Showcasing Efficient Low-Power Techniques Using RISC-V CPU Cores with DC-NXT And ICC2

Using GF's 22FDX-PLUS Technology

Farid Labib, SMTS Application Engineering
GlobalFoundries

Agenda

- Introduction
  - Motivation, Test Vehicle, Tools & Scripts
- Focus On
  - Area Reduction (ICC2)
  - Low Power Implementation Techniques (DC-NXT, ICC2)
- Verification
- Tips & Tricks
- Issues & Solutions
- Conclusion / Outlook

Introduction

Motivation, Design & Technology, Tools & Scripts

Motivation

Efficiency in Low Power Implementation is key towards the targeted PPA

- Technology, and optimized IP can contribute significantly → But …
  - Special std-cell architecture and multi-voltage components, require special handling
  - Tools need to be guided to get most out of the deliveries

- Thus, providing GF Customers with a demonstrator how to unfold the potential of a technology and respective IP, is key
  - ✓ Script set tuned to play well with the respective technology and IP
  - ✓ Evaluated in collaboration with the EDA and IP Vendor
  - ✓ Reduces trial & error cycles

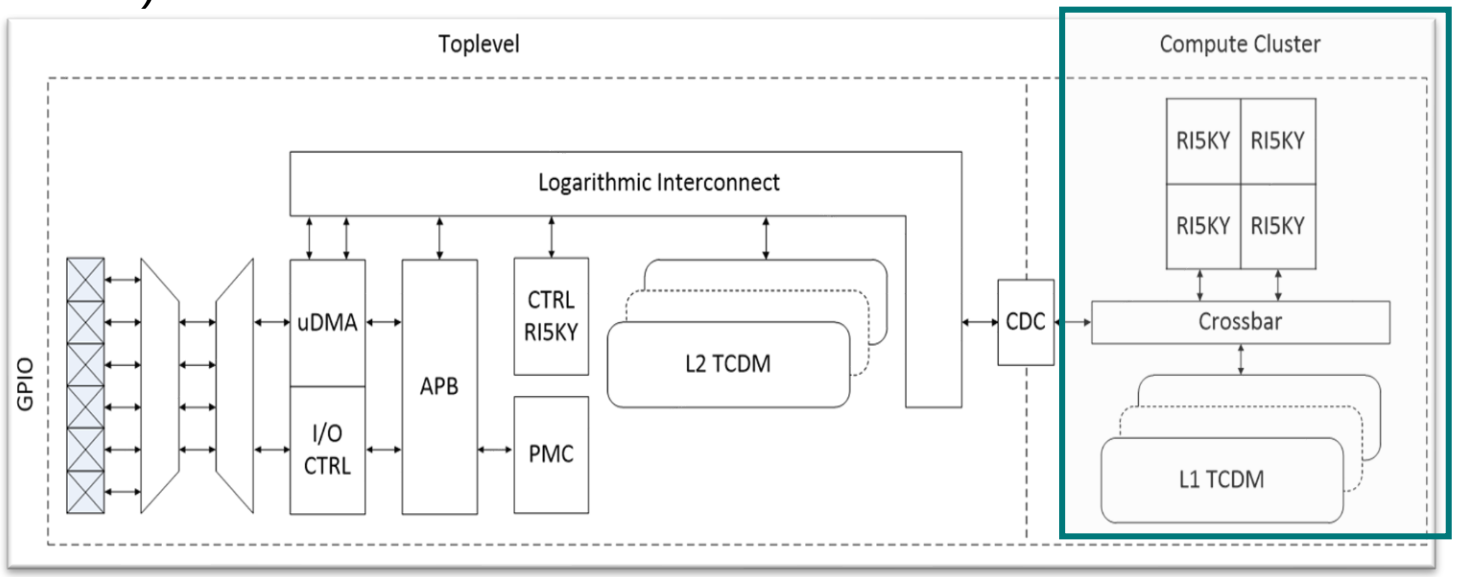
- The motivation for presenting at SNUG
  - ✓ Sharing *generally applicable* best practices, tips & tricks, with the broader user community
  - ✓ Sharing issues and workarounds

Design & Technology

Need of a complex enough, and sharable, design example

- "PULP KRAKEN" SoC by ETH Zurich → focus on "Compute Cluster"
- 4 RISC-V CPU cores → Ideal for splitting into switchable domains
  - ✓ Shared Bank of Tightly Coupled Memories (TCM) and related control logic → Real design with memories
  - ✓ Open-source → Can be shared with our Customers
  - ✓ Not too complex → Acceptable runtime for a demonstrator

- 22FDX-PLUS, 22nm fully depleted SOI (FDSOI)
- Stack: 6 routing layers, one thick layer
- Cell Libraries:
  - Ultra-dense / ultra-low power cell architecture
  - Multi-height cells, such as multi-bit flip-flops
  - Retention flip-flops, switch cells, isolation cells, always-on buffers
  - Dual-rail memory
  - Adaptive Body Bias (ABB)



Tools, Scripts

Tools

- DC-NXT (2022.12-SP7)
- ICC2 (2022.12-SP6-T-20231012)
- Redhawk In-Design (2022 R2.3)
- Formality (2022.03)

- PEX, STA, ICV (not subject of this presentation)

RM-Scripts

- DC-NXT: U-2022.12-SP4
- ICC2: U-2022.12-SP6

Area Reduction in ICC2

mixed preferred routing direction, density control

Area Reduction in ICC2

#1 mixed preferred routing directions

Different IP may come with different preferred routing directions (layout & pin shapes)

- Area optimized std-cells → HVH *)
- Memory macros → VHV **)

What to consider:

