In-rush Analysis For Power Gated Designs

RedHawk-SC Modular Training Series

Version : RedHawk-SC 2020_R3

Ansys

Prerequisites for the training

No	Training Program	Expectations – Must Know
1	RedHawk-SC Quick Start Training	• Reading in input data and performing data integrity checks
2	RedHawk-SC Dynamic Analysis Training	• Doing a dynamic analysis

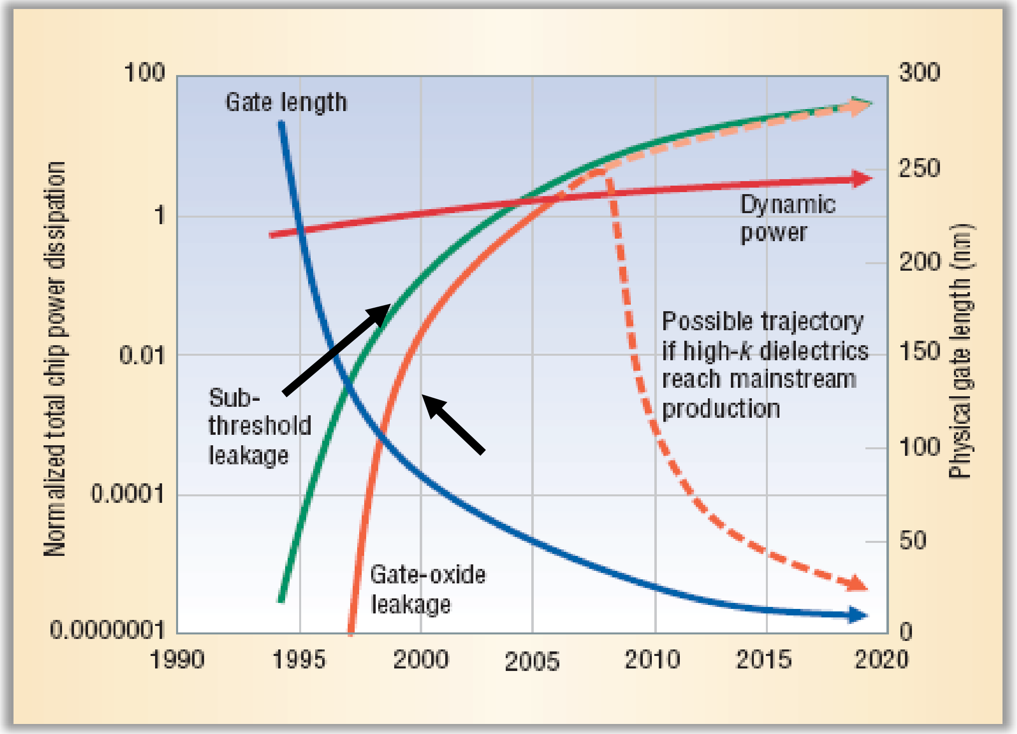
Training Agenda

- Power Gating Theory
- Input Data Requirement
- Inrush Analysis Flow In Redhawk SC
- Training Labs

Power Gating Theory

Controlling Leakage Power

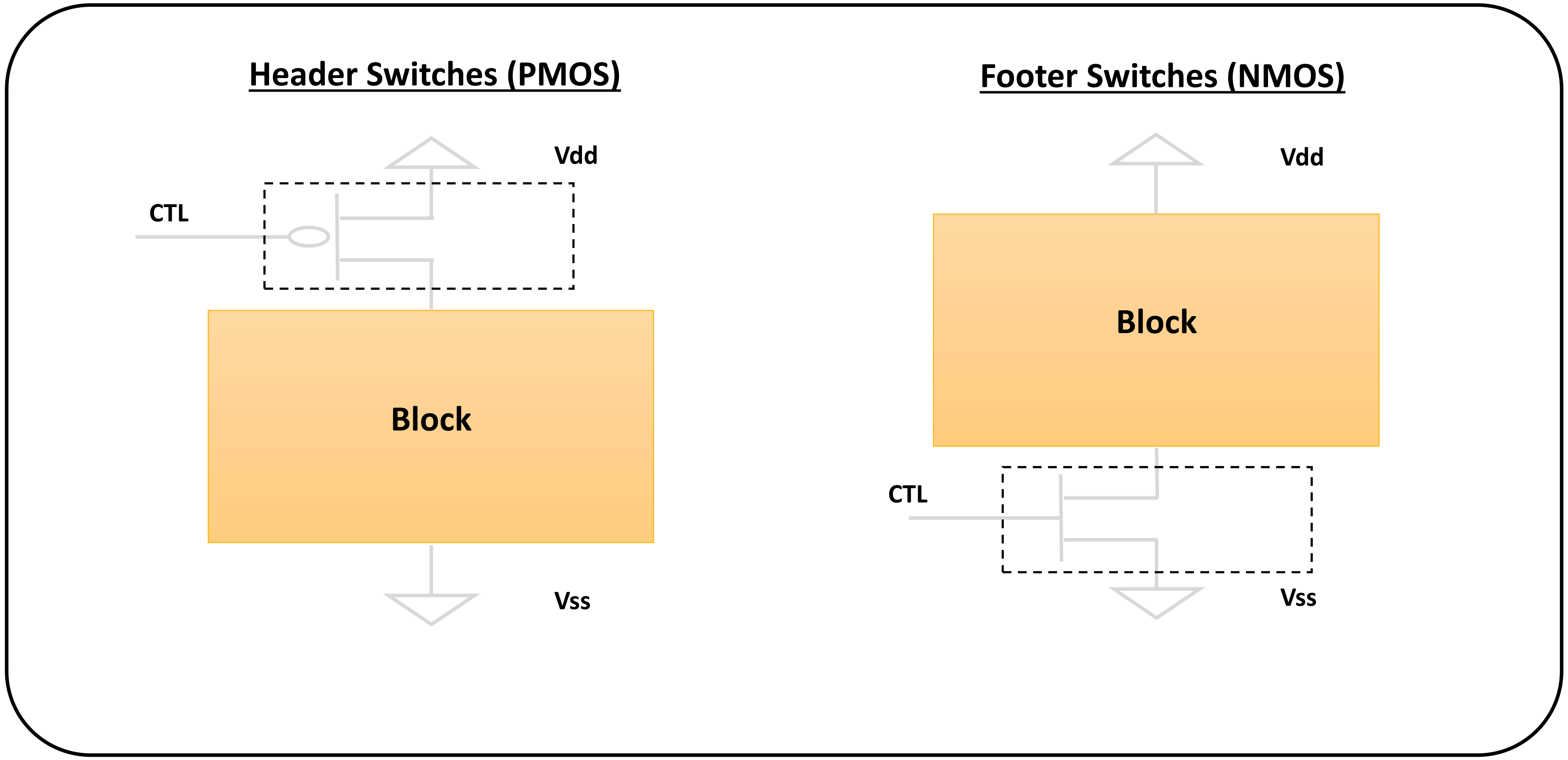
- **Next generation chips in 7nm and 5nm**
  - > 50 % of total power goes to leakage
- **Cell phone battery life**
  - Standby power cannot exceed 5% of full operational power
- **Product's competitiveness depends on its ability to control leakage**



Kim et al, "Leakage Current: Moore's Law Meets Static Power," *IEEE Transactions on Computers,* Vol. 36, No. 12, December 2003, pp. 68-77

Different Switch Architectures

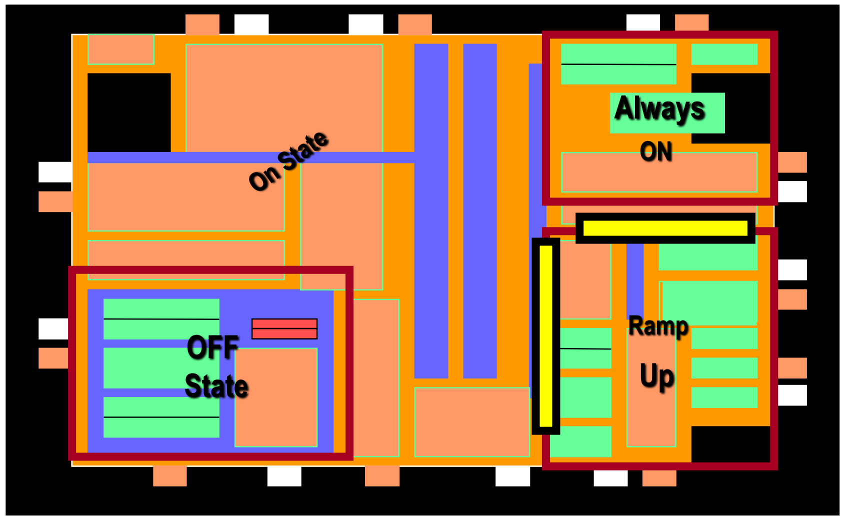
Power Gating Switch Models



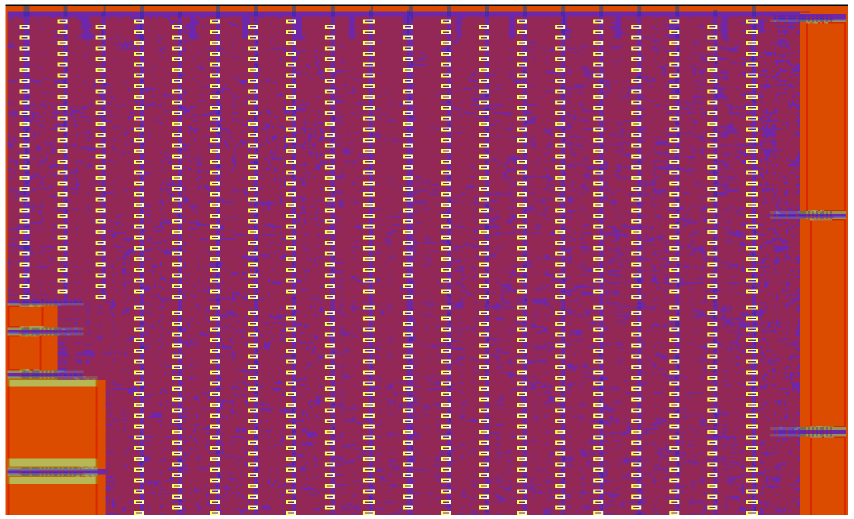
Different Switch Implementation Methods

Full-chip Implementation

Coarse Grain



Fine Grain



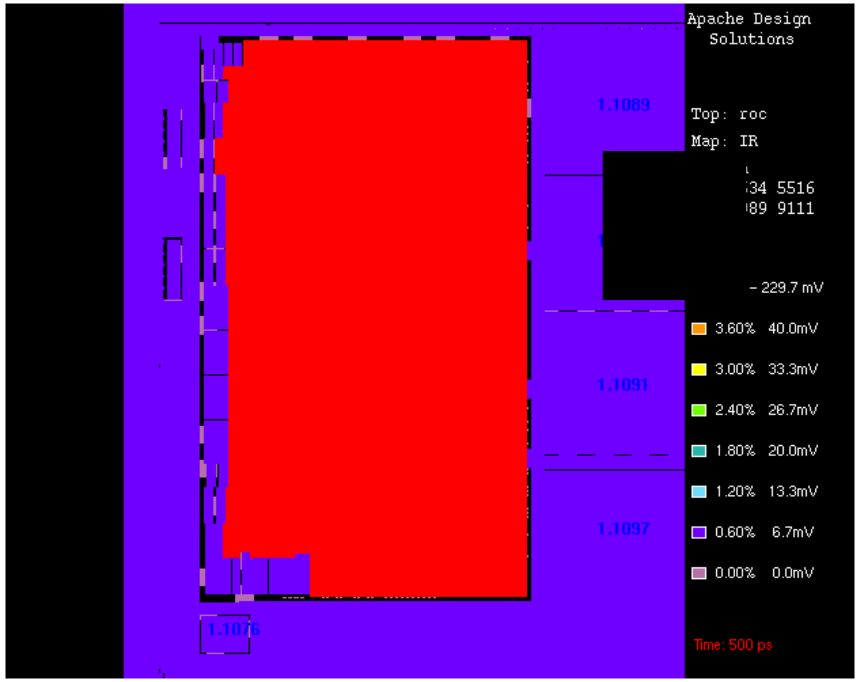
Switches highlighted in yellow

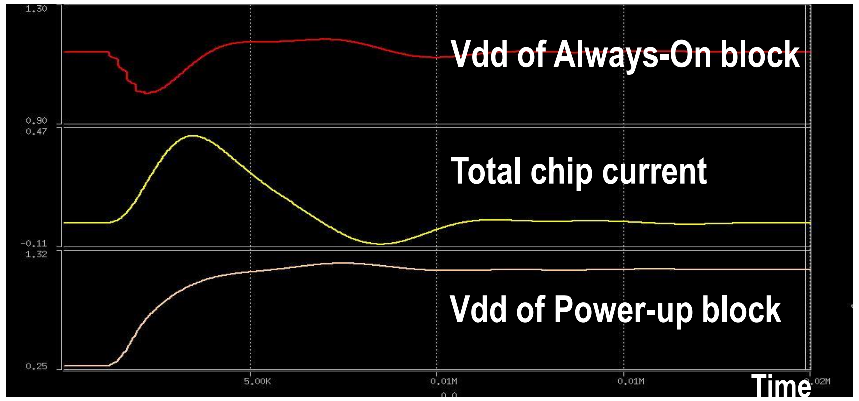
Different Switch Operating States

Mode	Comment
OFF	When block is inactive
ON	When block is functional
Powerup	When block is transitioning from inactive to functional state
Powerdown	When block is transitioning from functional state to inactive state

