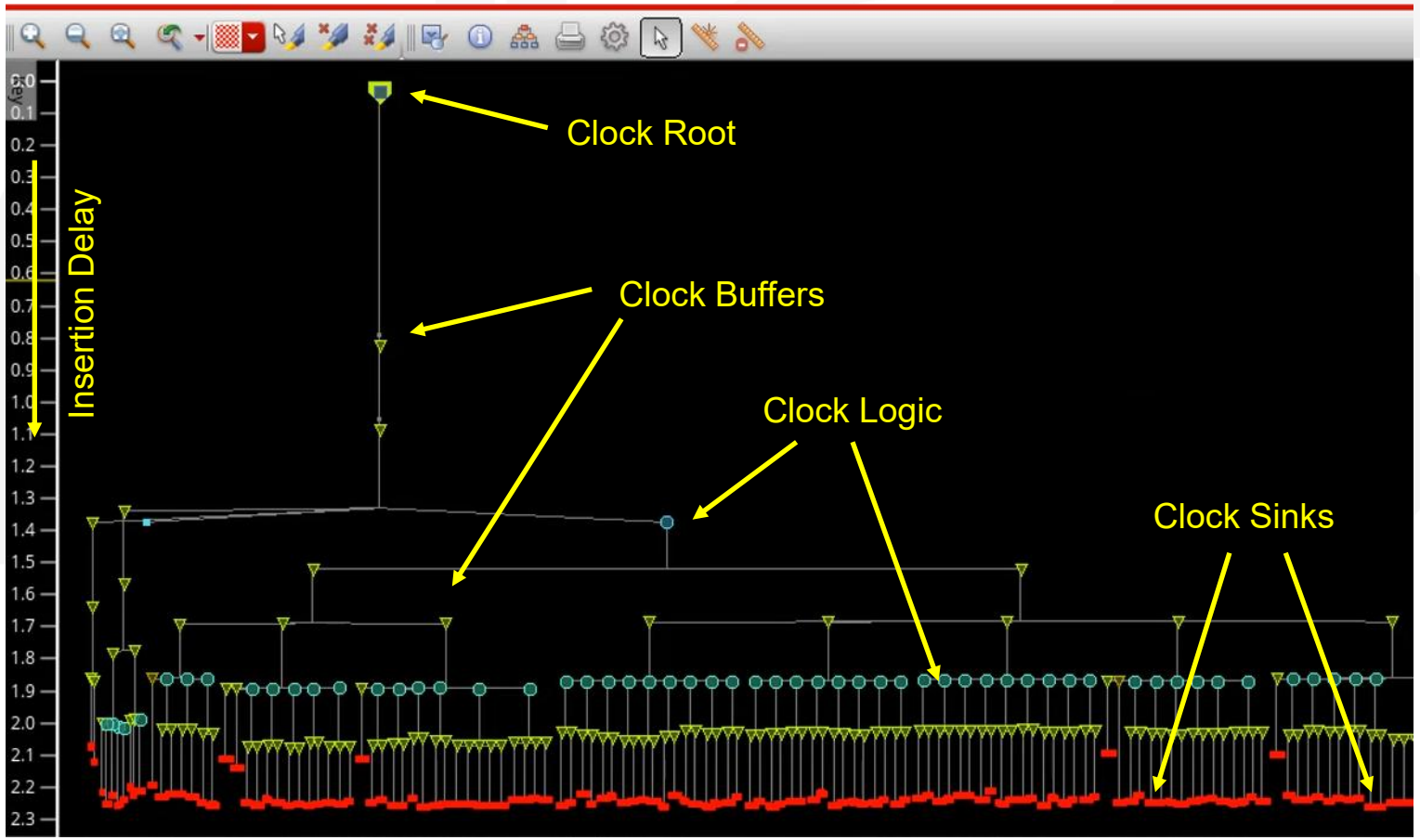
```
report_timing -early
```

```
time_design -post_cts -hold
```

Clock Tree Debugger

- Open with `gui_open_ctd` or Clock→CCOpt Clock Tree Debugger





RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.8: Routing and Timing Reports

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Routing

- **At this point:**
  - All cells have been placed.
  - Clock buffers have been inserted and clock tree has been routed.
  - Timing (setup and hold) is evaluated based on global route parasitics.
- **Concurrent routing and optimization is done with `route_opt_design`:**
- **The NanoRoute router features:**
  - Timing Driven
  - SI-Aware
  - Aware of manufacturing technology features and constraints (DRCs)

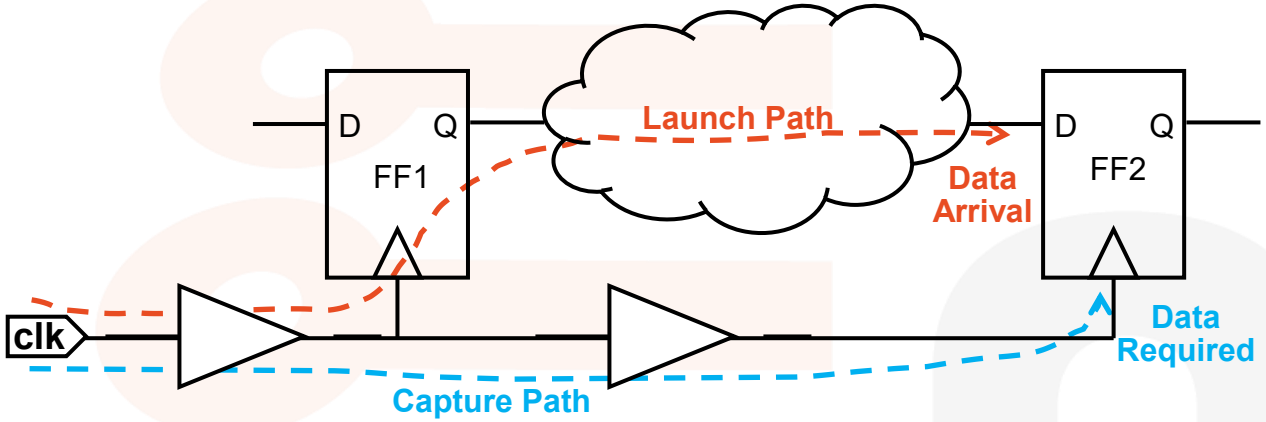
```
set_db route_with_timing_driven true
set_db route_with_si_driven true
route_opt_design
```