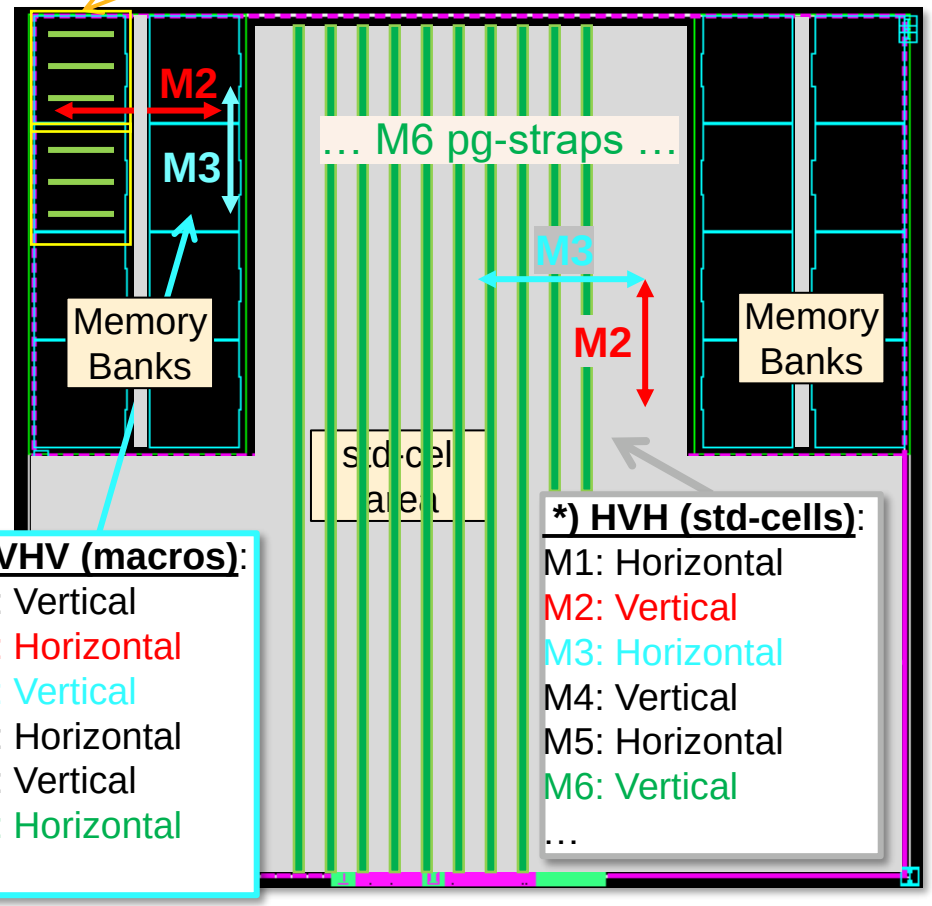
- ✓ Power mesh in the different areas (“HVH” and “VHV”)
- ✓ Power mesh alignment in the transition from “HVH” to “VHV”
- ✓ Signal routing over the macros (avoid potential fringe caps)

Alternatives:

- ✓ Compilers with option to flip preferred routing direction scheme, and block certain layers to avoid fringe cap issues

```
create_routing_guide \
  -switch_preferred_direction \
  -layers {…} -river_routing \
  -boundary {<Macro-Boundary>}
```

Floorplan (Simplified)



More details on pg-examples in Paper AND SNUG 2022 Presentation

Area Reduction in ICC2

#2 density control – key contributors

Area optimized std-cells can come with dense and special pin layout

✓Routing track alignment for optimal pin access

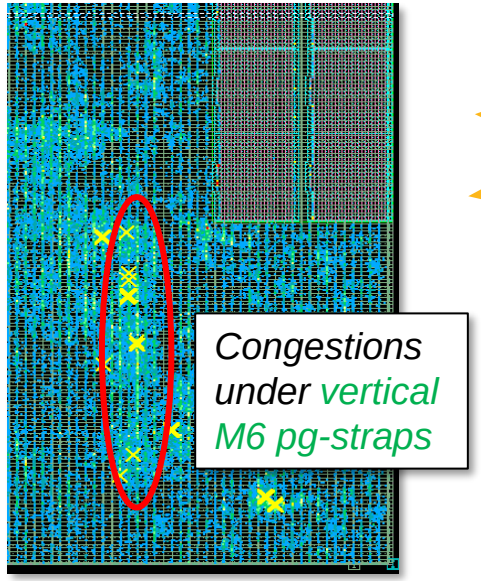
→ To align lowest metal tracks with std-cell pin layout

➢ Command **`set_wire_track_pattern`**

✓Controlling of cell density under power mesh

→ To avoid congestions under pg-structures

➢ App options
`place.common.pnet_aware_density/layers`



Details to the actual settings in the paper

✓Controlling overall utilization for optimal cell density w/o running into congestions

→ Look out for PLACE-027 messages place_opt.log file

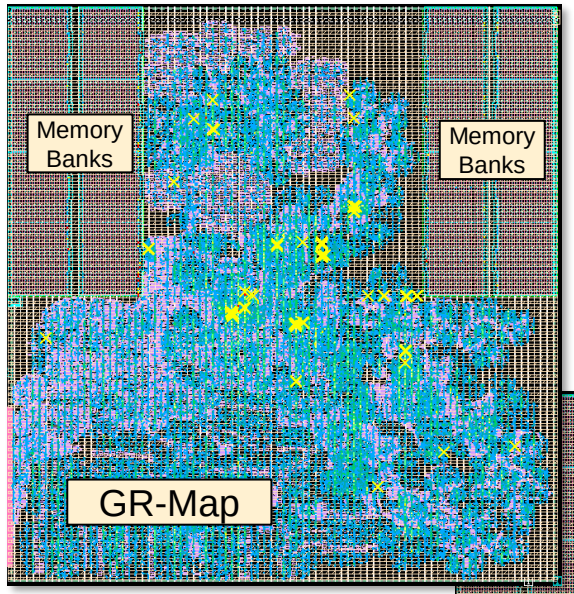
→ Use defaults to start with

➢ App options
`place.coarse.auto_density_control/max_density`

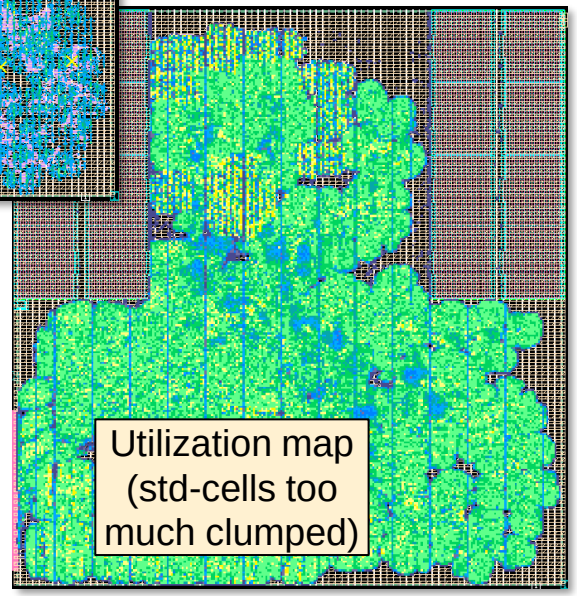
```
Information: Automatic density control has
selected the following settings: max_density
0.48, congestion_driven_max_util 0.87.
(PLACE-027)

…

Information: Automatic density control has
selected the following settings: max_density
0.48, congestion_driven_max_util 0.89.
(PLACE-027)
```

Area Reduction in ICC2 – Results

- ✓ Efficient use of the std-cell area by utilizing the right app options *)
- ✓ Efficient routing over the macros by utilizing routing guides and pg-scheme alignment *)