Challenges in Power Gated Designs

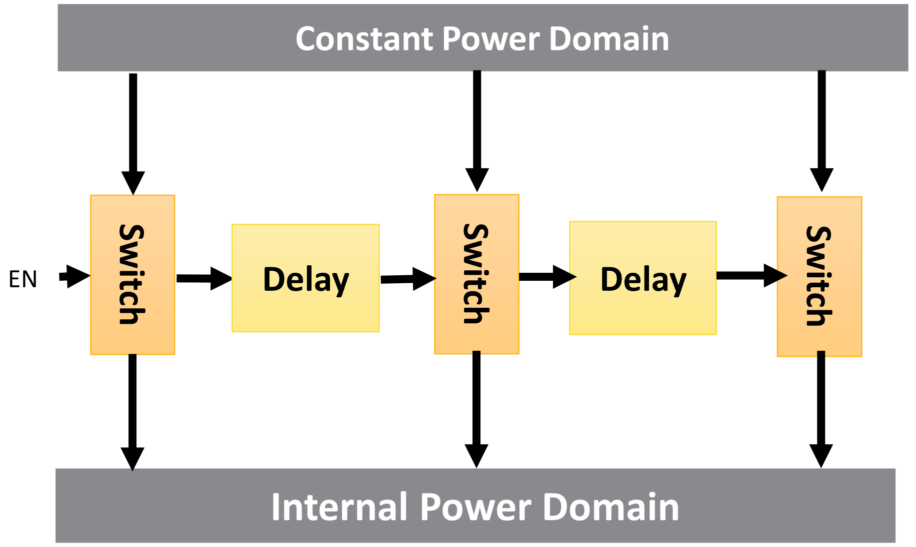
- High surge of current to charge/ discharge the internal capacitors during powerup
- This can cause huge voltage drop on neighboring “always on” blocks
- Peak current flowing through the switches should be within limits

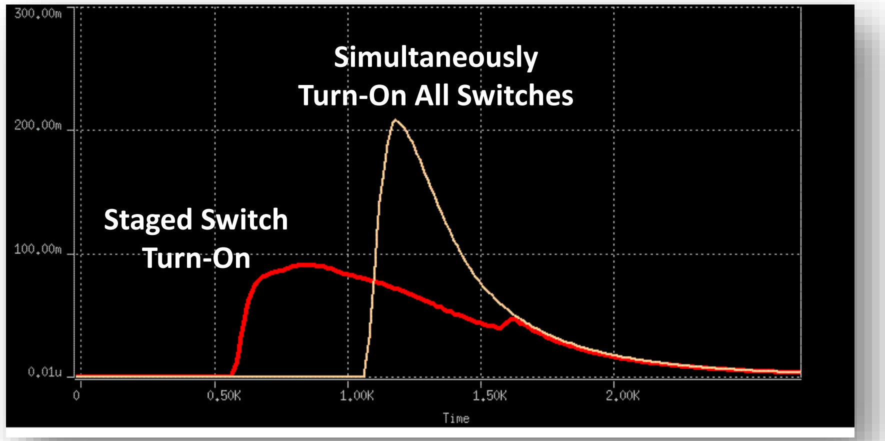




Challenges in Power Gated Designs *(Cont'd)*

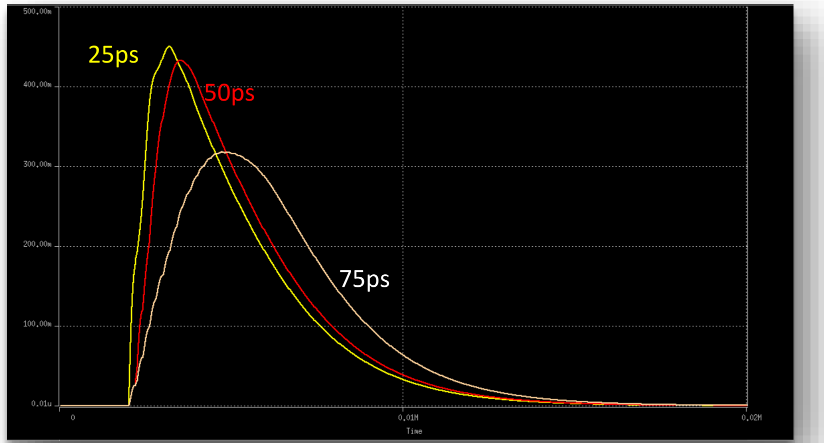
- Daisy chain structure for switch enable helps reducing rush current



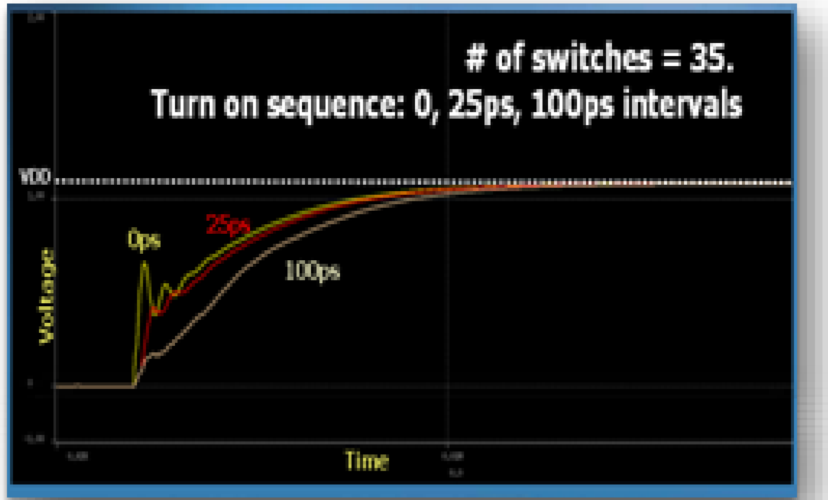


Challenges in Power Gated Designs *(Cont'd)*

- Rush current can be reduced by increasing the delay between switches

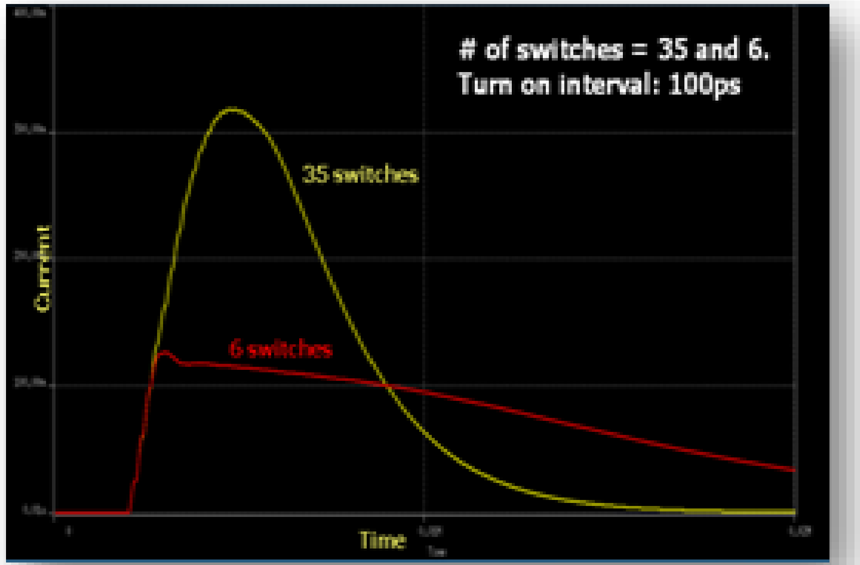


- Increasing the delay will also cause an increase in block wake-up time



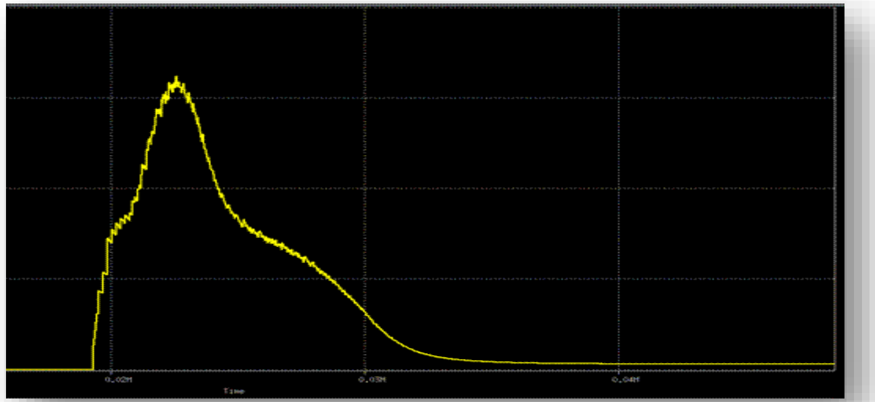
Challenges in Power Gated Designs *(Cont'd)*

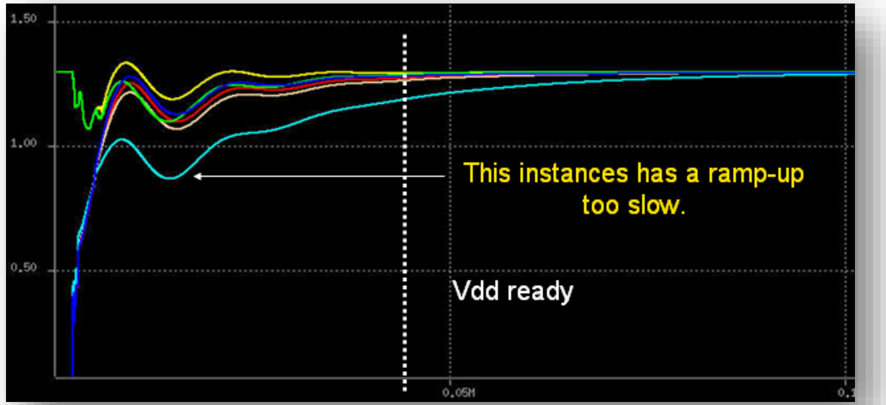
- Rush current can be reduced by reducing the number of switches
- But this will increase the current flow through individual switches
- Also increases the ON state power grid resistance
- Increases the wake-up time



Challenges in Power Gated Designs *(Cont'd)*

- High rush current can cause oscillations in the power ground network due to fast turn ON (L di/dt)





Challenges in Power Gated Designs *(Cont'd)*

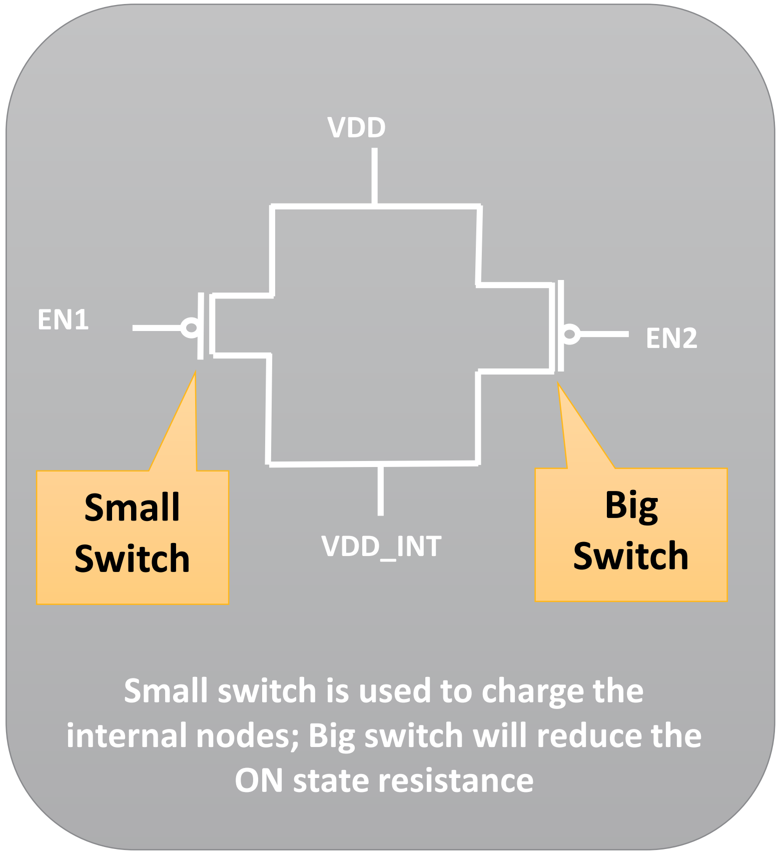
- Power switches add more resistance in the power grid
- Increase in static/dynamic voltage drop in ON state analysis
- Drop increases if you reduce the number of switches
- Hot spot regions will require more switches

Can we increase the number of switches to reduce the ON state drop ?