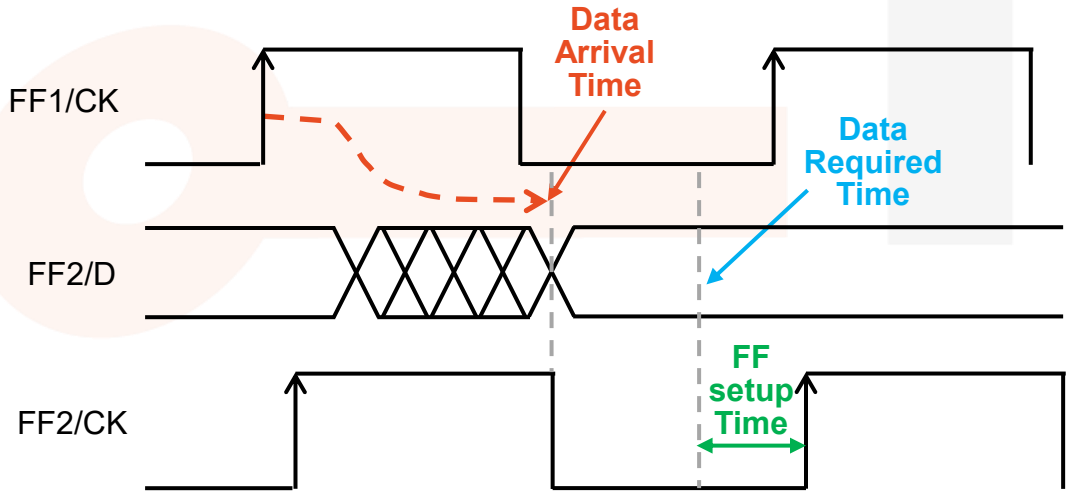
- **Routing is carried out in two steps:**
  - Global routing assigns nets to GCells, after which congestion can be analyzed.
  - Detailed routing lays down actual wires and connects to pins.
- **ECO routing is used for incremental fixes after detailed routing.**

Timing Reports

- **Perhaps the most important analysis after any stage is timing.**

`report_timing`





$$T + \delta_{skew} - 2\delta_{jitter} > t_{CQ} + t_{logic} + t_{setup} + \delta_{margin}$$

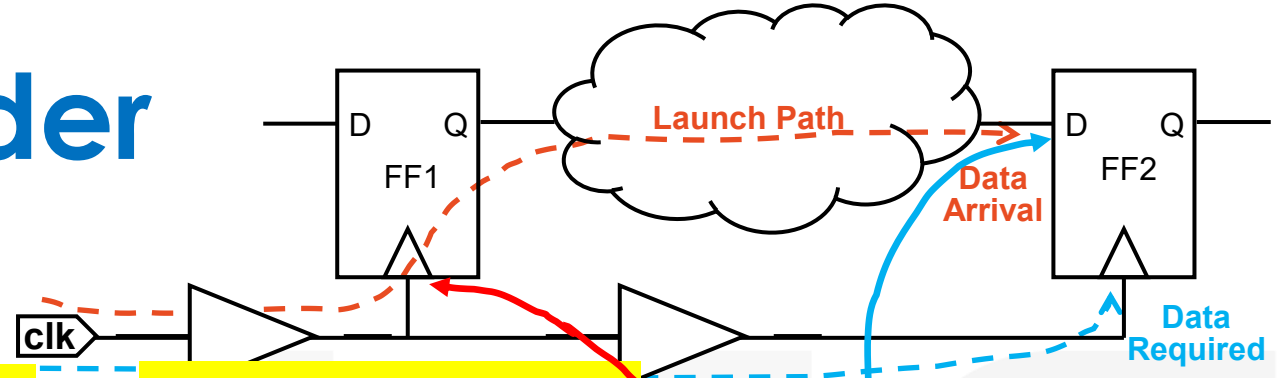
```
Path 1: VIOLATED (-0.003 ns) Setup Check with Pin id_stage_i/mult_dot_op_b_ex_o_reg[27]/CK->D
              View: wc_analysis_view
             Group: reg2reg
        Startpoint: (R) id_stage_i/mult_sel_subword_ex_o_reg/CK
             Clock: (R) clk
          Endpoint: (F) id_stage_i/mult_dot_op_b_ex_o_reg[27]/D
             Clock: (R) clk

                      Capture       Launch
       Clock Edge:+     6.000        0.000
      Src Latency:+    -0.916       -0.916
      Net Latency:+     0.937 (P)    0.935 (P)
          Arrival:=     6.020        0.019

            Setup:-     0.090
      Uncertainty:-     0.125
      Cppr Adjust:+     0.000
    Required Time:=     5.806
     Launch Clock:=     0.019
        Data Path:+     5.790
            Slack:=    -0.003
```