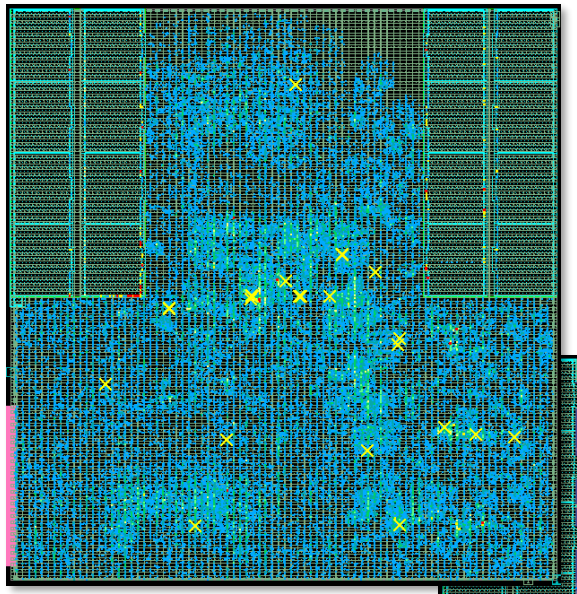


~300 DRC violations

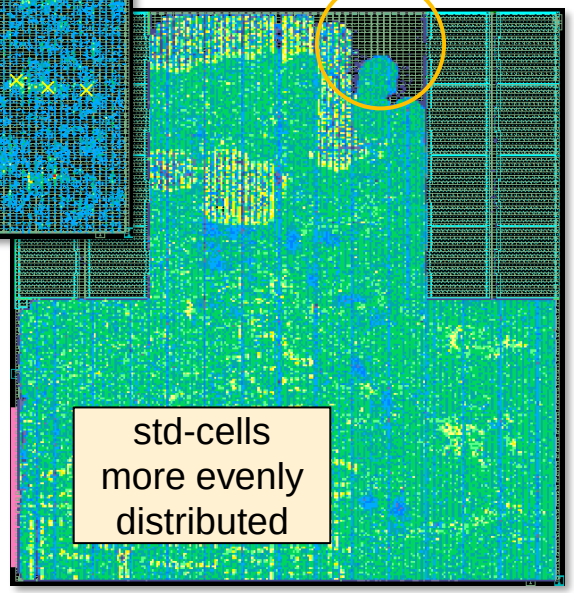


After "applying Slides 8,9"

**)



- ✓ <30 DRC violations
- ✓ ~7% additional area reduction



*) more info on experiments in the paper

**) *Mainly manually; automation not scope of the exercise (→ outlook)*

Low Power Implementation Techniques DC-NXT and ICC2

partitioning, switch cells, retention, mesh alignment…

Low Power Implementation Techniques

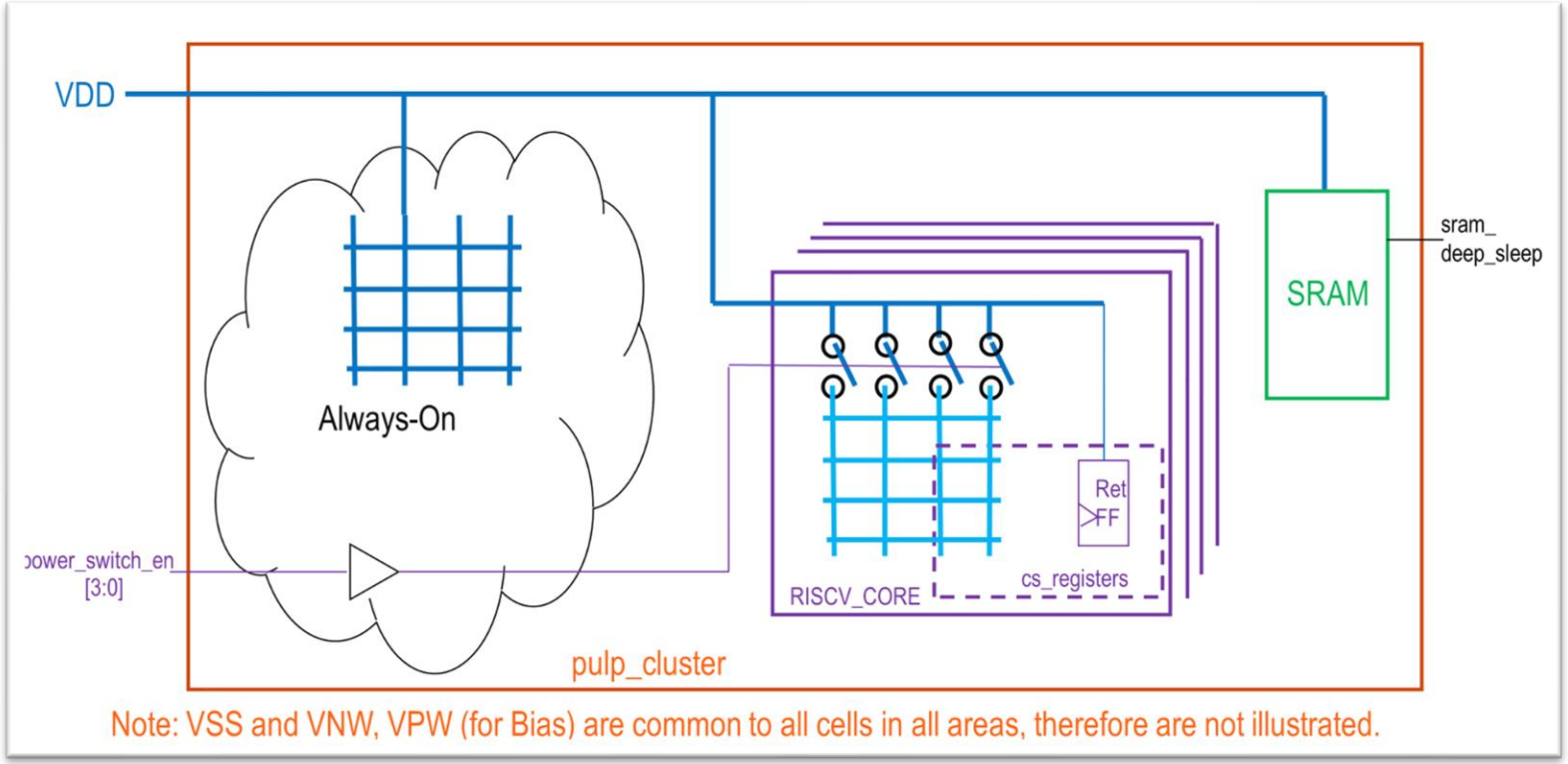
To Demonstrate Multi-Voltage Usage and Energy Efficiency

- ✓ Switchable RISC-V Cores (domain separation, power switches and isolation)
- ✓ Retention of control logic (energy efficiency)
- ✓ Two backup power schemes
- ✓ Adaptive Body Bias (dealing with bias routes)
- ✓ VSS, VNW and VPW is shared
- ✓ Utilizing power modes of SRAMs

i

There is always a tradeoff to be made, overhead of the applied techniques versus power savings.