- Increase in area
- More off state leakage current
- High rush current

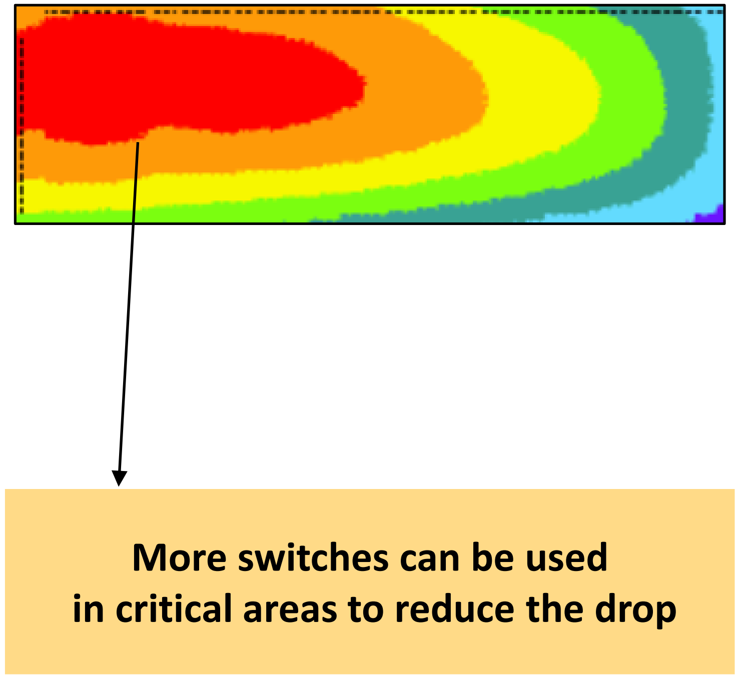
Challenges in Power Gated Designs *(Cont'd)*

- Alternate switch architecture to reduce rush current and ON state voltage drop



Small switch is used to charge the internal nodes; Big switch will reduce the ON state resistance

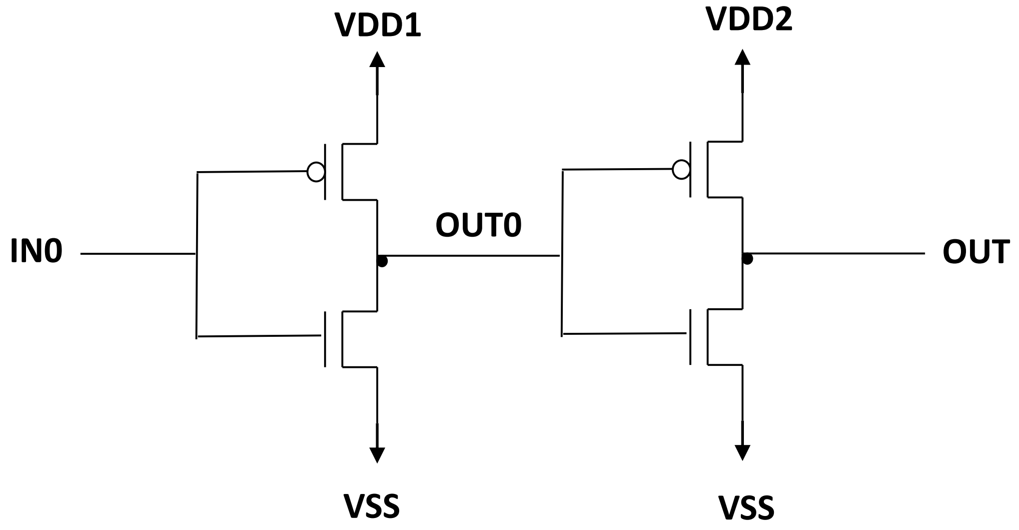
- Switch placement can be improved to optimize the ON state drop

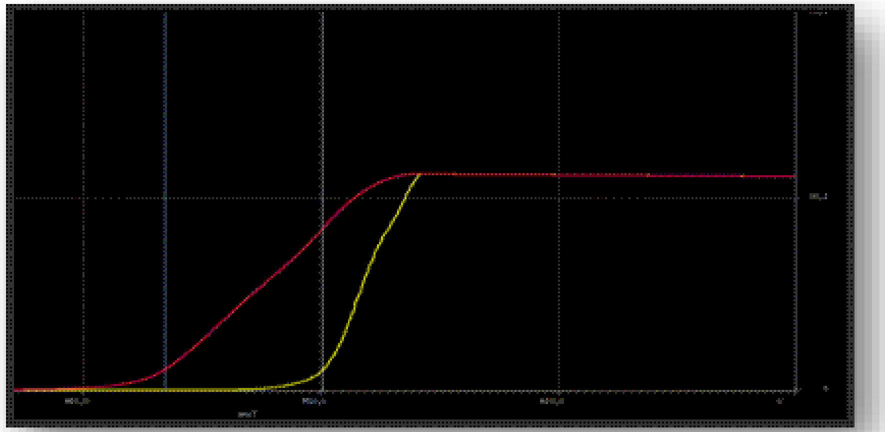


More switches can be used in critical areas to reduce the drop

Challenges in Power Gated Designs *(Cont'd)*

- If VDD2 ramps up faster than VDD1, OUT0 can stay at a level above threshold voltage for more time.
- There could be high crowbar current in the receiver cell
- More significant when wakeup time is more
- Need to analyse the voltage differential analysis on driver receiver pair





Input Data Requirements

Input Requirements

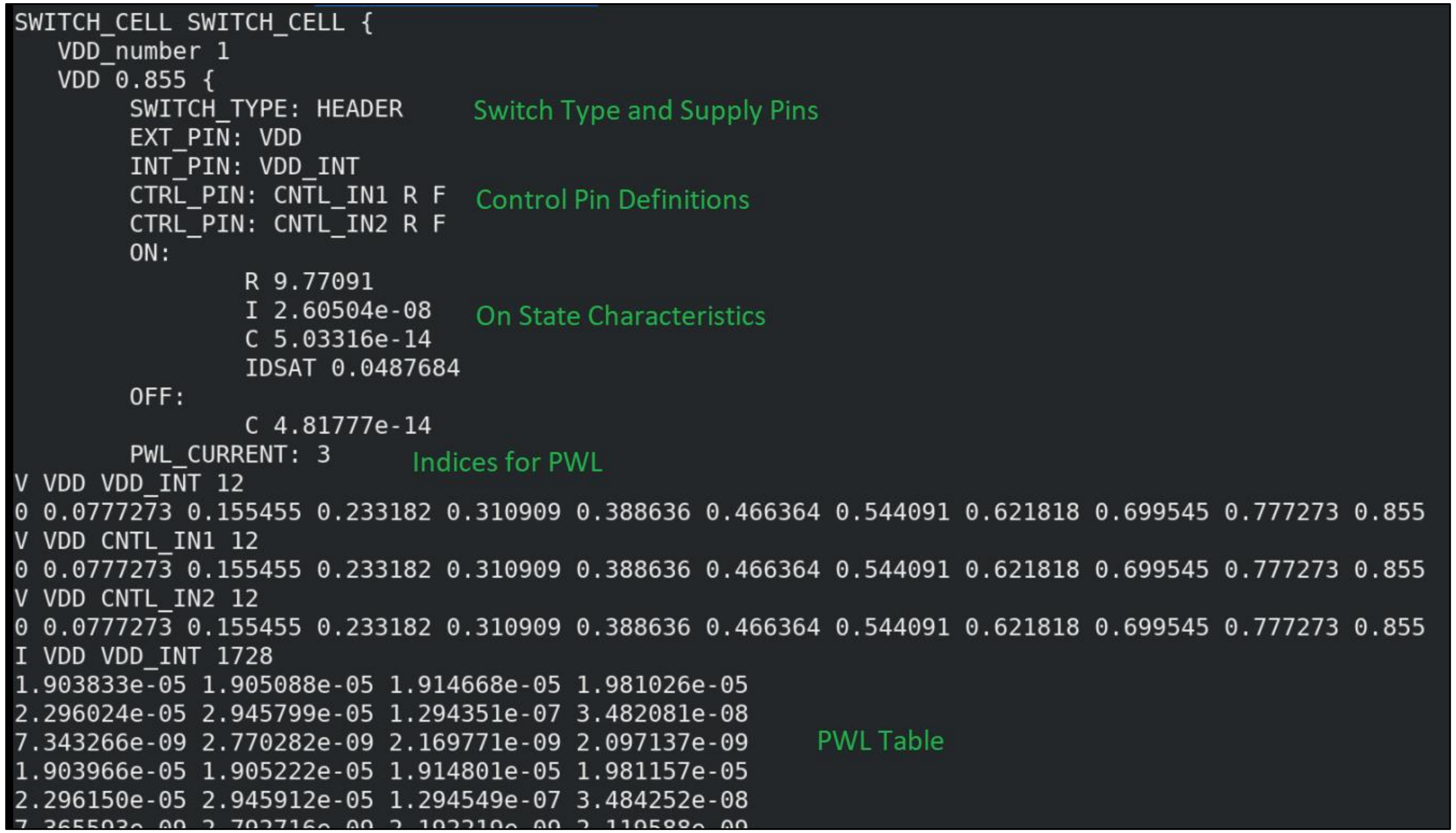
- LibertyView
  - APL Switch Model Files (PWL or 3D models mandatory!)
  - APL PWCAPs (For offstate voltage calculation)

- ScenarioView
  - LSO for switch control pins
    - User can apply LSO per switch or LSO at starting switch of daisy chain

  - External currents dumped from rampup AnalysisView

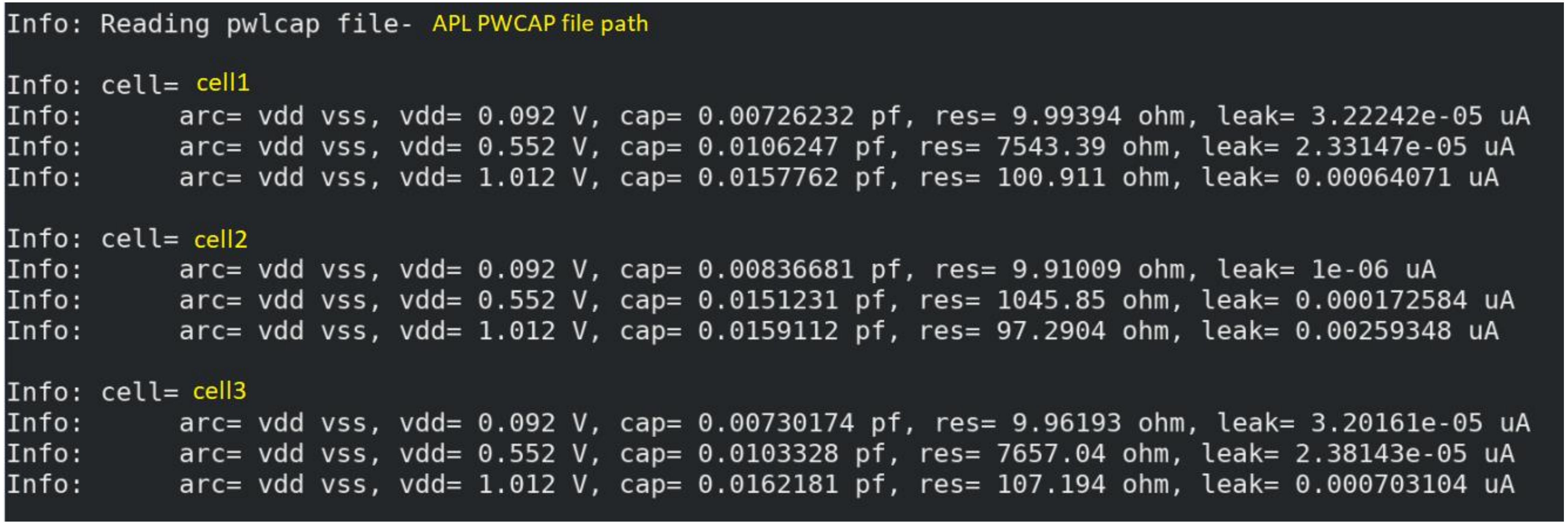
Inputs: Switch Model File

- 3D PWL Switch Model
  - ‘MD_PWL 1’ in APL Config



```
SWITCH_CELL SWITCH_CELL {
   VDD_number 1
   VDD 0.855 {
        SWITCH_TYPE: HEADER
        EXT_PIN: VDD
        INT_PIN: VDD_INT
        CTRL_PIN: CNTL_IN1 R F
        CTRL_PIN: CNTL_IN2 R F
        ON:
                R 9.77091
                I 2.60504e-08
                C 5.03316e-14
                IDSAT 0.0487684
        OFF:
                C 4.81777e-14
        PWL_CURRENT: 3
V VDD VDD_INT 12
0 0.0777273 0.155455 0.233182 0.310909 0.388636 0.466364 0.544091 0.621818 0.699545 0.777273 0.855
V VDD CNTL_IN1 12
0 0.0777273 0.155455 0.233182 0.310909 0.388636 0.466364 0.544091 0.621818 0.699545 0.777273 0.855
V VDD CNTL_IN2 12
0 0.0777273 0.155455 0.233182 0.310909 0.388636 0.466364 0.544091 0.621818 0.699545 0.777273 0.855
I VDD VDD_INT 1728
1.903833e-05 1.905088e-05 1.914668e-05 1.981026e-05
2.296024e-05 2.945799e-05 1.294351e-07 3.482081e-08
7.343266e-09 2.770282e-09 2.169771e-09 2.097137e-09
1.903966e-05 1.905222e-05 1.914801e-05 1.981157e-05
2.296150e-05 2.945912e-05 1.294549e-07 3.484252e-08
7.365593e-09 2.792716e-09 2.192219e-09 2.119588e-09
```

Inputs: APL PieceWiseCap (PWCAP) File



```
Info: Reading pwlcap file- APL PWCAP file path

Info: cell= cell1
Info:       arc= vdd vss, vdd= 0.092 V, cap= 0.00726232 pf, res= 9.99394 ohm, leak= 3.22242e-05 uA
Info:       arc= vdd vss, vdd= 0.552 V, cap= 0.0106247 pf, res= 7543.39 ohm, leak= 2.33147e-05 uA
Info:       arc= vdd vss, vdd= 1.012 V, cap= 0.0157762 pf, res= 100.911 ohm, leak= 0.00064071 uA

Info: cell= cell2
Info:       arc= vdd vss, vdd= 0.092 V, cap= 0.00836681 pf, res= 9.91009 ohm, leak= 1e-06 uA
Info:       arc= vdd vss, vdd= 0.552 V, cap= 0.0151231 pf, res= 1045.85 ohm, leak= 0.000172584 uA
Info:       arc= vdd vss, vdd= 1.012 V, cap= 0.0159112 pf, res= 97.2904 ohm, leak= 0.00259348 uA

Info: cell= cell3
Info:       arc= vdd vss, vdd= 0.092 V, cap= 0.00730174 pf, res= 9.96193 ohm, leak= 3.20161e-05 uA
Info:       arc= vdd vss, vdd= 0.552 V, cap= 0.0103328 pf, res= 7657.04 ohm, leak= 2.38143e-05 uA
Info:       arc= vdd vss, vdd= 1.012 V, cap= 0.0162181 pf, res= 107.194 ohm, leak= 0.000703104 uA
```