Header

Launch Path

Timing Point	Flags	Arc	Edge	Fanout	Trans (ns)	Delay (ns)	Arrival (ns)
id_stage_i/mult_sel_subword_ex_o_reg/CK		CK	R	13	0.100		0.019
id_stage_i/mult_sel_subword_ex_o_reg/Q		CK->Q	F	1	0.100	0.396	0.415
ex_stage_i_mult_i/FE_RC_1472_0/Y		B->Y	R	2	0.044	0.119	0.534
ex_stage_i_mult_i/FE_RC_418_0/Y		A1->Y	F	1	0.143	0.135	0.669
ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y		A->Y	F	21	0.101	0.171	0.840
⋮							
ex_stage_i_mult_i/sra_107_120_g4848/Y		A0->Y	F	1	0.139	0.106	4.950
ex_stage_i_mult_i/FE_RC_888_0/Y		A->Y	R	1	0.098	0.135	5.085
ex_stage_i_mult_i/FE_RC_887_0/Y		A->Y	F	1	0.161	0.115	5.199
g2031/Y		A0->Y	R	1	0.105	0.123	5.323
g1998/Y		A->Y	F	6	0.104	0.111	5.433
id_stage_i/FE_OCPC3458_regfile_alu_wdata_fw_27/Y		A->Y	F	1	0.115	0.117	5.551
id_stage_i/g34311/Y		A->Y	R	1	0.034	0.082	5.633
id_stage_i/FE_RC_546_0/Y		C->Y	F	3	0.119	0.176	5.809
id_stage_i/mult_dot_op_b_ex_o_reg[27]/D		D	F	3	0.187	0.002	5.809

Report Timing - Header



Path # - ordered by WNS

Did we meet timing?

Setup or Hold?

Path Group

Start Point

Endpoint

Clock Edges

Source Latency

Clock Net Latency

```
Path 1: VIOLATED (-0.003 ns) Setup Check with Pin id_stage_i/mult_dot_op_b_ex_o_reg[27]/CK->D
           View: wc_analysis_view
          Group: reg2reg
     Startpoint: (R) id_stage_i/mult_sel_subword_ex_o_reg/CK
          Clock: (R) clk
       Endpoint: (F) id_stage_i/mult_dot_op_b_ex_o_reg[27]/D
          Clock: (R) clk

                    Capture       Launch
     Clock Edge:+     6.000        0.000
     Src Latency:+   -0.916       -0.916
     Net Latency:+    0.937 (P)    0.935 (P)
         Arrival:=    6.020        0.019

           Setup:-    0.090
     Uncertainty:-    0.125
     Cppr Adjust:+    0.000
   Required Time:=    5.806
    Launch Clock:=    0.019
       Data Path:+    5.790
           Slack:=   -0.003
```

Rising or falling

Flop Setup Time

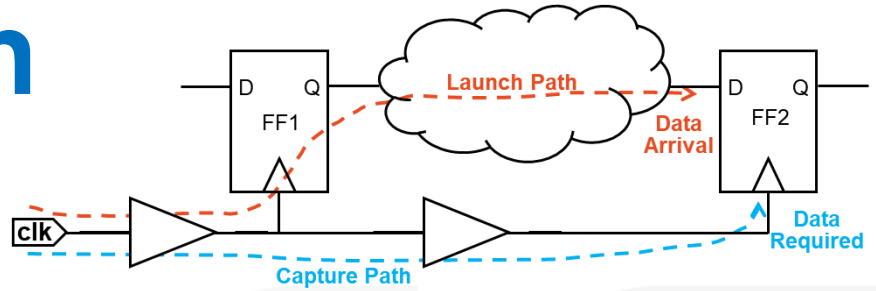
Clock Uncertainty (Jitter)

Required Time = Arrival – Setup – Jitter

Data Path arrival time

Final Slack Calculation

Report Timing – Launch Path



- **Standard timing report only shows the data delay of the launch path and very basic information.**

Timing Point	Flags	Arc	Edge	Fanout	Cap	Trans	Incr	Arrival (ns)
id_stage_i/mult_sel_subword_ex_o_reg/CK		CK	R	13	0.100			0.019
id_stage_i/mult_sel_subword_ex_o_reg/Q		CK->Q	F	1	0.100	0.396		0.415
ex_stage_i_mult_i/FE_RC_1472_0/Y		B->Y	R	2	0.044	0.119		0.534
ex_stage_i_mult_i/FE_RC_418_0/Y		A1->Y	F	1	0.143	0.135		0.669
ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y		A->Y	F	21	0.101	0.171		0.840
...								
ex_stage_i_mult_i/sra_107_120_g4848/Y		A0->Y	F	1				4.950
ex_stage_i_mult_i/FE_RC_888_0/Y		A->Y	R	1				5.085
ex_stage_i_mult_i/FE_RC_887_0/Y		A->Y				0.161	0.115	5.199
g2031/Y		A0->Y				0.105	0.123	5.323
g1998/Y		A->Y	F	6	0.104	0.111		5.433
id_stage_i/FE_OCPC3458_regfile_alu_wdata_fw_27/Y		A->Y	F	1	0.115	0.117		5.551
id_stage_i/g34311/Y		A->Y				0.034	0.082	5.633
id_stage_i/FE_RC_546_0/Y		C->Y				0.119	0.176	5.809
id_stage_i/mult_dot_op_b_ex_o_reg[27]/D		D	F	3	0.187	0.002		5.809