Complete UPF code in paper

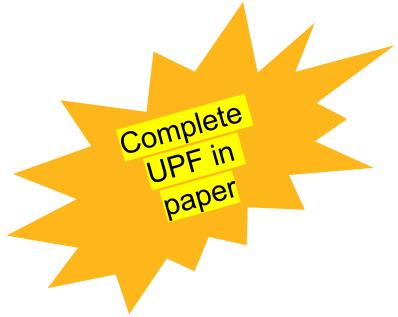


Low Power Implementation Techniques

Considerations for Synthesis w/ DC-NXT

- Power intent fully coded in UPF 2.1 (prime flow used), and Body-Bias enabled

```
set_design_attributes -elements {.} -attribute enable_bias true
…
create_supply_set ss_vdd \
  -function {nwell NET_BIAS_VNW_N} -function {pwell NET_BIAS_VPW_W} –update
```



- For proper DFT insertion ensure that retention enable pins (Save/Restore), are set to a mode, where scan chains are traceable

```
set_dft_signal -view existing_dft -type Constant \
  -port [list {ret_en0 ret_en1 ret_en2 ret_en3}] -active_state 1
```

- For proper always-on-buffering be careful with “dont_use” list specifications;
  → Usage of Always-On buffers also require always-on inverters to be present in the use-list

Low Power Implementation Techniques

Creating Switchable Power Domains, Floorplaning Voltage Areas

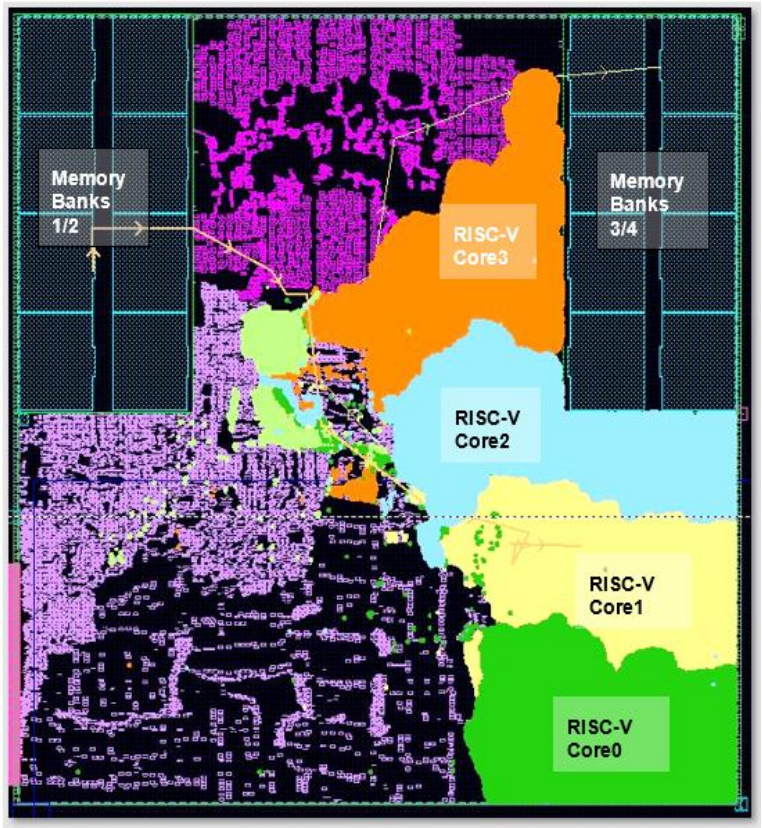
```
create_power_domain pd_top

create_supply_set ss_vdd

associate_supply_set ss_vdd \
-handle pd_top.primary
```

```
create_power_domain pd_core2 \
-elements CORE[2].core_region_i

create_supply_set ss_vdd2 \
-function {ground ss_vdd.ground} \
-function {nwell ss_vdd.nwell} \
-function {pwell ss_vdd.pwell}

associate_supply_set ss_vdd2 \
-handle pd_core2.primary
```

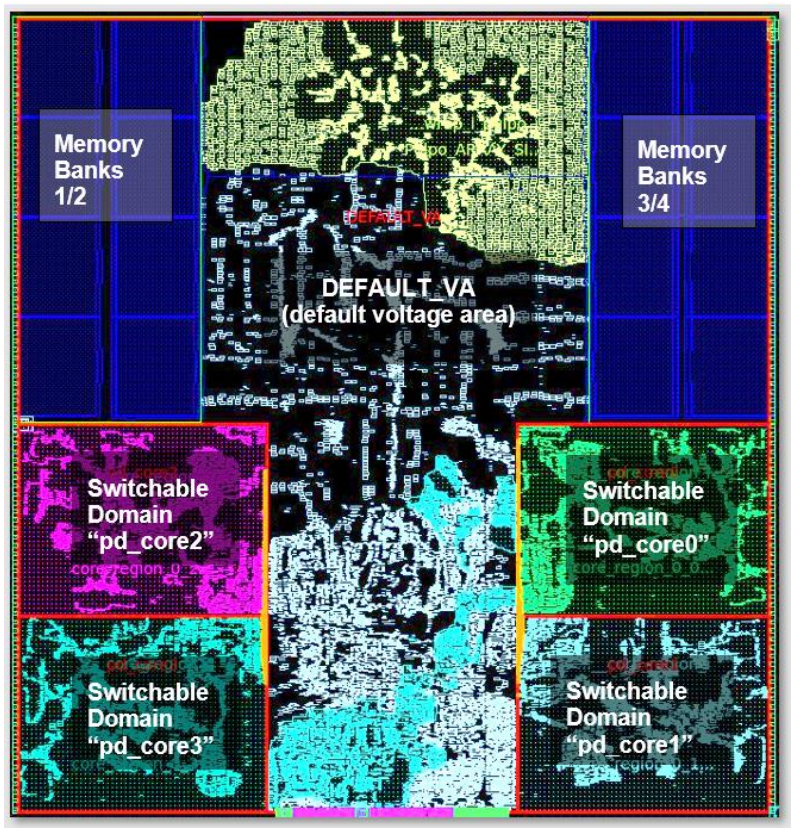
Ground and body-bias is shared between all domains

- ✓ Analyzing logic distribution
- ✓ Analyzing critical paths

Partitioning analysis trials *)



Final floorplan with Voltage Areas



*) Mainly manually; automation not scope of this exercise (→ outlook)

Low Power Implementation Techniques

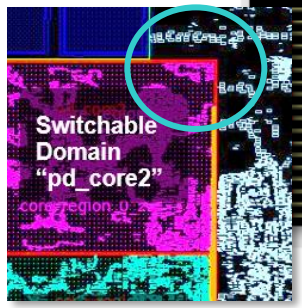
Transition from Always-On to Switchable Domains