- Information used: Max C, Avg R, Offstate Leakage

Inputs: LibertyView

Use the following template to specify the file pointers

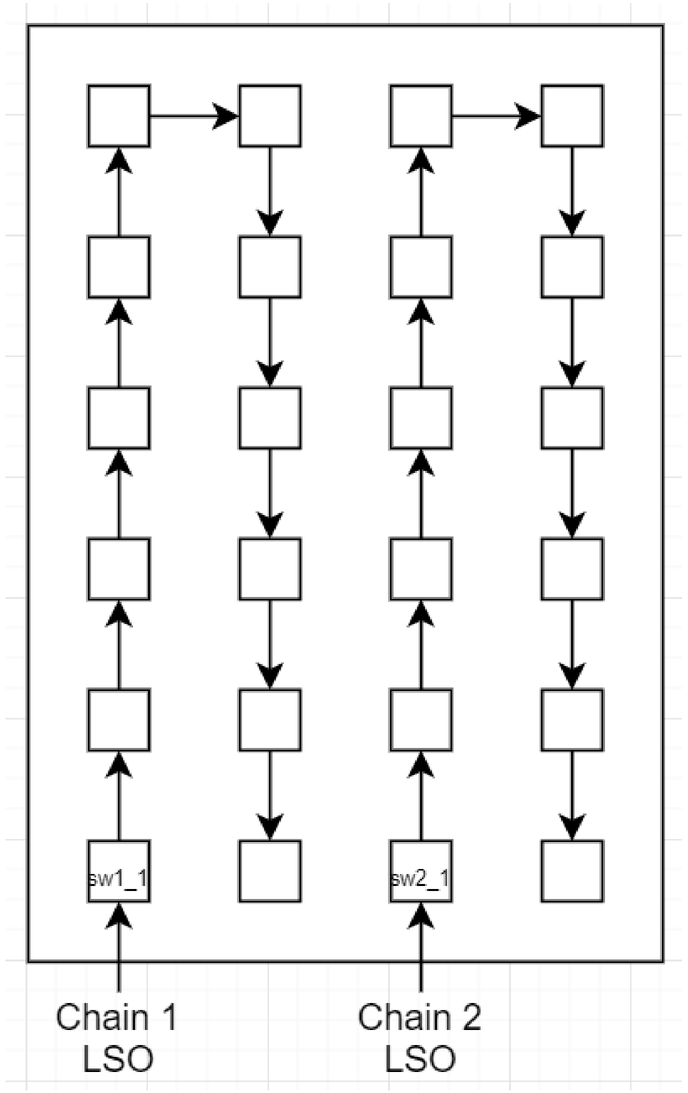
```
apl_files = [ <.. add any APL current files etc ..>,
         {‘file_name’ : <path_to_pwcap_file_1> },
         {‘file_name’ : <path_to_pwcap_file_2> },]

switch_files = [
         { ‘file_name’ : <path_to_switch_model_file_1>, ‘temperature’ : <T> }
         { ‘file_name’ : <path_to_switch_model_file_2>, ‘temperature’ : <T> }]
```

Above pointers passed using the following API

```
lv = db.create_liberty_view( ...,
 apl_files = apl_files,
 apl_switch_file_names = switch_files)
```

Inputs: Logic Signal Override

- LSO is used to assign timing at the power gate control pins
- Rising or falling control signal at head of chain
- RHSC propagation downstream
- Also possible to control per power gate



```
# version statement is mandatory
$version = 1.0
# Set the time unit
$time_unit = 1.0e-09
# Each entry is in the following format
# @ip <inst_name>/<pin_name> {
#     (t1, logic_at_t1),
#     (t2, logic_at_t12,
#     (t1, logic_at_t1),
#     (tn, logic_at_t1n,
# }
# Specify a logic rise at the head of switch chain 1 ,
# Starting at 10 ns with a slew of 40 ps
@ip head_sw_1/ctrl_in {
    (0.0, 0),
    (10.0, 0),
    (10.04, 1)
}
# Specify a logic fall at the head of switch chain 2,
# Starting at 15 ns with a slew of 100 ps
@ip head_sw_2/ctrl_in {
    (0.0, 1)
    (15.0, 1),
    (15.1, 0)
}
```

Helpful Functions: Logic Signal Override

The following RedHawk-SC API's will be useful while creating the LSO

```
# Returns a dict of { Net : List of Instances }

>>> power_gate_instances = dv.get_power_gate_instances()
```

```
# Returns a dict of { Net : List of Instance, Pin tuples }

>>> power_gate_instance_pins = dv.get_power_gate_instance_pins()
```

```
# Check power gate annotation in TimingView

>>> tv.get_tw_attributes( switch_instance )
```

Scripts are available for

- Generating an LSO from a TimingView (`get_lso_from_tv`)

Inrush Analysis Flow In Redhawk SC

How the Flow Works in RHSC

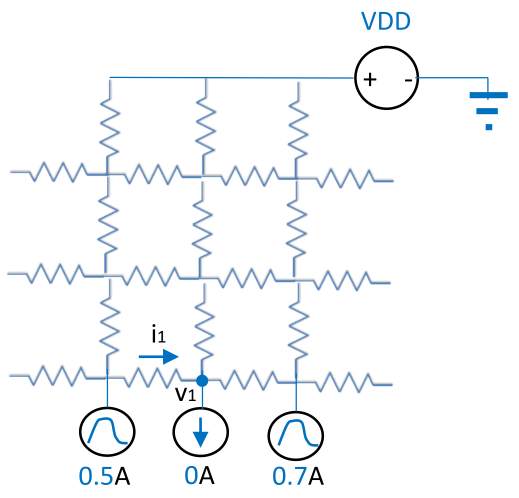
- Principle of superposition

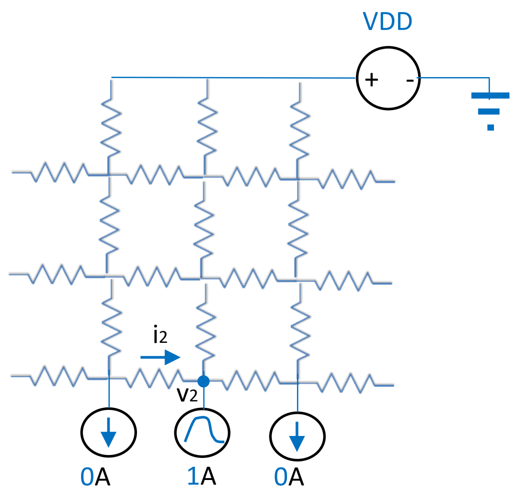
- Requires (at least) 3 sets of ScenarioViews and AnalysisViews
  - For characterizing each rampup block
  - To read in the characterized currents
  - To provide activity on the always-on domain

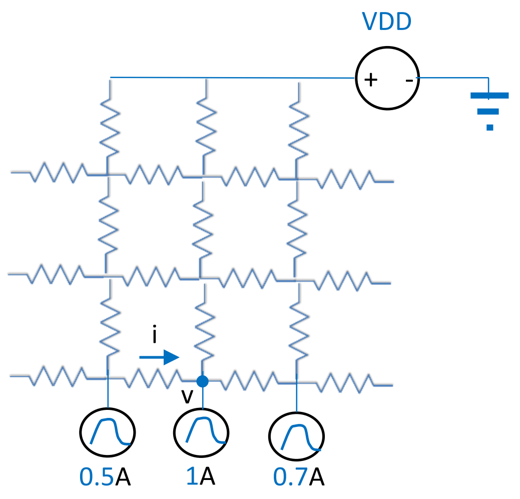
- AnalysisComboView feature
  - User can ‘overlay’ the rampup AV over the baseline AV
  - Analyze the impact of rampup on this baseline AV

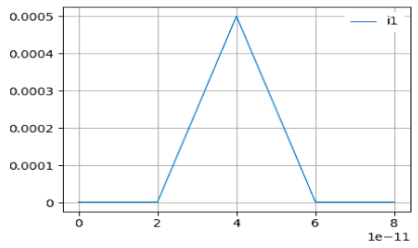
Principle of Superposition

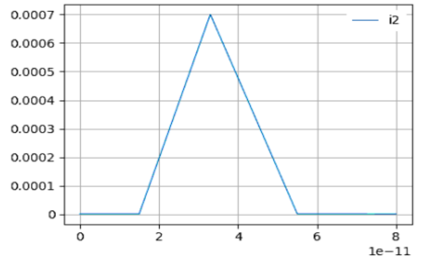
The total current in any part of a linear circuit equals the algebraic sum of the currents produced by each source individually.

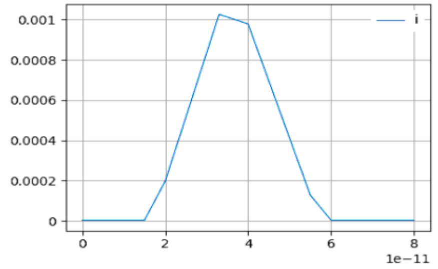












$$\boxed{i = i_1 + i_2}$$

In-rush flow overview

Step1 Characterizing Rampup Blocks

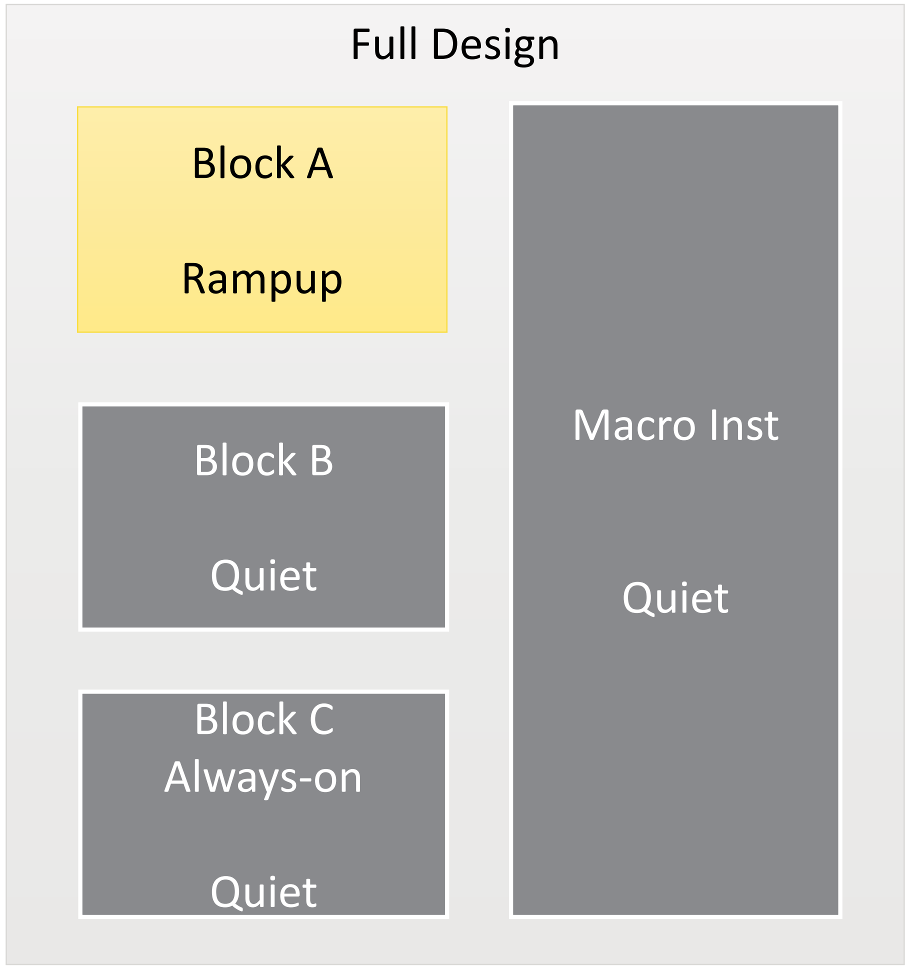
Step2 External ScenarioView and AnalysisView

Step3 Always-On Activity & AnalysisComboView

Flow Step 1: Characterizing Rampup Blocks

- Rampup on one power gated block, quiet everywhere else
- Save the power gate currents to a file
- Repeat for each rampup block