Launch clock insertion delay

Fanout

rising/falling

Instance name

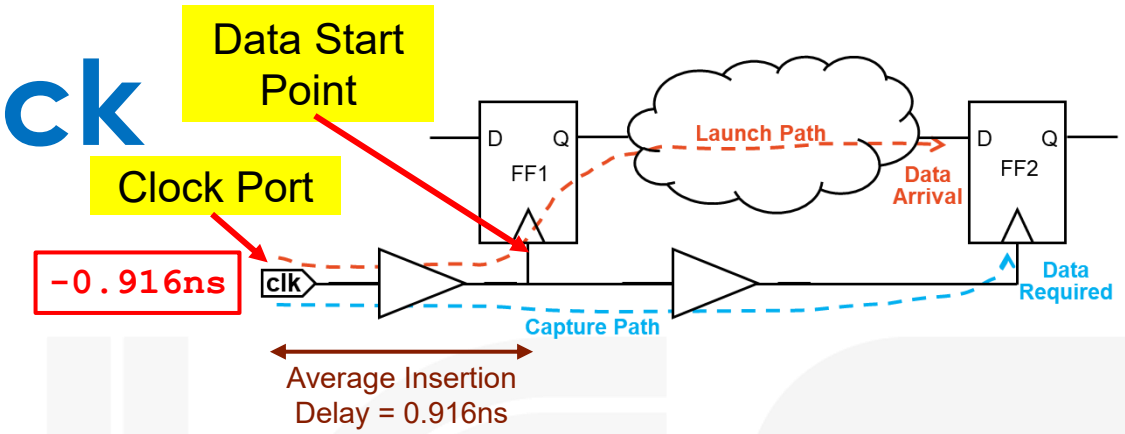
Timing Arc

Arrival Time

Transition

Gate + Wire Delay

Report Timing – Full Clock



- To get more data about the clock propagation, use the `full_clock` option:

```
report_timing \
 -path_type full_clock
```

- Pay attention – launch path now has many more details!

```
                  Capture      Launch
    Clock Edge:+    6.000       0.000
    Drv Adjust:+    0.079       0.079
   Src Latency:+   -0.916      -0.916
   Net Latency:+    0.857 (P)   0.856 (P)
       Arrival:=    6.020       0.019

         Setup:-    0.090
   Uncertainty:-    0.125
   Cppr Adjust:+    0.000
  Required Time:=   5.806
   Launch Clock:=   0.019
     Data Path:+    5.790
         Slack:=   -0.003
Timing Path:
+-------------------------------------------------------------------------------------------------------------------+
|                  Timing Point                  | Flags |   Arc    | Edge | Fanout | Trans | Delay | Arrival |
|                                                |       |          |      |        | (ns)  | (ns)  |  (ns)   |
+------------------------------------------------+-------+----------+------+--------+-------+-------+---------+
| clk_i                                          |       | clk_i    | R    |      3 | 0.163 | 0.018 |  -0.819 |
| g1033/Y                                        |       | B->Y     | R    |      3 | 0.166 | 0.173 |  -0.646 |
| CTS_ccl_a_BUF_clk_G1_L2_3/Y                    |       | A->Y     | R    |      4 | 0.071 | 0.160 |  -0.486 |
| id_stage_i/CTS_ccl_a_BUF_clk_G1_L3_10/Y        |       | A->Y     | R    |      5 | 0.096 | 0.163 |  -0.324 |
| id_stage_i/RC_CG_HIER_INST9/RC_CGIC_INST/ECK   |       | CK->ECK  | R    |      4 | 0.083 | 0.167 |  -0.157 |
| id_stage_i/CTS_ccl_a_BUF_clk_G2_L5_102/Y       |       | A->Y     | R    |     13 | 0.093 | 0.175 |   0.019 |
| id_stage_i/mult_sel_subword_ex_o_reg/Q         |       | CK->Q    | F    |      1 | 0.100 | 0.396 |   0.415 |
| ex_stage_i_mult_i/FE_RC_1472_0/Y               |       | B->Y     | R    |        |       |       |   0.534 |
| ex_stage_i_mult_i/FE_RC_418_0/Y                |       | A1->Y    | F    |        |       |       |   0.669 |
| ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y      |       | A->Y     | F    |     21 | 0.101 | 0.171 |   0.840 |
```

Source insertion delay is calculated to average out I/O clocking

Actual starting time is Src Latency+DRV Adjust+Delay

Clock Port

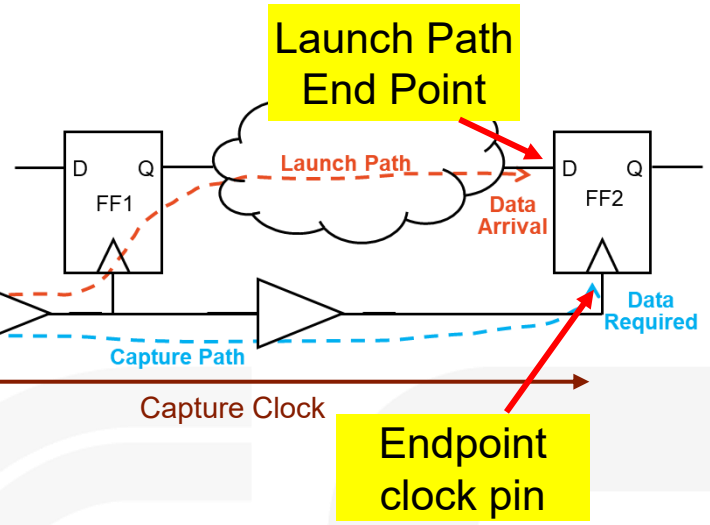
Launch Clock

Data Start Point

Timing report continues…

Report Timing – Full Clock (2)

- We also get to see the Capture Path.