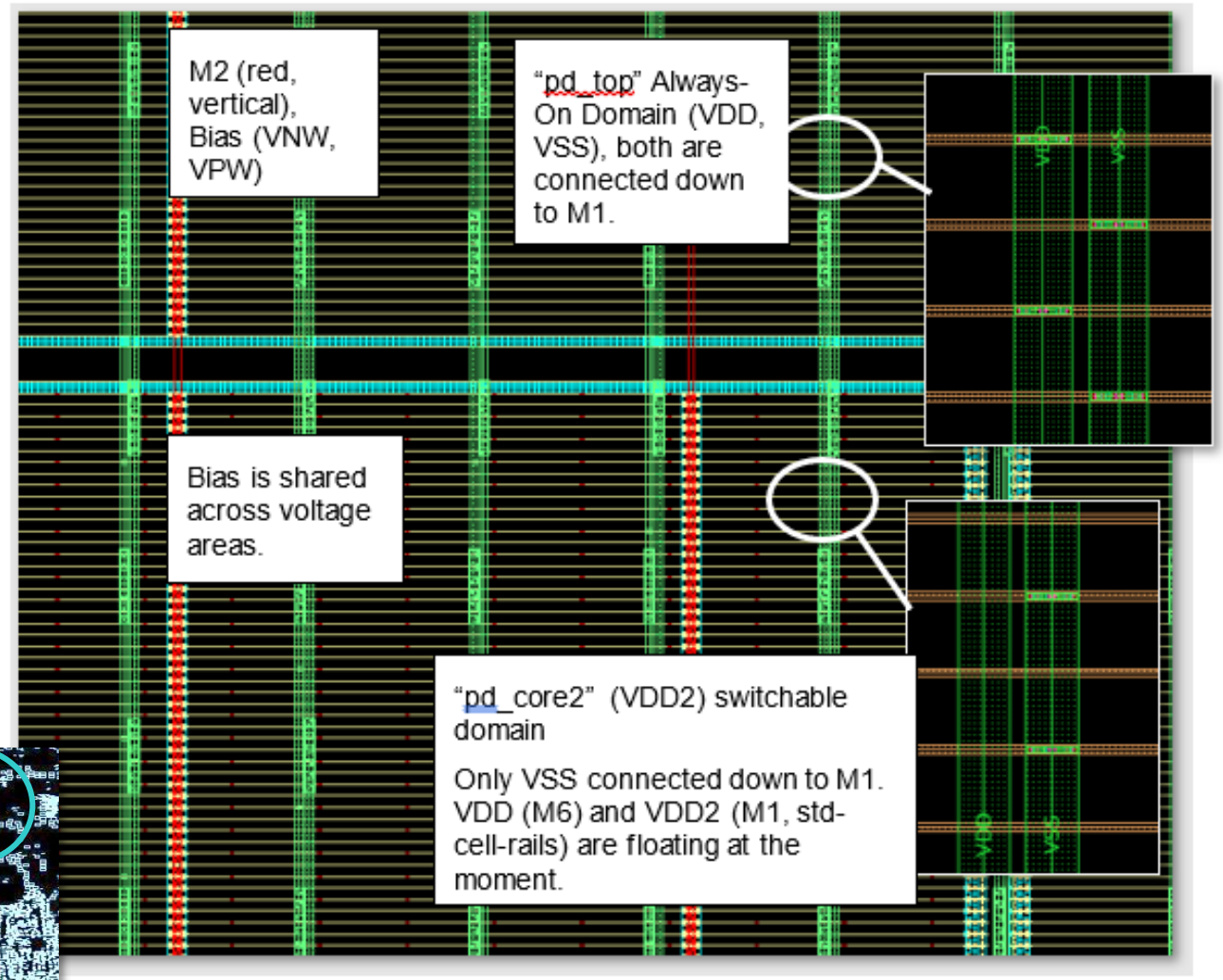
```
create_power_switch SW2 \
-domain               pd_core1 \
-supply_set           ss_vdd2 \
-input_supply_port    {VDDP ss_vdd.power} \
-output_supply_port "VDDC ss_vdd2.power" \
-control_port         "EN pwr_off2" \
-on_state             {P_on VDDP {!EN}} \
-off_state            {P_off {EN}
```

```
compile_boundary_cells -voltage_area …

create_tap_cells -voltage_area …
```

- Separate Voltage Areas
- Separate Boundary and Tap Cells
- Always-On M6 VDD and VSS down to M1
- Switchable: M6 VDD “floating”, only VSS down to M1

Note: Body-Bias supply nets are shared





Low Power Implementation Techniques

Insert Power Switch Arrays

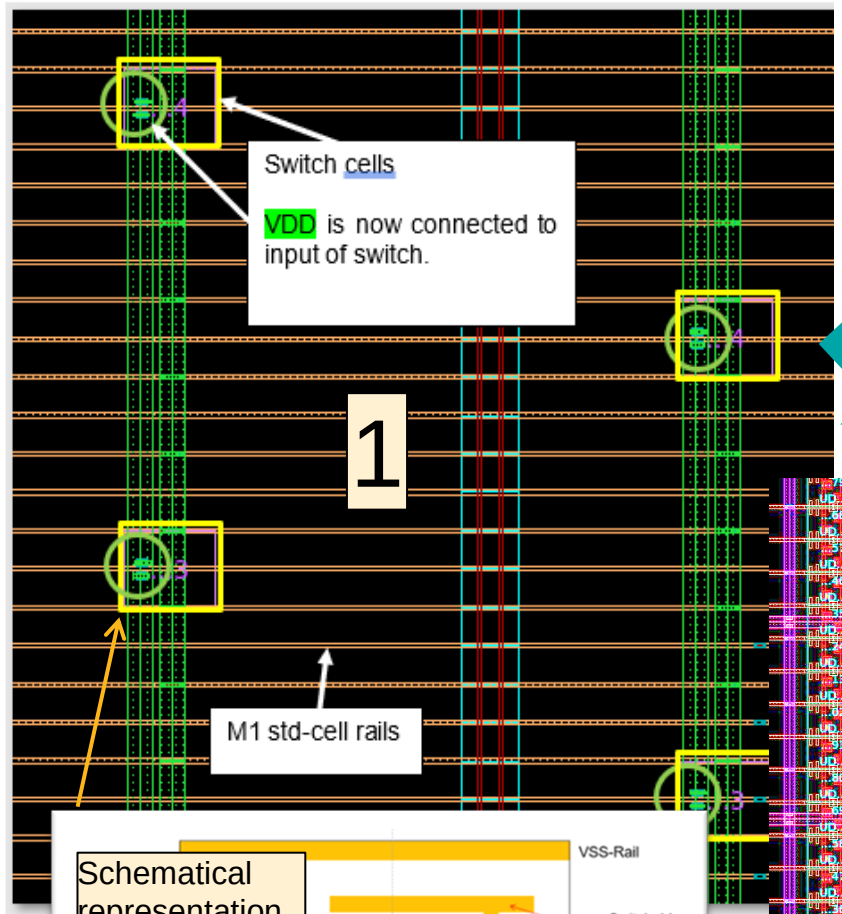
1 Alignment of switch cells with primary mesh (always-on)