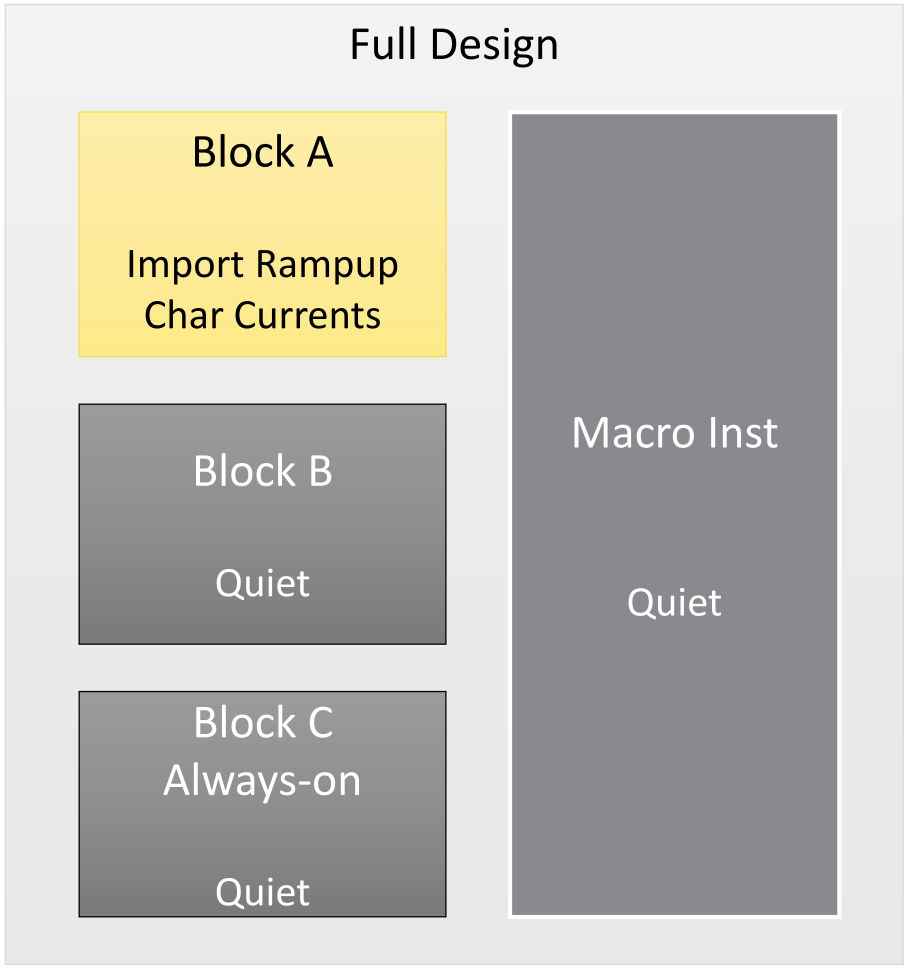
- Views can be used for rampup time, total power gate currents, switch turn-on heatmaps



Flow Step 2: Creating External ScenarioView and AnalysisView

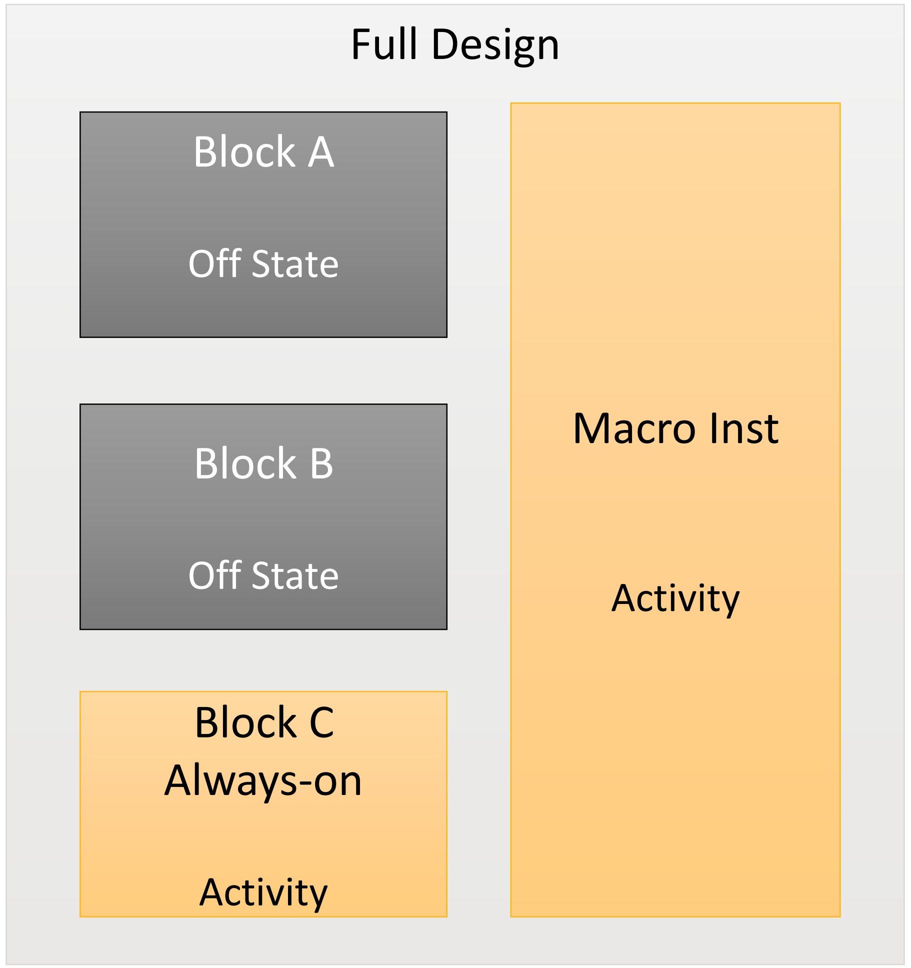
- Read in the characterized currents file from step 1
- Create ScenarioView and AnalysisView
- Very fast intermediate step

- These views can be queried for IR heatmaps – instance IR, node IR etc.



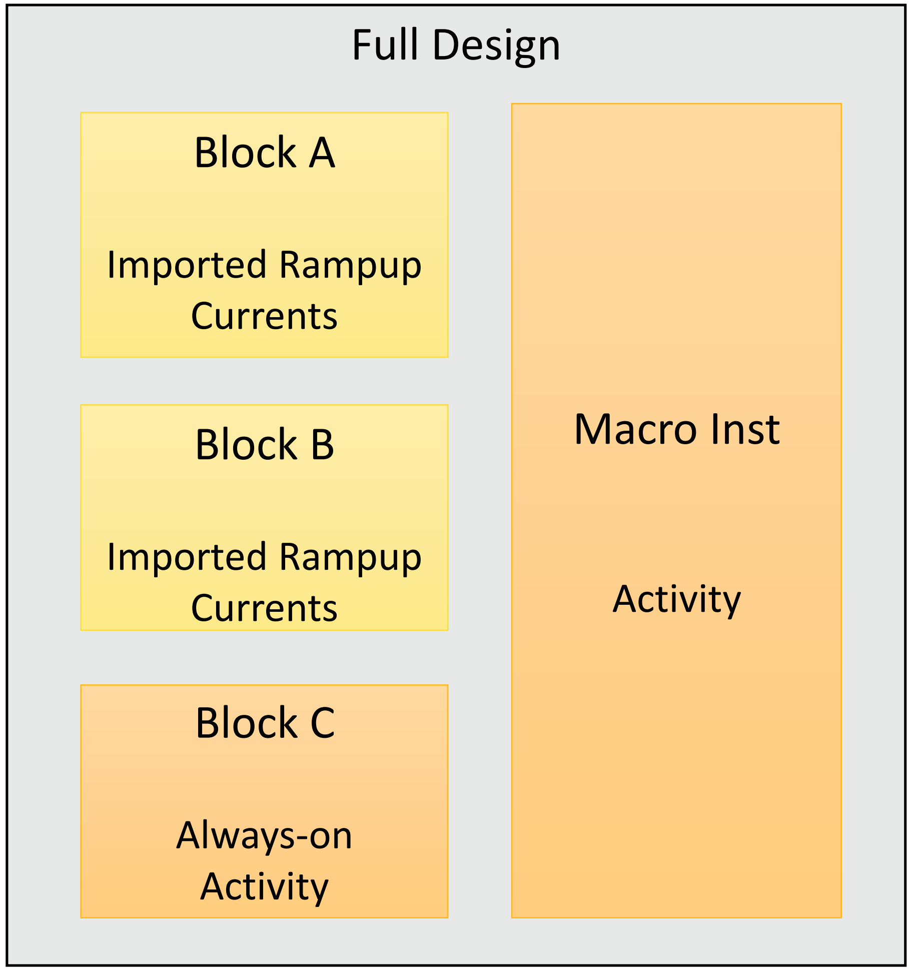
Flow Step 3a: Analysis For Always-On Activity Blocks

- Regular RHSC ScenarioView and AnalysisView
  - Can be vector-based, vectorless, or any combination
- Previously characterized rampup blocks in off state
- Gives IR drop, heatmaps, voltages stats etc **when rampup is not considered**



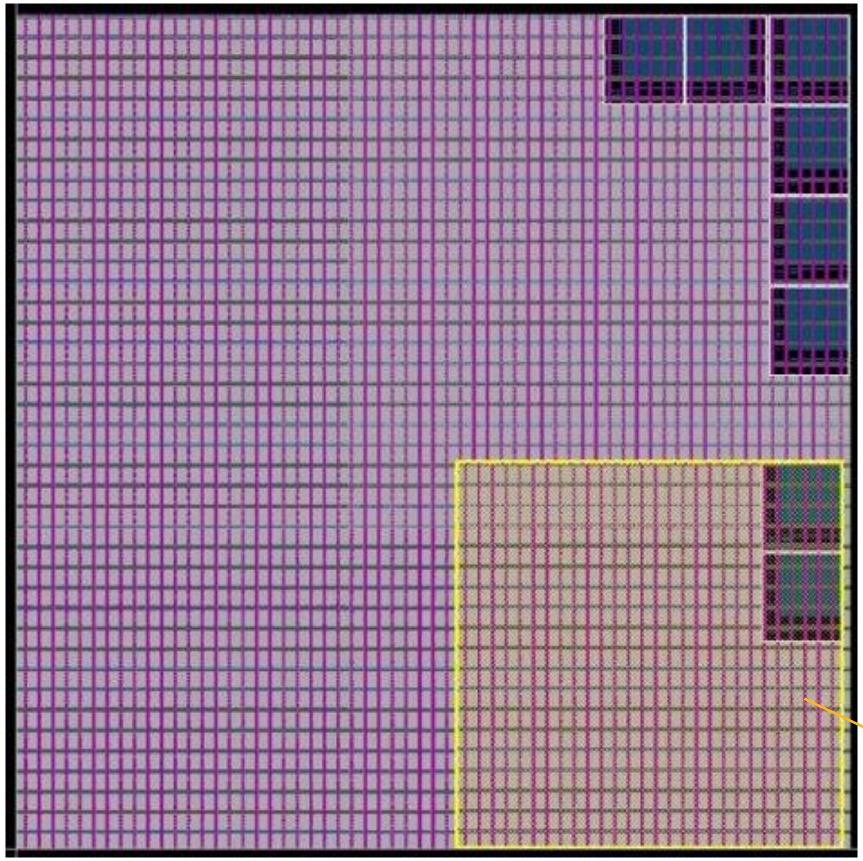
Flow Step 3b: Creating Combined Results In AnalysisComboView

- Overlay the rampup currents on base AnalysisView
- Can be inserted with separate offset per rampup AnalysisView
- Gives IR drop, heatmaps, voltages stats etc **when rampup is considered**



Training labs

Training Testcase (Galaxy) Overview



Galaxy Testcase Details	
Instance count	1.2M
Node count	5M
Domains	1 Always-ON VDD domain 1 Power-gated VDD domain 1 GND domain
#Memories	8
Tech Node	45nm
#Layers	10

Power Gated Block

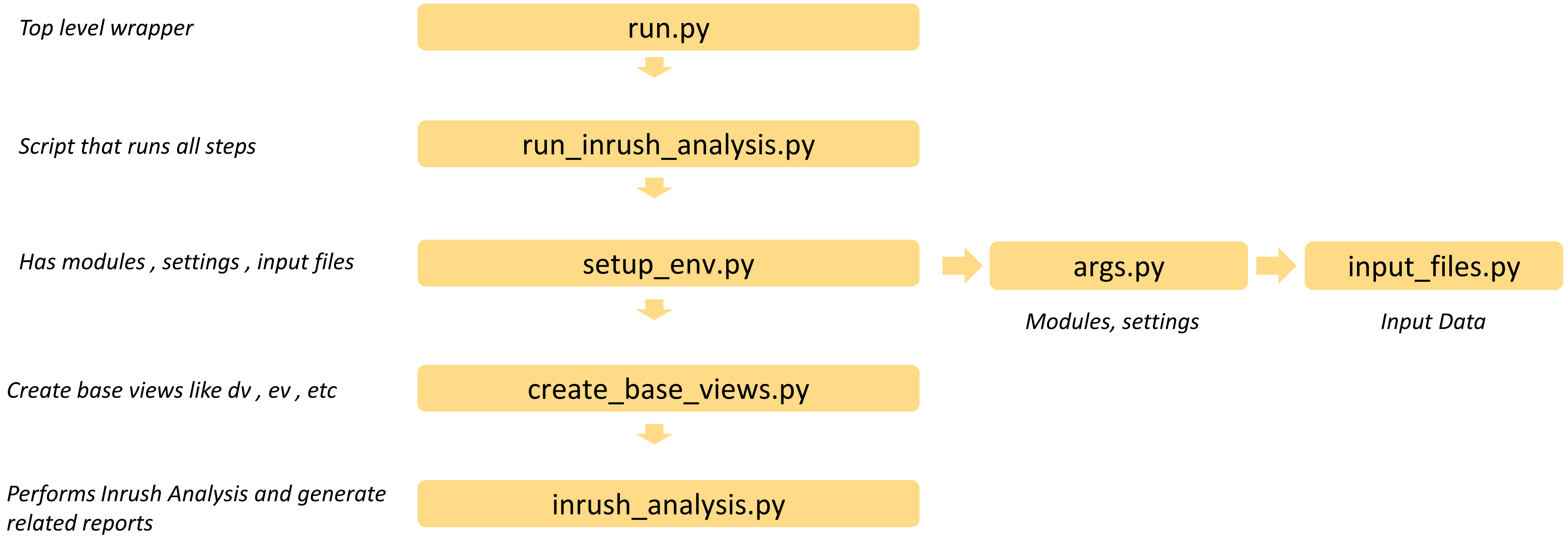
Acknowledgements:

Galaxy Testcase is created using Silvaco 45nm FreePDK libraries through Si2 OpenAccess Program

Getting Familiar with Training Lab Scripts

Lab Instructions :

- Download the Galaxy_Training.tar.gz Training Bundle
- Move to Modular_Training/08_InRush_Analysis directory



Top level wrapper
run.py
Script that runs all steps
run_inrush_analysis.py
Has modules , settings , input files
setup_env.py
args.py
Modules, settings
input_files.py
Input Data
Create base views like dv , ev , etc
create_base_views.py
Performs Inrush Analysis and generate related reports
inrush_analysis.py

Getting Familiar with Training Lab Scripts

File : run.py

```
include('../scripts/run_inrush_analysis.py')
```

Single command file that will invoke all steps

File : ../scripts/run_inrush_analysis.py

```
include('setup_env.py')
include('create_base_views.py')
include('inrush_analysis.py')
```

Script runs everything from scratch and completes inrush analysis

Setting up the environment

File : scripts/setup_env.py

```
import pprint
open_scheduler_window()

ll = create_local_launcher('local')
register_default_launcher(ll, min_num_workers=10)


design_data_path = '../../design_data/’


db = gp.open_db('db’)

# Auto-load view tags from an existing db
# Needed only for incremental(jump-start) runs
gp.populate_view_tags()


include('args.py')
```

Setup for auto launching workers; here local launcher used

Set `design_data_path` variable to central design data path area

Set the DB location settings using the open_db command

Auto view-tag loading for incremental (jump-start) runs

Set the arguments for various view creation commands in args.py

Specifying modules , arguments , settings

File : scripts/args.py

```
import package


include('./input_files.py')


options = get_default_options()
focus_pg_nets = ['VDD', 'VSS']
voltage_levels = {'VDD':1.1, 'VSS':0.0}
```

Import required Python modules

Include all design input file pointers

Include all tool options , analysis settings , etc

Specifying input files and settings

File : scripts/input_files.py

```
def_files = [
design_data_path + '/defs/Galaxy.def'
.. .. ..
]


lef_files = [
design_data_path + '/lefs/switch_cell.lef',
.. .. ..
]
```

Specify all the design files in input_files.py

File : scripts/args.py

```
dv0_args = dict(
    def_files=def_files,
    lef_files=lef_files,
    focus_pg_nets=focus_pg_nets,
    top_cell_name='Galaxy',
    tag='dv0',
    options=options)
```

Setup arguments for view creation commands in args.py

File : scripts/create_base_views.py

```
dv0 = db.create_design_view(tech_view=nv,
lib_views=lv, **dv0_args)
```

Call the arguments in view creation commands

Launching RedHawk-SC - Get started with your labs

- Batch mode execution example:
  - `<path_to_rhsc_installation>/bin/redhawk_sc run.py`
- Interactive mode execution example:
  - `<path_to_rhsc_installation>/bin/redhawk_sc –i`
  - It needs an exit() command in script or entered manually to exit the Python shell
- Connecting to a live RedHawk-SC run:
  - RedHawk-SC allows querying of data/results from an active session, by remotely attaching to the session
  - Multiple users can attach to the same session from multiple machines for querying/viewing results
  - `<path_to_rhsc_installation>/bin/redhawk_sc –r <gp_dir>`
- Execution Log Files:
  - All RHSC log files reside by default in gp<> folder
    - If the run is fired in the same directory, tool will create gp.1, gp2 incrementally
    - 'latest.gp' link will point to the most recent gp directory
  - Main log file for RedHawk-SC would be <gp_directory>/run.log file.

Characterizing Rampup Blocks : Syntax

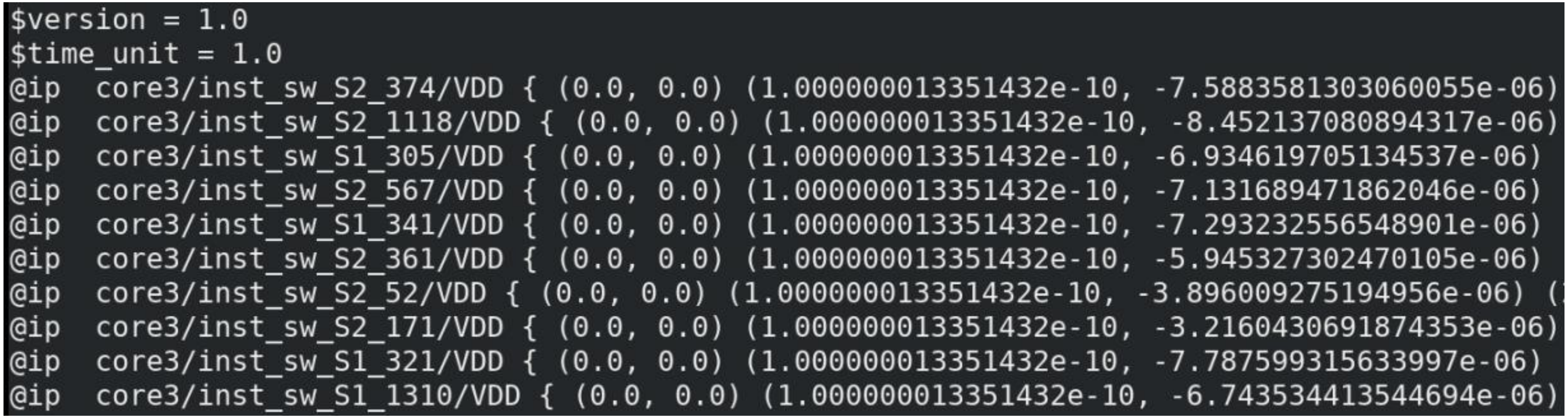
Step1: Create a quiet ScenarioView using LSO files

```
activity_level = [{ 'block_name' : '*', activity : 0.0 }]
clock_activity = { '*' : 0.0 }

scn_quiet = db.create_scenario_view(..., tv = tv, lso_files = <pointer_to_lso_file>),
activity_level =
activity_level, clock_source_toggle_rates = clock_activity, scenario_type =
'RampUp')
```

Step 2: Create the AnalysisView using this ScenarioView and write out the power gate currents

```
av_ru_quiet = db.create_analysis_view(..., scenario_views = scn_quiet, ramp_up_nets =
[Net('<block_internal_net>')], keep_stats_level = KeepStats('Low', pin_voltages =
True, power_gate_currents = True))

write_power_gate_currents(av_char, '<block_name>.power_gate_currents',
[Net('<block_internal_net>')])
```

Characterized Currents File



```
$version = 1.0
$time_unit = 1.0
@ip  core3/inst_sw_S2_374/VDD { (0.0, 0.0) (1.000000013351432e-10, -7.588358130306005 5e-06)
@ip  core3/inst_sw_S2_1118/VDD { (0.0, 0.0) (1.000000013351432e-10, -8.452137080894317e-06)
@ip  core3/inst_sw_S1_305/VDD { (0.0, 0.0) (1.000000013351432e-10, -6.934619705134537e-06)
@ip  core3/inst_sw_S2_567/VDD { (0.0, 0.0) (1.000000013351432e-10, -7.131689471862046e-06)
@ip  core3/inst_sw_S1_341/VDD { (0.0, 0.0) (1.000000013351432e-10, -7.293232556548901e-06)
@ip  core3/inst_sw_S2_361/VDD { (0.0, 0.0) (1.000000013351432e-10, -5.945327302470105e-06)
@ip  core3/inst_sw_S2_52/VDD { (0.0, 0.0) (1.000000013351432e-10, -3.896009275194956e-06) (
@ip  core3/inst_sw_S2_171/VDD { (0.0, 0.0) (1.000000013351432e-10, -3.216043069187435 3e-06)
@ip  core3/inst_sw_S1_321/VDD { (0.0, 0.0) (1.000000013351432e-10, -7.787599315633997e-06)
@ip  core3/inst_sw_S1_1310/VDD { (0.0, 0.0) (1.000000013351432e-10, -6.743534413544694e-06)
```

- Current LSO for each power gate

External ScenarioView and AnalysisView : Syntax

Step 1: Create a pointer to the characterized currents file

```
current_source_files = [<block_name> + '.power_gate_currents’]
```

Step 2: Import these into an ‘External’ ScenarioView

```
scn_ru = db.create_scenario_view(design_view = dv, current_source_files =
current_source_files, scenario_type = ‘External’, tag = ‘scn_ru’, voltage_levels =
voltage_levels, options = options)
```

Step 3: Create an AnalysisView using this ScenarioView

```
av_ru = db.create_analysis_view(..., scenario_views = scn_ru, off_state_nets =
[Net(‘<block_internal_net>’)], keep_stats_level = ‘Full’, time_step = <t_step_ru>,
duration = <t_sim_ru>)
```

Aways-On Activity & AnalysisComboView : Syntax

Step 1: Create a ScenarioView and AnalysisView to use as the base view

- scn_always_on > A RHSC scenario of any type (regular, NPV etc.) with activity on always-on instances, macros etc. May have a mix of vectorless and vector-based activity.

```
av_always_on = db.create_analysis_view(..., scenario_views = scn_always_on, off_state_nets
= [Net('<block_internal_net>')], keep_stats_level = 'Full', time_step = <t_step_ru>,
duration = <t_sim_ru>)
```

Step 2: Create the AnalysisComboView using the above AnalysisView

- The arguments for create_analysis_combo_view are:
  1. The base AnalysisView object (av_always_on)
  2. A list of rampup AnalysisViews ([av_ru])
  3. A list of offsets for each rampup AnalysisView ([10.0e-9])
  4. A tag representing the view name

```
av_combo = db.create_analysis_combo_view(av_always_on, [av_ru], [10.0e-09], tag =
'av_combo_1')
```

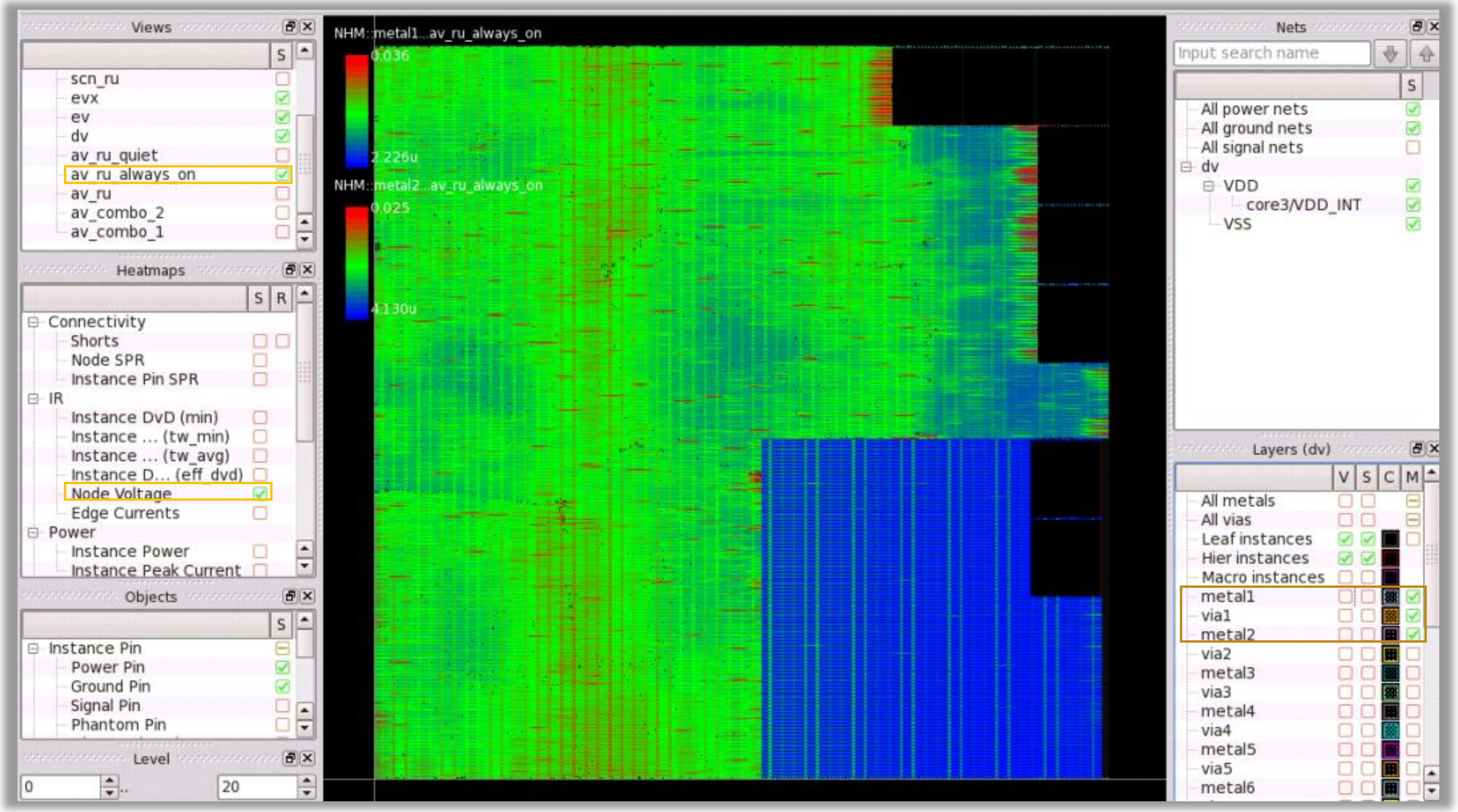
Rampup-specific Reporting

- Heatmaps, IR drop impact, Package effects
  - Requires AnalysisComboView (or external ScenarioView/AnalysisView)