Clock Period – Avg. Insertion Delay = 5.084ns

Continued from last slide

Launch path endpoint

Endpoint data pin

Actual starting time is Src Latency+DRV Adjust+Delay

Same Clock Port

Capture Clock

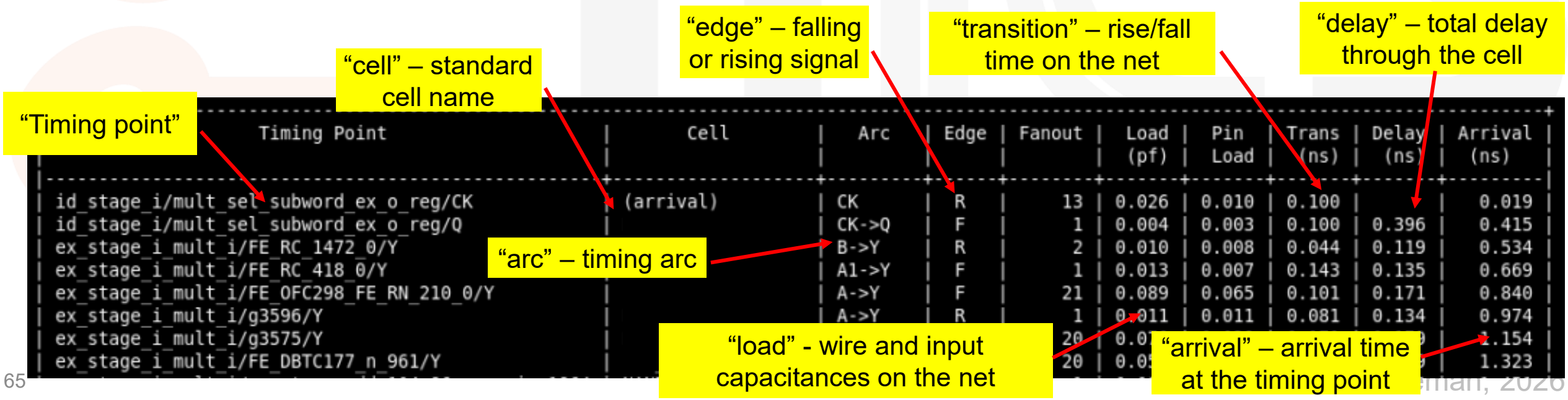
Endpoint clock pin

```
| ex_stage_i_mult_i/FE_RC_887_0/Y                                  |       | A->Y    | F |     1 | 0.161 | 0.115 |   5.199 |
| g2031/Y                                                          |       | A0->Y   | R |     1 | 0.105 | 0.123 |   5.323 |
| g1998/Y                                                          |       |         | F |     6 | 0.104 | 0.111 |   5.433 |
| id_stage_i/FE_OCPC3458_regfile_alu_wdata_fw_27/Y                 |       |         | F |     1 | 0.115 | 0.117 |   5.551 |
| id_stage_i/g34311/Y                                              |       |         | R |     1 | 0.034 | 0.082 |   5.633 |
| id_stage_i/FE_RC_546_0/Y                                         |       | C->Y    | F |     3 | 0.119 | 0.176 |   5.809 |
| id_stage_i/mult_dot_op_b_ex_o_reg[27]/D                          |       | D       | F |     3 | 0.187 | 0.002 |   5.809 |
+------------------------------------------------------------------------------------------------------------------------------+
Other End Path:
+------------------------------------------------------------------------------------------------------------------------------+
|                Timing Point                | Flags |   Arc   | Edge | Fanout | Trans | Delay | Arrival |
|                                            |       |         |      |        |  (ns) |  (ns) |  (ns)   |
|--------------------------------------------+-------+---------+------+--------+-------+-------+---------|
| clk_i                                      |       | clk_i   | R    |      3 | 0.163 | 0.018 |   5.181 |
| g1033/Y                                    |       |         | R    |      3 | 0.166 | 0.173 |   5.354 |
| CTS_ccl_a_BUF_clk_G1_L2_3/Y                |       |         | R    |      4 | 0.066 | 0.158 |   5.512 |
| id_stage_i/CTS_ccl_a_BUF_clk_G1_L3_10/Y    |       |         | R    |      5 | 0.096 | 0.163 |   5.674 |
| id_stage_i/RC_CG_HIER_INST8/RC_CGIC_INST/ECK |       | CK->ECK | R    |      5 | 0.083 | 0.183 |   5.858 |
| id_stage_i/CTS_ccl_a_BUF_clk_G2_L5_108/Y   |       | A->Y    | R    |     20 | 0.122 | 0.163 |   6.020 |
| id_stage_i/mult_dot_op_b_ex_o_reg[27]/CK   |       | CK      | R    |     20 | 0.071 | 0.000 |   6.020 |
+------------------------------------------------------------------------------------------------------------------------------+
```