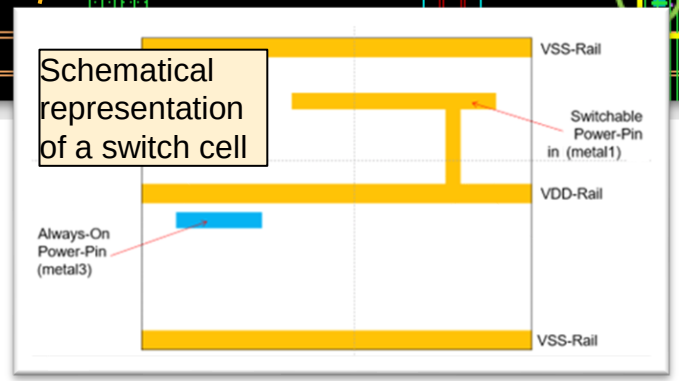
```
create_power_switch_array -
lib_cell … \
 -pg_straps [get_shapes \
 -filter "tag==C4_mesh_VDD"]
…
```

i *Use “tags” to identify pg-objects (for various purposes)*

2 Alignment of secondary mesh with switch outputs (calculating based position of switch cells)

i *Alignment of pg-objects by using multiple of tile width and height*



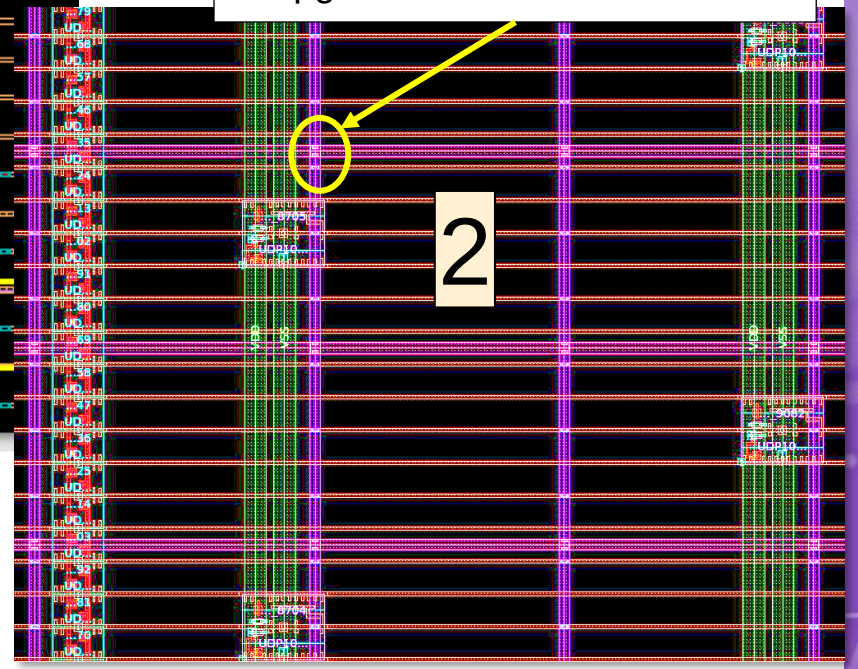


Secondary mesh (VDD2)

M4 (vertical, purple)
M5 (horizontal, pink)

Alignment

- vertical M4 with switch output pin
- Horizontal M5 to between M1 pg-rails



Low Power Implementation Techniques

Retention Flip-Flop Always-On Power

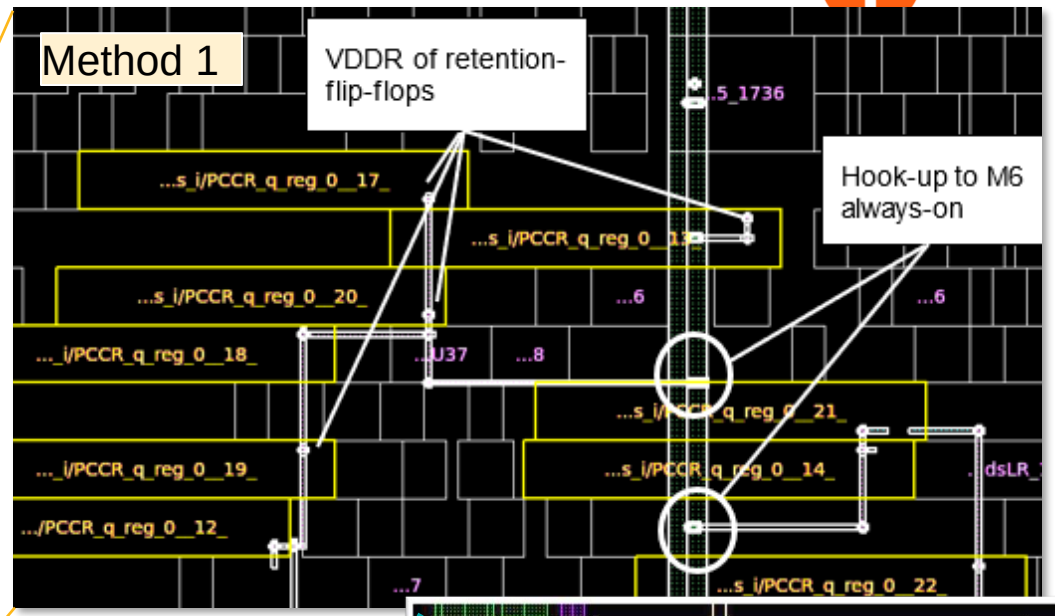
Dual-Rail cells can be for instance:

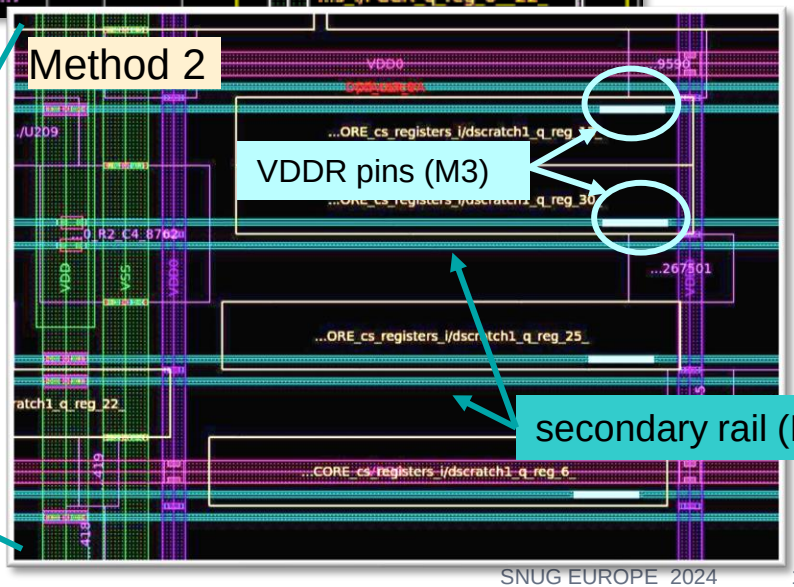
- Retention Flip Flops
- Isolation Cells
- Level Shifters

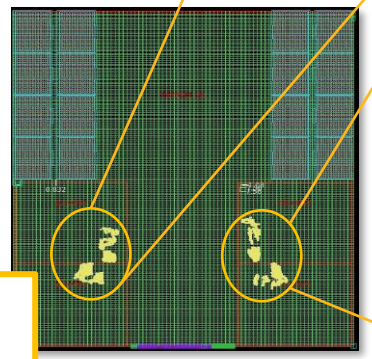
Two demonstrated methods how to connect backup-power of dual-rail cells:

Method 1 Connections via signal router

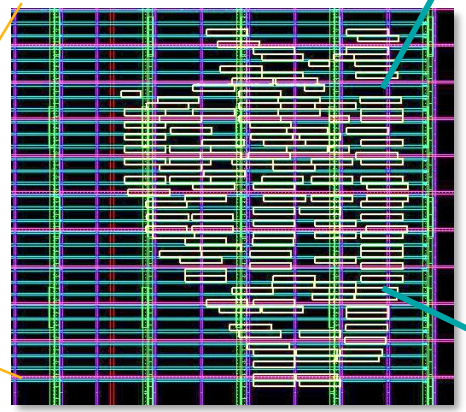
Method 2 Connection via secondary rail, aligned with position of backup power pins *(if the mv-cells' pin layout support this)*







Retention Flip-Flops



Which approach is more efficient depends … e.g., how many dual rail cells in the given area.