- In-built APIs for many common in-rush related plots and metrics
  - Off-state voltage
  - In-rush currents (per switch and total)
  - Rampup voltage waveforms
  - Switch turn-on times
  - All available from characterization ScenarioView and AnalysisView

- Scripts for differential voltage report, worst rampup switch analysis

Always-On Node Voltage heatmap



Views

	S
scn_ru	☐
evx	✓
ev	✓
dv	✓
av_ru_quiet	☐
av_ru_always_on	✓
av_ru	☐
av_combo_2	☐
av_combo_1	☐

Heatmaps

	S	R
Connectivity		
Shorts	☐	☐
Node SPR	☐	
Instance Pin SPR	☐	
IR		
Instance DvD (min)	☐	
Instance ... (tw_min)	☐	
Instance ... (tw_avg)	☐	
Instance D... (eff_dvd)	☐	
Node Voltage	✓	
Edge Currents	☐	
Power		
Instance Power	☐	
Instance Peak Current	☐	

Objects

	S
Instance Pin	☐
Power Pin	✓
Ground Pin	✓
Signal Pin	☐
Phantom Pin	☐

Level: 0 .. 20

Nets

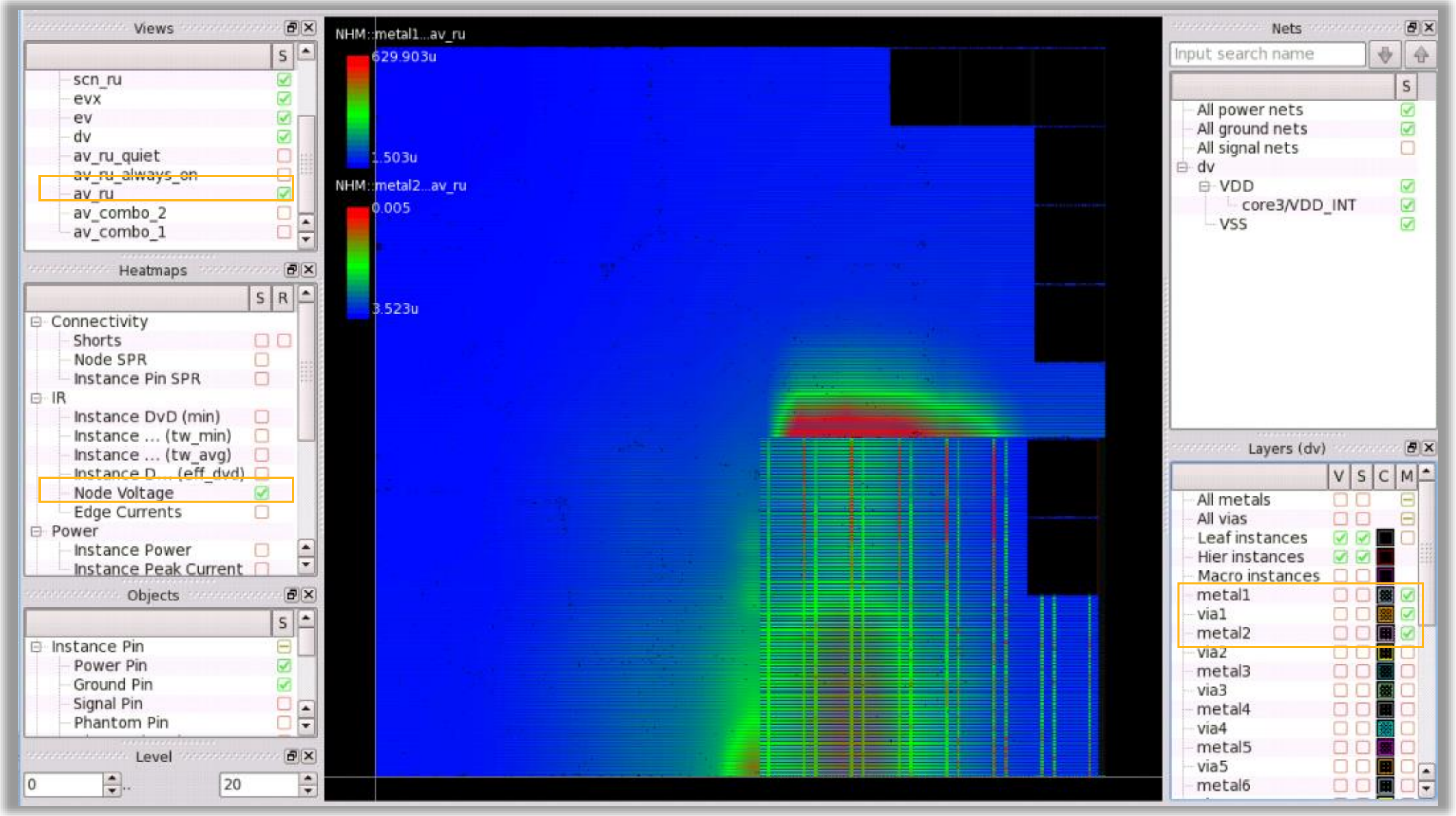
Input search name

	S
All power nets	✓
All ground nets	✓
All signal nets	☐
dv	
VDD	✓
core3/VDD_INT	✓
VSS	✓

Layers (dv)

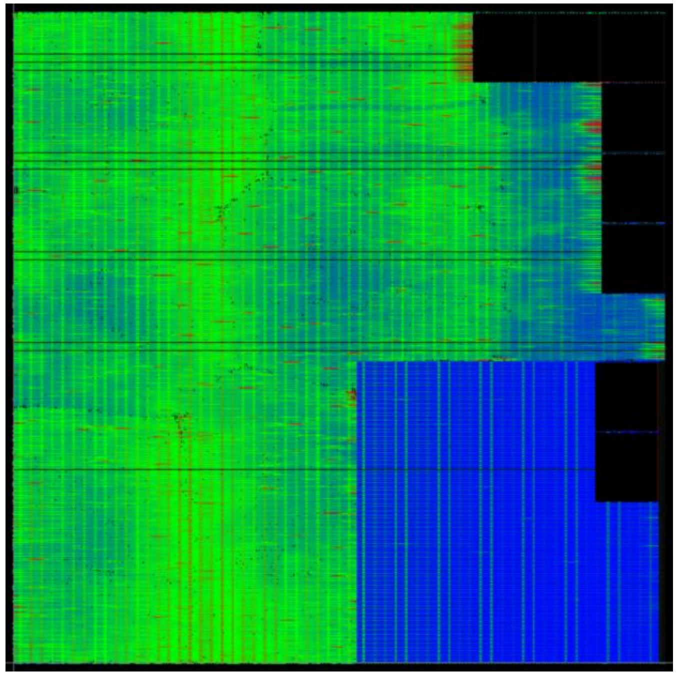
	V	S	C	M
All metals	☐	☐		☐
All vias	☐	☐		☐
Leaf instances	✓	✓	■	☐
Hier instances	✓	✓	■	
Macro instances	☐	☐	■	
metal1	☐	☐	▨	✓
via1	☐	☐	▨	✓
metal2	☐	☐	▨	✓
via2	☐	☐	▨	☐
metal3	☐	☐	▨	☐
via3	☐	☐	▨	☐
metal4	☐	☐	▨	☐
via4	☐	☐	▨	☐
metal5	☐	☐	▨	☐
via5	☐	☐	▨	☐
metal6	☐	☐	▨	☐

Rampup AnalysisView Node voltage heatmap

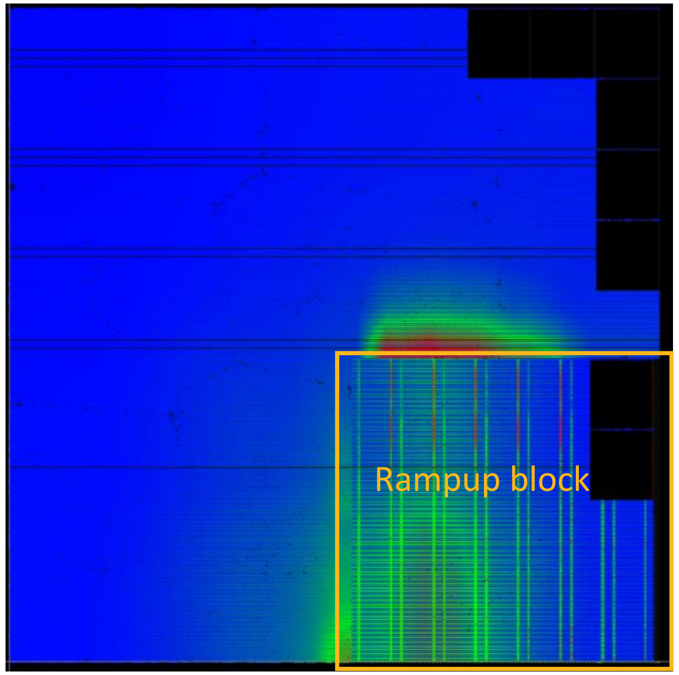


AnalysisCombo View Heatmaps

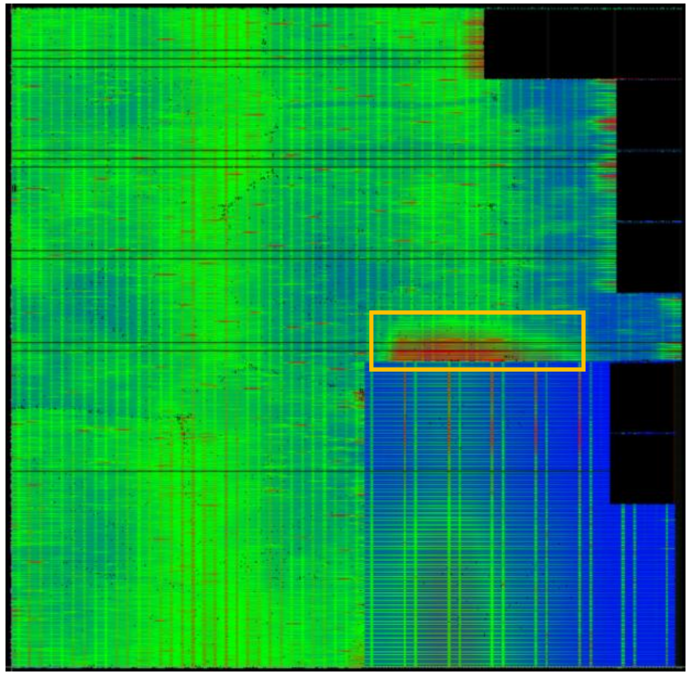
From Always-On AnalysisView



From Rampup AnalysisView



From AnalysisComboView



- AnalysisCombo view generated using the principal of superposition .

Rampup Heatmaps Code Snippet

- Generating the rampup time heatmaps

Option 1: 10% to 90% transition heatmap

```
>>> from thpkgs.ae_utils import rampup_get_worst_node

>>> heatmaps_ru = rampup_get_worst_node.get_rampup_node_heatmap(

        av         = av_char,

        domains    = [Net('<internal_net>')])

>>> gui.add_layer(heatmaps_ru[Net('<internal_net>')])
```

Option 2: Absolute Switch Turn-on Time heatmap

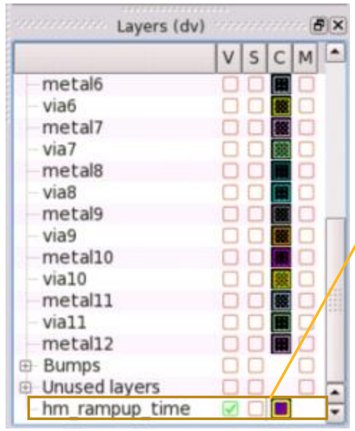
```
>>> from thpkgs.ae_utils import rampup_get_worst_node
>>> heatmaps_abs = rampup_get_worst_node.get_rampup_node_heatmap(
        av                     = av_char,
        domains                = [Net('<internal_net>')],
        from_zero              = True,
        check_only_power_gates = True)

>>> gui.add_layer(heatmaps_abs[Net('<internal_net>')])
```