Report Timing – `fields` option

- To debug timing, we would like more information, for example, the net name, the wire capacitance, the pin capacitance, etc.
- Use the `-fields` option to get the info you *really* need.
- For example:

```
report_timing -fields "timing_point cell arc edge fanout load pin_load transition delay arrival"
```



Timing Point	Cell	Arc	Edge	Fanout	Load (pf)	Pin Load	Trans (ns)	Delay (ns)	Arrival (ns)
id_stage_i/mult_sel_subword_ex_o_reg/CK	(arrival)	CK	R	13	0.026	0.010	0.100		0.019
id_stage_i/mult_sel_subword_ex_o_reg/Q		CK->Q	F	1	0.004	0.003	0.100	0.396	0.415
ex_stage_i_mult_i/FE_RC_1472_0/Y		B->Y	R	2	0.010	0.008	0.044	0.119	0.534
ex_stage_i_mult_i/FE_RC_418_0/Y		A1->Y	F	1	0.013	0.007	0.143	0.135	0.669
ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y		A->Y	F	21	0.089	0.065	0.101	0.171	0.840
ex_stage_i_mult_i/g3596/Y		A->Y	R	1	0.011	0.011	0.081	0.134	0.974
ex_stage_i_mult_i/g3575/Y				20	0.0				1.154
ex_stage_i_mult_i/FE_DBTC177_n_961/Y				20	0.0				1.323

Report Timing - Hold

- To report hold timing, just add the `-early` option

```
report_timing -early
```

```
Path 1: MET (0.001 ns) Hold Check with Pin id_stage_i/controller_i/jump_done_q_reg/CK->D
              View: bc_analysis_view
             Group: reg2reg
        Startpoint: (R) id_stage_i/controller_i/jump
             Clock: (R) clk
          Endpoint: (F) id_stage_i/controller_i/jump
             Clock: (R) clk

                      Capture       Launch
       Clock Edge:+     0.000        0.000
      Src Latency:+    -0.240       -0.240
      Net Latency:+     0.210 (P)    0.210 (P)
          Arrival:=    -0.030       -0.030

             Hold:+     0.010
      Uncertainty:+     0.125
      Cppr Adjust:-     0.000
    Required Time:=     0.105
     Launch Clock:=    -0.030
        Data Path:+     0.137
            Slack:=     0.001
```

Timing Point	Arc	Edge	Fanout	Load (pf)	Pin Load	Trans (ns)	Delay (ns)	Arrival (ns)
id_stage_i/controller_i/jump_done_q_reg/CK	CK	R	16	0.053	0.016	0.051		-0.030
id_stage_i/controller_i/jump_done_q_reg/Q	CK->Q	F	2	0.006	0.003	0.051	0.094	0.063
id_stage_i/controller_i/g4794/Y	B->Y	R	1	0.002	0.002	0.014	0.022	0.085
id_stage_i/controller_i/g4788/Y	B->Y	F	1	0.004	0.002	0.022	0.022	0.106
id_stage_i/controller_i/jump_done_q_reg/D	D	F	1	0.004		0.023	0.000	0.106

Now it's a hold check!

The analysis view changed to the Best Case corner

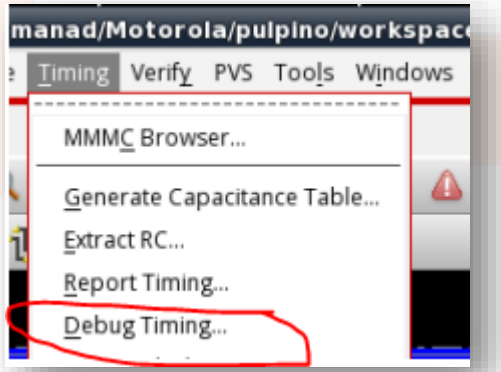
Launch and capture clock at the same edge