Verification

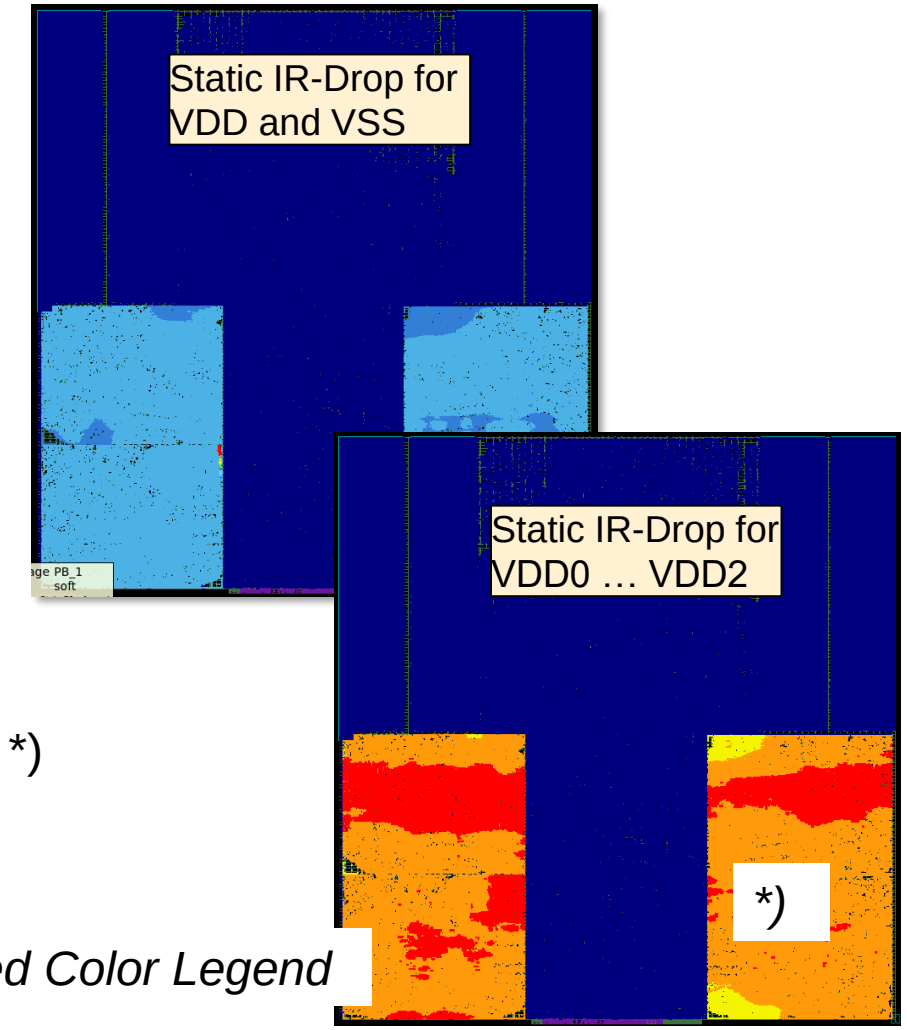
Redhawk, *(Formality only in Paper)*

Verification – IR Drop and EM

ICC2 In-Design Redhawk

- ✓ Using In-Design Redhawk out of the box was very effective
- ✓ Most of the inputs could be directly specified in ICC2
- ✓ Custom gsr-file used only for switch model files, EM-techfiles, and few others

```
analyze_rail -nets {VDD VDD0 VDD1 VDD2 VDD3 VSS} \
  -voltage_drop static \
  -electromigration \
  -extra_gsr_option_file ./ICC2/RH/pulp_cluster.gsr
```



✓IR-Drop (static, dynamic) and EM- analysis where well within limits *)

But .. one “nice” observation …

*) *Exaggerated Color Legend*

Verification – IR Drop and EM

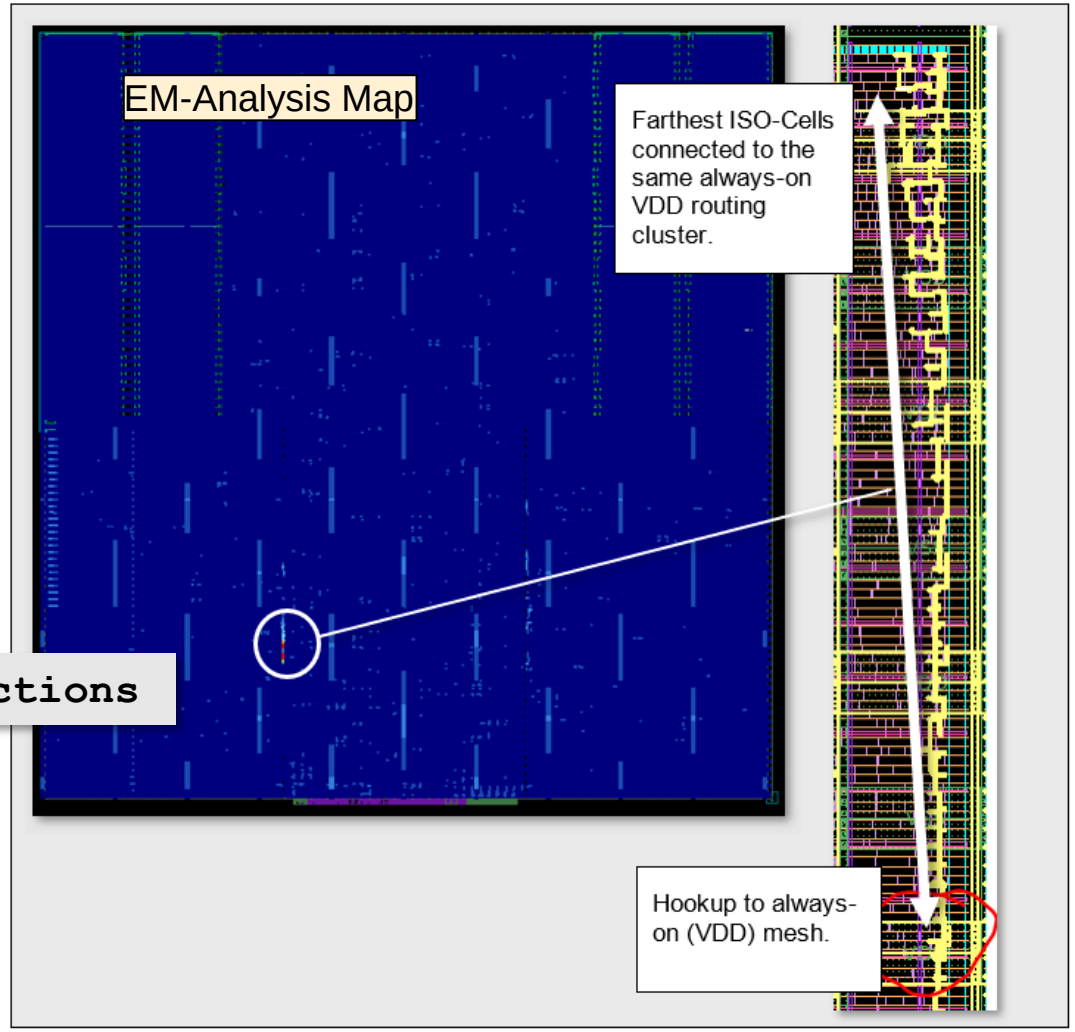
ICC2 In-Design Redhawk

But .. one “nice” observation …

- Analysis unveiled a weakness in the backup power connection
- Too many dual-rail cells (isolation buffers) where connected to just one “VDDR” branch

➢ Utilize following app-option

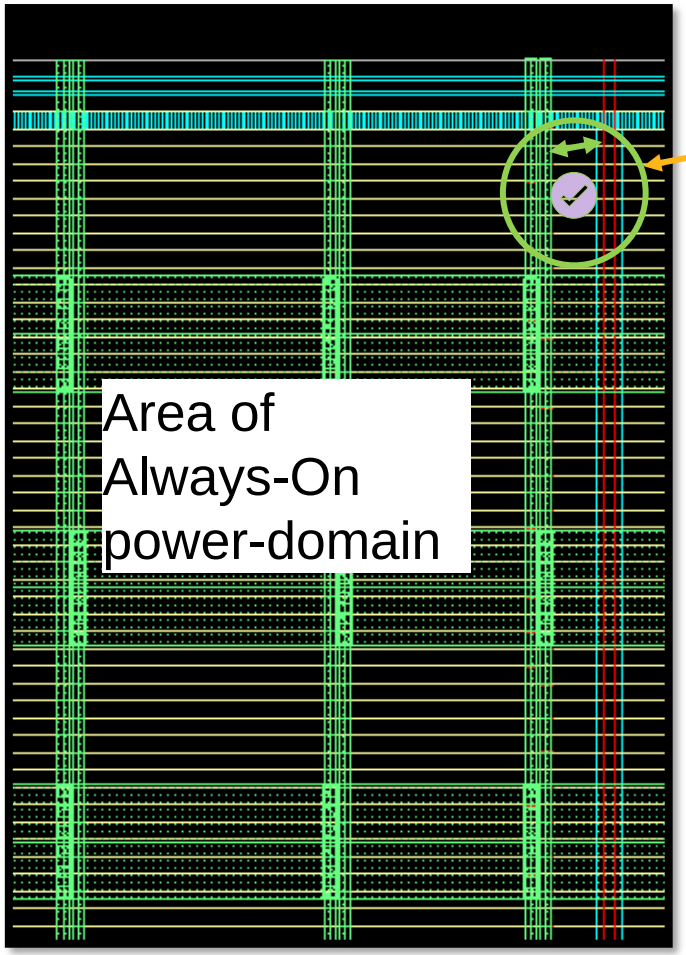
```
route.common.number_of_secondary_pg_pin_connections
```



Tips & Tricks

Tips & Tricks *(Just A Snippet From Paper)*

Avoiding pg-mesh conflicts




```
get_cells -intersect \
<tap-cell list>
```

was used here to space M6 VSS/VDD apart from M2 VNW and VPW

More tips & tricks in the paper

7. Tips and Tricks

- We used `get_cells -intersect` to identify M6 straps overlapping with bias tap cells, and then, automatically shifted them by a pre-defined offset. The "-intersect" option is very powerful and available on a couple of other commands too.
- We also used `get_attribute [get_cells -hierarchical -filter "full_name =~ CORE_${i}__*cs_registers_i* && is_sequential==true" bbox]` to determine the boundary enclosing the location of all retention flip flops (via tcl "lsort" of the resulting list); then further processed that information for `create_pg_straps` commands.
- We used `get_attribute [get_site_row -intersect <coordinate of the first stripe's start-point + small offset>] site_orientation` to determine the orientation of the first row with retention-flip-flops (flipped, non-flipped), and thus, what's the offset to the M3 VDDR pins respectively.
- It turned out to be beneficial to always work with a multiply of track-height and site-width, when calculating offsets, start points, etc., to allow pg-structure alignment with the std-cell library structure.
- Be aware, that certain objects provide boundary coordinates with and without margin, such as for `voltage_area_shapes` ("boundary" versus "bbox"), or, for macro-objects (e.g., "boundary" includes keepout margins) → important distinguishing when calculating coordinates during power planning.
- If you are stuck with pg-strategies (sometimes, they are not always intuitive to work with), use `create_pg_strap` command and do some "hand-calculation" to define the bounds of strap creation.
- Generally, utilizing the option `-voltage_areas` for various commands (such as `set_pg_strategy, create_tapcell` and `compile_boundary_cells`) is very convenient.
- `check_legality -verbose` → look out for `pg_drc` rules and examine the error types. This can be very helpful, in particular if certain `pg_drc` rules turn out to be over conservative (router techfile versus signoff DRC). Counter-measures then can be taken by utilizing the app option `place_legalize_disable_drc_rules`
- Guide bias tap cell placement in channels or, at irregular shape boundaries with `placement_blockages`
- Tag your individual pg-structures; this allows dedicated removal of structures again; in case they have to be corrected. E.g., `create_pg_vias -nets {VDD VSS} -tag \ mesh_M6_lib_connect ...`, allows you to remove just those vias again, if needed, by the following command: `remove_vias [get_vias -filter "tag == \ mesh_M64_lib_connect"]`; this approach is also applicable to shapes, and, it can be used in conjunction with `create_pg_strap, compile_pg`, for instance too.

Conclusion

Issues, Outlook, Wrap-up & Acknowledgement

Issues

- DC-NXT (STAR): “Dual-rail clock gates falsely inserted in always-on domain.”
  → *WA: Put them to don’t-use (luckily, we don’t need them).*

- ICC2 (STAR): “M3 VDDR pins can’t be legalized under M3 VDDR rail”
  → *WA: insert after place_opt*

→ A full list of issues (STARs, CASEs) can be found in the paper.
→ All issues are understood, solved or work arounds exist.

Outlook

- Additional experiments with other Std-Cell architectures in the work
- Further area reduction identified → Collaboration planned with Synopsys to utilize DSO.ai
- Update the flow to FusionCompiler

Wrap-up

- Dedicated technology and IP requires dedicated adjustments to the flow.
- Flow and commands work straight forward.
- The topic is not difficult, but “the devil is in the detail”; early planning and trials highly recommended.
- A lot of the work was alignment of objects (mathematical planning).
- A full demonstrator is available to GF customers.

Acknowledgement

- ETH Zurich for the RTL (Pulp-Cluster).
- Synopsys (Jens Peters and Michael Confal) for tool and flow support.


THANK YOU
Our Technology, Your Innovation™