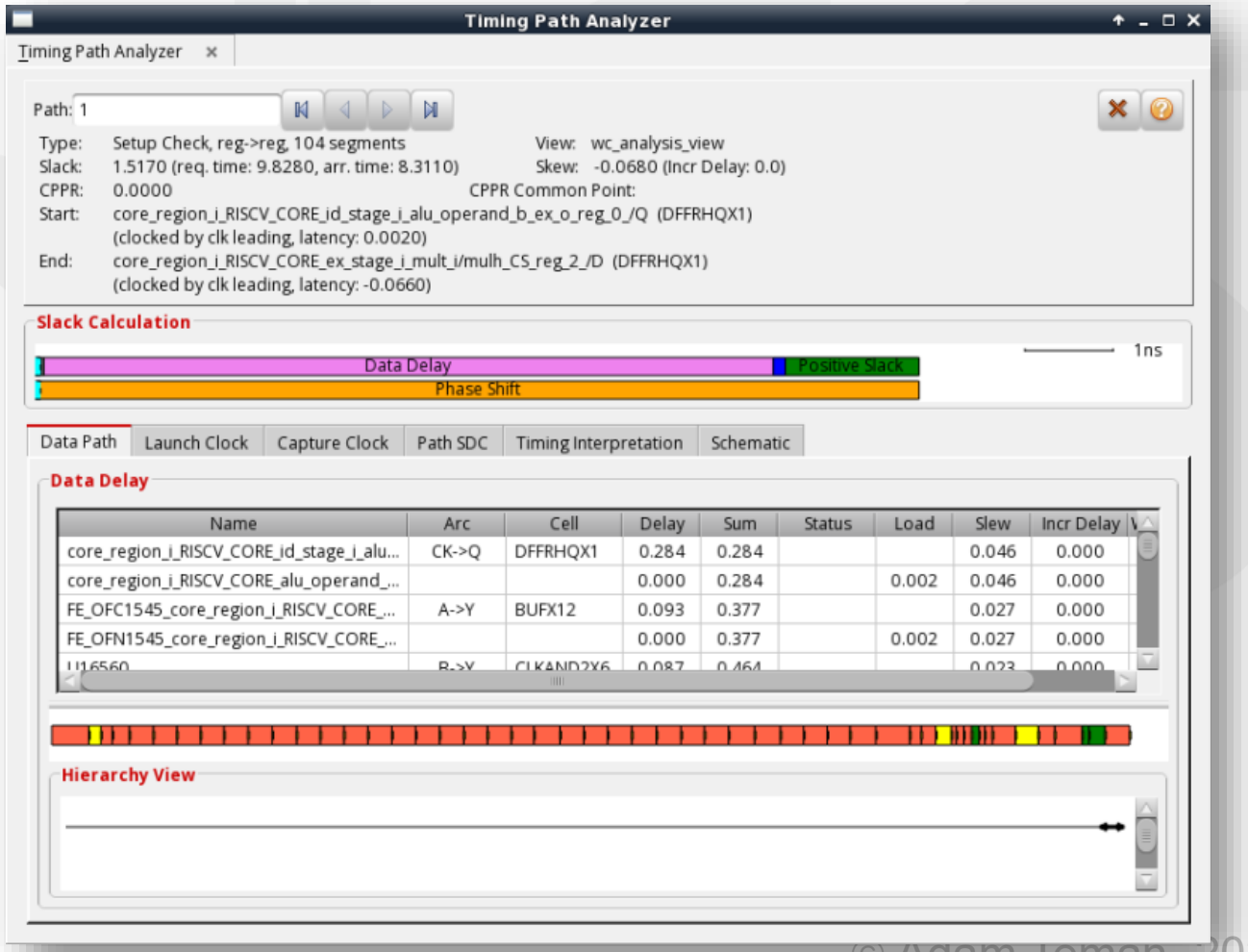
Register hold constraint

Now, it's Slack=Arrival – Required

Report Timing Debugger

- **A very good GUI option is to use the Innovus “Debug Timing” tool.**
  - This tool lets you explore the timing report interactively, even showing path schematics, SDC, and highlighting the path in the layout.





Post Route Reporting and Checks

- **The primary report for routing analysis is `report_route`:**
  - Congestion, Track utilization and DRCs
  - Wire length and Wire density
  - Single-cut/Multi-cut vias
  - EM coverage
- **Additional reports include `report_wires` and `report_congested_area`.**
- **At this point, you can run physical verification checks:**
  - `check_drc`: Runs a DRC check (based on the techlef)
  - `check_connectivity`: Runs an Innovus internal LVS
  - `check_antenna`: Verifies that there are no antenna violations.
  - `check_metal_density`: Checks for density violations.

RTL2GDS Demo Part 5:

Simple SoC Example Full RTL2GDS Flow

Section 5.9: Post Route and SignOff

Prof. Adam Teman

EnICS Labs, Bar-Ilan University

www.enicslabs.com

Emerging Nanoscaled
Integrated Circuits and Systems Labs

The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Post Route Optimization and Signoff

- **At this point we have a fully routed design and we are ready to finish the physical design stage of our design.**
- **In this final section of our small SoC demonstration, we will briefly go over:**
  - Post route optimizations, including design for manufacturing (DFM)
  - DRC fixing within Innovus
  - Applying ECOs within Innovus
  - Finishing up the design (fillers, metal fill, and design cleanup)
  - Signoff timing
  - Exporting the design

Post-Route Optimizations

- **Post-route timing optimization is run with:**

```
opt_design -post_route -setup -hold
```

- **Several options can be set for specific optimizations:**
  - Fixing SI slew violations:
    - (note that SI glitch violations are fixed by default)

```
set_db opt_post_route_fix_si_transitions true
```

  - Fixing Signal EM violations:
    - Wire Widening
    - Driver Downsizing
    - Buffering

```
fix_ac_limit_violations \
      -allow_down_size true \
      -allow_add_buffer true
```

  - Optimizing power:
    - Ratio 0.0 for dynamic power optimization
    - Ratio 1.0 for leakage power optimization

```
set_db design_power_effort <none|low|high>
set_db opt_leakage_to_dynamic_ratio <0.0 - 1.0>
```

  - Post-route useful skew:

```
set_db opt_skew_post_route true
```

Post-Route DFM Optimization

- **At this point we can run some optimizations for improving yield:**
  - Wire optimization (straightening, widening, spreading)

```
set_db route_design_detail_post_route_spread_wire true
route_design -wire_opt
```

```
set_db route_design_detail_post_route_wire_widen_rule <ruleName>
set_db route_design_detail_post_route_wire_widen widen
route_design -wire_opt
```

  - Via optimization (via reduction, multi-cut)

```
set_db route_concurrent_minimize_via_count_effort high
route_design -via_opt
```

```
set_db route_design_reserve_space_for_multi_cut true
route_design
set_db route_design_detail_post_route_swap_via true
set_db route_design_with_timing_driven false
route_design -via_opt
```

  - Lithography-aware routing

```
set_db route_design_detail_post_route_litho_repair true
route_design
```

Post Route DRC Fixing

- After routing, there are often DRC violations.

```
check_drc -limit 1000000000
```

- To fix DRCs, use ECO routing:

```
set_db route_with_eco true
route_design
```

```
route_eco -fix_drc
```

- If that doesn't work, try deleting and re-routing the nets with violations

```
check_drc -limit 1000000000
delete_routes -regular_wire_with_drc
route_design
```

- To save runtime try routing only selected nets:

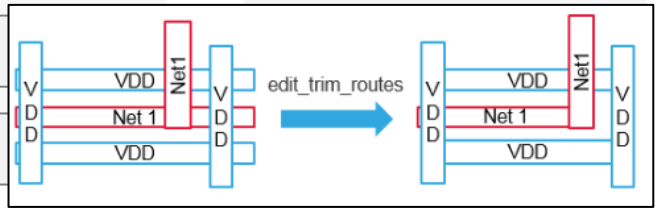
```
set_db route_selected_net_only true
route_design
```

- Delete all gap, notch and hole geometries

```
delete_notch_fill
```

- Remove or trim dangling wires.

```
edit_trim_routes
```



- If you have DRC violations that you only see in your signoff tool (e.g., PVS/Calibre):

```
read_markers <file> -rule_map_file <rule_file> -type <pvs/Calibre/..>
route_fix_signoff_drc ;# Optional -fix_rules or -exclude_rules
```

Applying ECOs

- **ECOs (Engineering Change Order) are small modifications made to the design at a very late stage. These usually are applied in two cases:**
  1. After routing on a large design to save time of rerunning the whole flow.
  2. To implement a metal fix after tape out (possibly before backend fabrication)
- **ECOs can be implemented automatically (e.g., through Tempus ECO flow) or manually:**
  - Start by turning off automatic ECO:

```
set_db eco_refine_place false
set_db eco_update_timing false
```

  - Manually add buffer:

```
eco_add_repeater -cells <libcell> -net <net> \
                 -location {<xcoord> <ycoord>}
```

  - Manually upsize/downsize cell:

```
eco_update_cell -insts <inst name> \
                -up_size/-down_size
```

  - Manually change the cell:

```
eco_update_cell -insts <inst name> -cell <libcell>
```

  - Then run placement legalization:

```
place_detail
```

Finishing the Design

- **At this point, we need to finish the design and clean things up:**
  - Add fillers, wherever there are no standard cells placed:

```
add_fillers
```

  - Alternatively, add decap cells:

```
add_decap_cell_candidates DECAP10 10
add_decaps -total_cap 1000 -cells DECAP10 DECAP8
```

  - Add and check metal fill within Innovus:

```
add_metal_fill -net vdd
check_metal_density
```

  - Fix timing violations due to metal fill:

```
trim_metal_fill_near_net -slack_threshold XXX \
      -min_trim_density XXX -spacing XXX \
      -spacing_above XXX -spacing_below XXX
```

  - Clean up netlist:

```
delete_assigns
delete_empty_hinsts
delete_dangling_ports
delete_floating_constants
```

Signoff Timing

- **For signoff, use the opt_signoff and time_design_signoff commands:**

```
opt_signoff -drv/-setup/-hold
```

```
time_design_signoff
```

  - `time_design_signoff` calls the Tempus signoff timing tool.
- **For signoff timing, advanced timing analysis modes should be used.**
- **This is a bit complex and beyond the scope of this tutorial but it includes**
  - On-chip Variation (OCV) with Clock Path Pessimism Removal (CPPR)
  - Advanced OCV (AOCV)
  - Liberty Variation Format (LVF) files and Statistical OCV (SOCV)
  - Need to define AOCV/SOCV in MMMC

```
set_timing_derate -late 1 -early 0.9 -clock
set_db timing_analysis_type ocv
set_db timing_analysis_cppr both
```

```
set_db timing_analysis_aocv true
```

```
set_db timing_analysis_socv true
set_timing_derate -sigma/mean \
   -cell_delay/net_delay 1.2 [get_lib_cells */*]
```

```
create_library_set -name XXX \
   -timing <LVF .lib files> -aocv/socv <aocv/socv side files>
```

Exporting the Design

- **Once everything is finished within Innovus, it is time to export the design:**
  - **`write_netlist`**: Exports Verilog netlist (`.v`) of the design.
  - **`write_def`**: Export DEF file including floorplan, placement, routing, etc.
  - **`write_lef_abstract`**: Generates abstract (`LEF`) of the current block.
  - **`write_timing_model`**: Builds a Liberty (`.lib`) format model for the top cell.
  - **`write_stream`**: Exports the layout to a `GDSII` Stream file.
  - **`write_db`**: Export design database in the native Innovus `.db` format
  - **`write_sdf`**: Exports timing delays to a Standard Delay Format (`.SDF`) file.
  - **`write_design`**: Exports the design for loading in Tempus.
  - **`write_do_lec`**: Creates `.dofile` for Conformal LEC.

Summary

- **In this demo, we saw how a simple, albeit full SoC is implemented from RTL to GDS:**
  - We started by introducing the SoC Architecture and showing that the provided RTL can run a full high-level language (C) compilation toolchain.
  - We explored the SoC, including instantiated hard macros, such as SRAMs and I/Os.
  - We ran full-chip synthesis with Genus.
  - We moved the design over to Innovus for physical implementation.
  - We created a full-chip floorplan, including macro placement and I/O ring.
  - We ran standard cell placement.
  - We synthesized the clock tree.
  - We routed the design and fixed DRCs.
  - We examined the timing reports.
  - We finished the design and exported our final files.
- **This was a simple example, run in a mature process node, but it included the majority of what goes into a large design in an advanced process.**