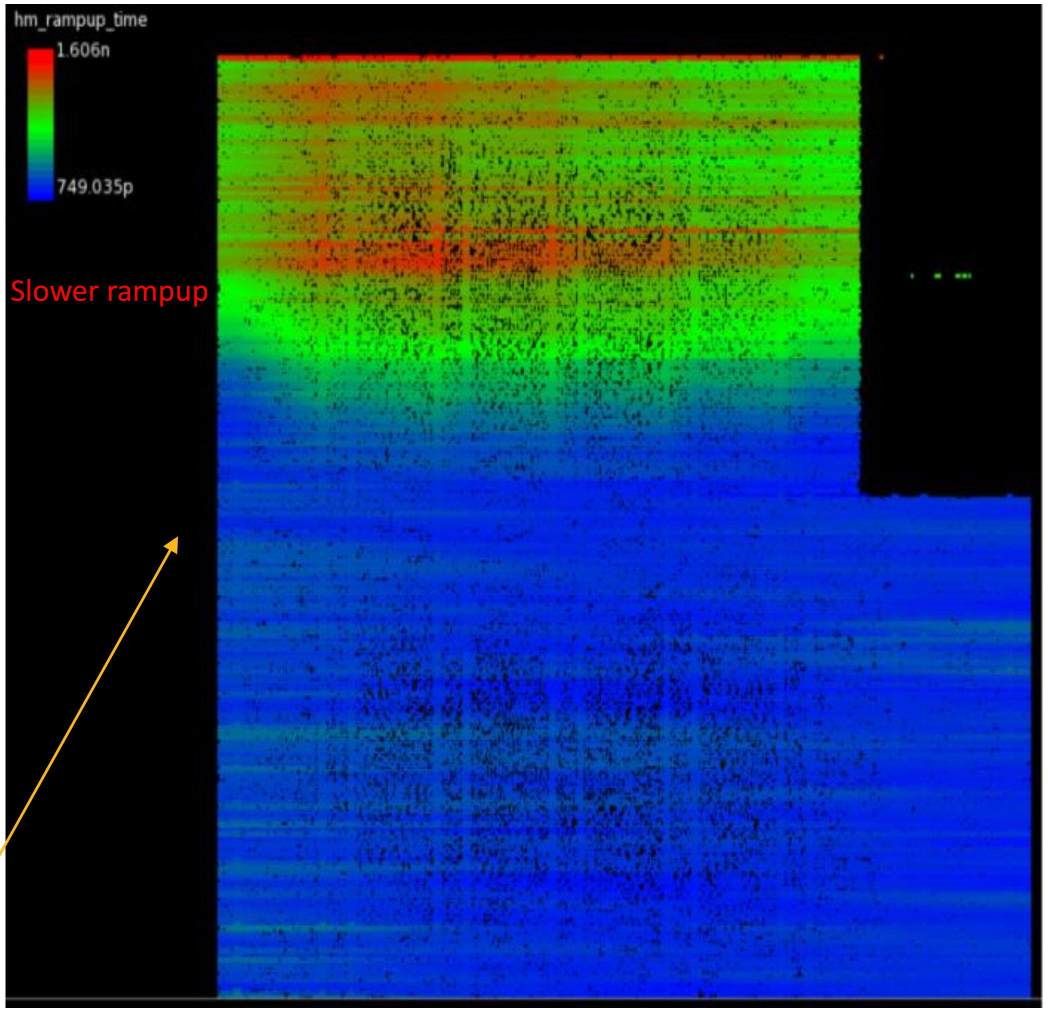
Rampup Time Checks Using Heatmaps

- Rampup Time -> Time for internal pin voltage to change from 10% to 90%
  - Adjusted for off-state voltage

```
>>> from thpkgs.ae_utils import rampup_get_worst_node
>>> heatmaps_ru =
rampup_get_worst_node.get_rampup_node_heatmap(
        av        = av_ru_quiet,
        domains   = [Net('core3/VDD_INT')])
>>> gui.add_layer(heatmaps_ru[Net('core3/VDD_INT')])
```

- Red regions have longer rampup times
  - Switch distribution issue?
  - Grid weakness?
  - Differential Voltage problems?



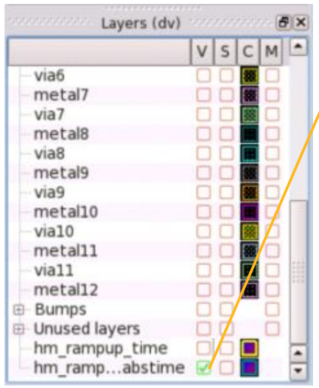


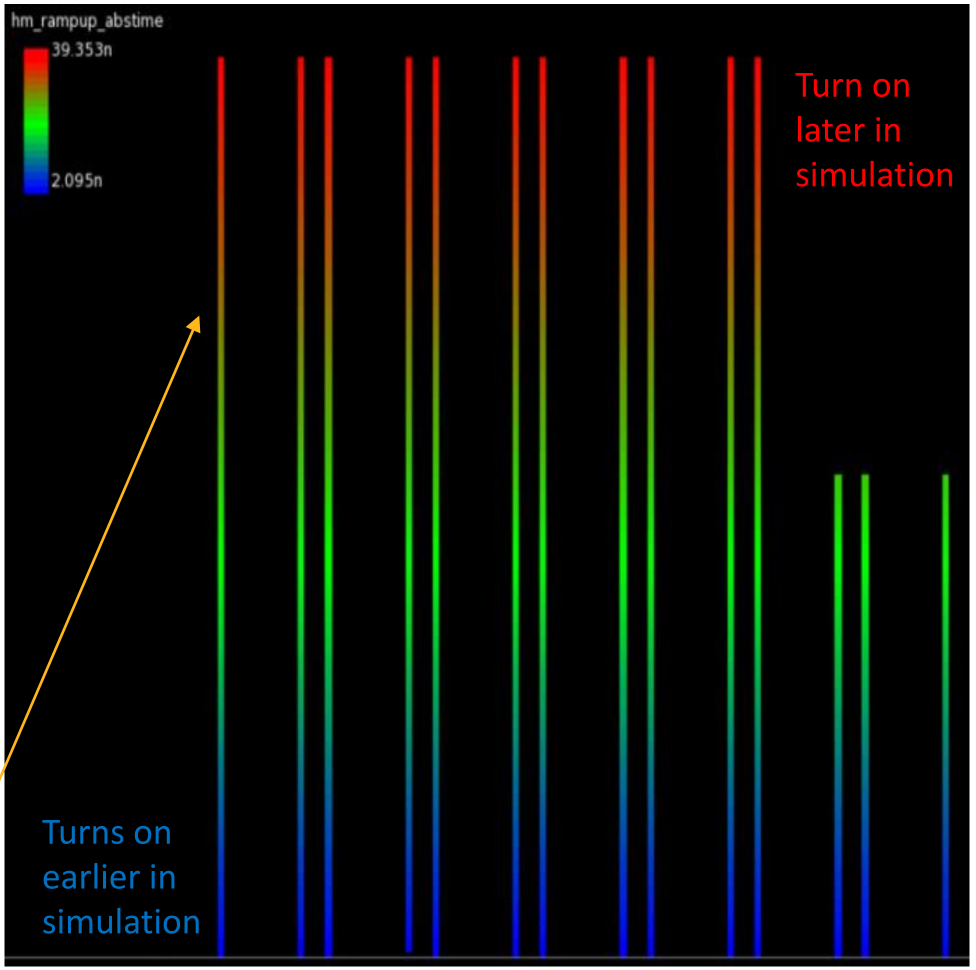
Absolute switch turn-on time Heatmaps

- Metric used: Time at which internal pin voltage reaches 90% of nominal

```
>>> from thpkgs.ae_utils import rampup_get_worst_node
>>> heatmaps_abs =
rampup_get_worst_node.get_rampup_node_heatmap(
        av                      = av_ru_quiet,
        domains                 = [Net('core3/VDD_INT')],
        from_zero               = True,
        check_only_power_gates = True)

>>> gui.add_layer(heatmaps_abs[Net('core3/VDD_INT')])
```

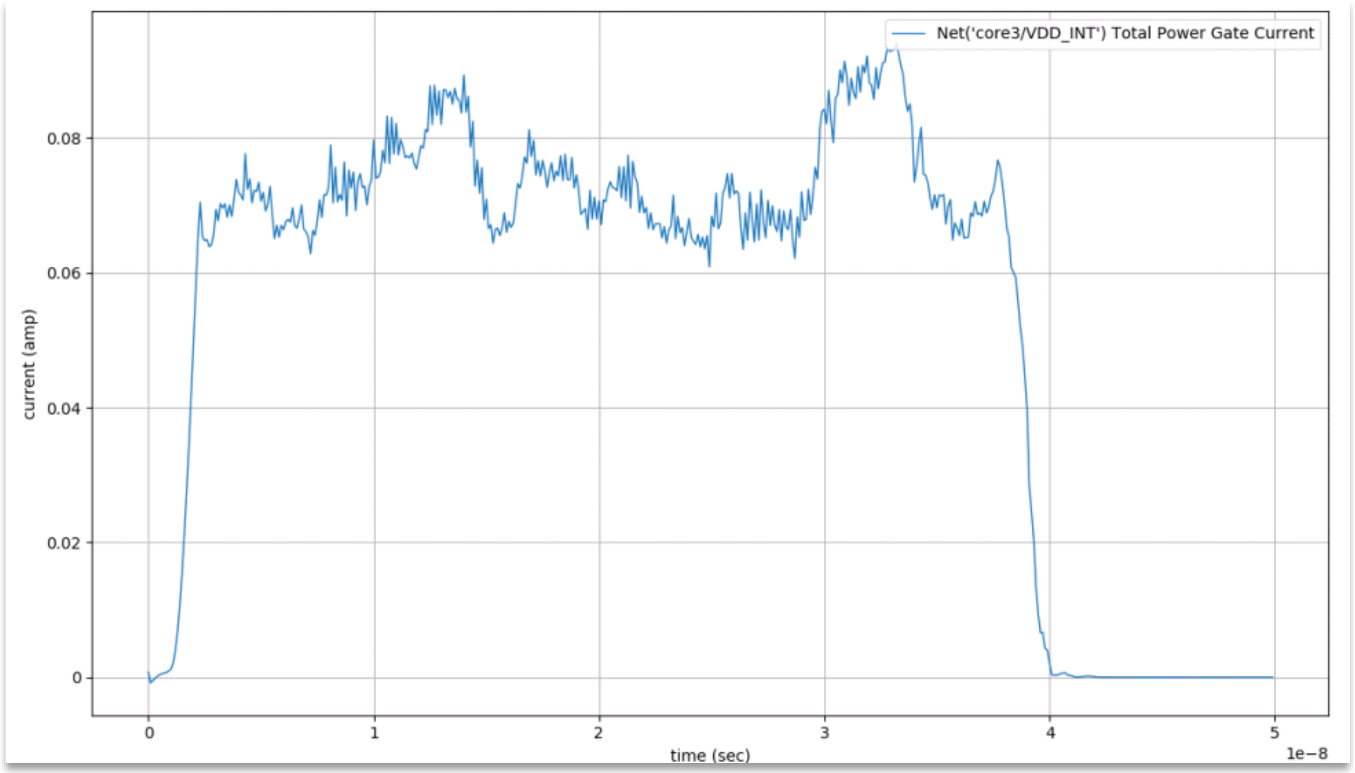
- Verify accuracy of switch control signal propagation
- Identify switches that do not ramp up





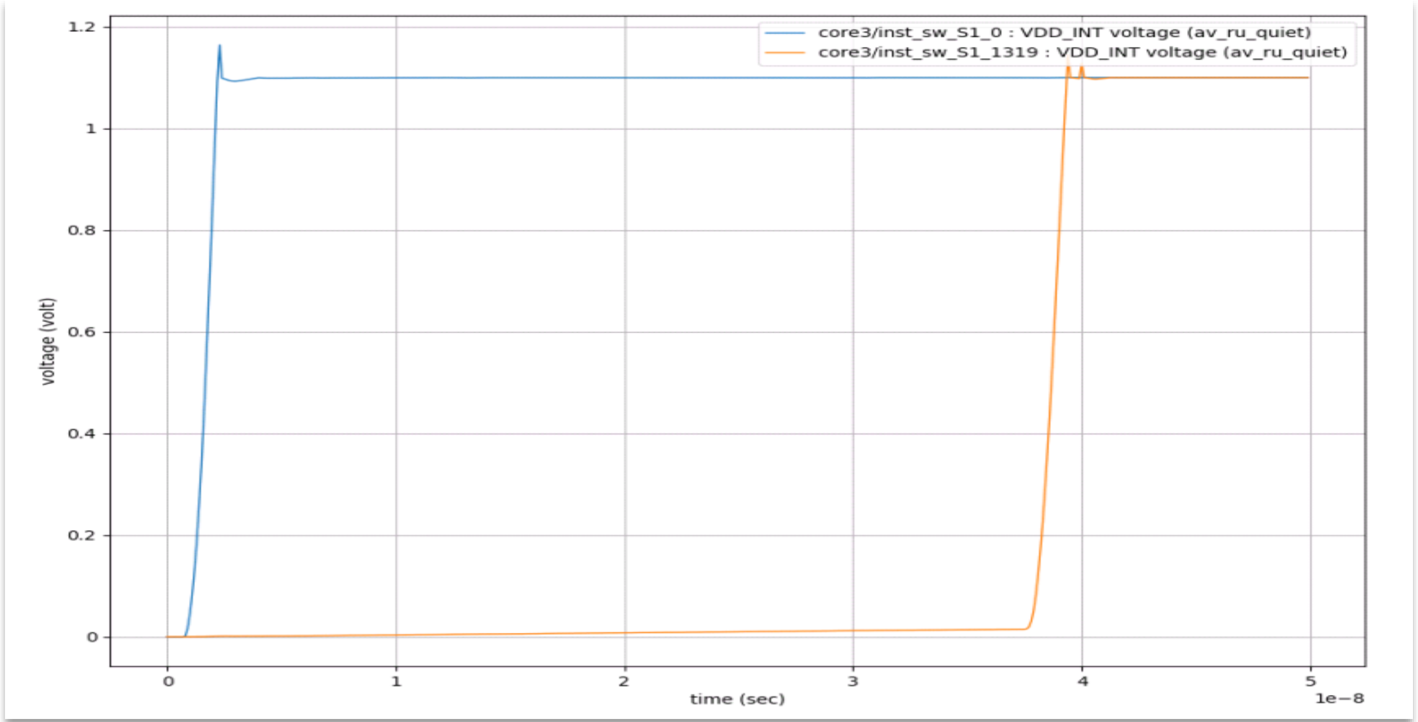
In-Rush Currents Per Domain

```
>>> av_ru_quiet.get_total_power_gate_currents().keys()
[Net('core3/VDD_INT’)]
>>> plot(av_ru_quiet.get_total_power_gate_currents()[Net('core3/VDD_INT')])
```



Rampup Voltage

```
av_ru_quiet.get_voltage(sw_inst, Pin('<internal_pin>'))
>>>first_sw_wfm = av_ru_quiet.get_voltage(Instance('core3/inst_sw_S1_0') , Pin('VDD_INT'))
>>>last_sw_wfm = av_ru_quiet.get_voltage(Instance('core3/inst_sw_S1_1319'),Pin('VDD_INT'))
>>>plot([first_sw_wfm , last_sw_wfm])
```

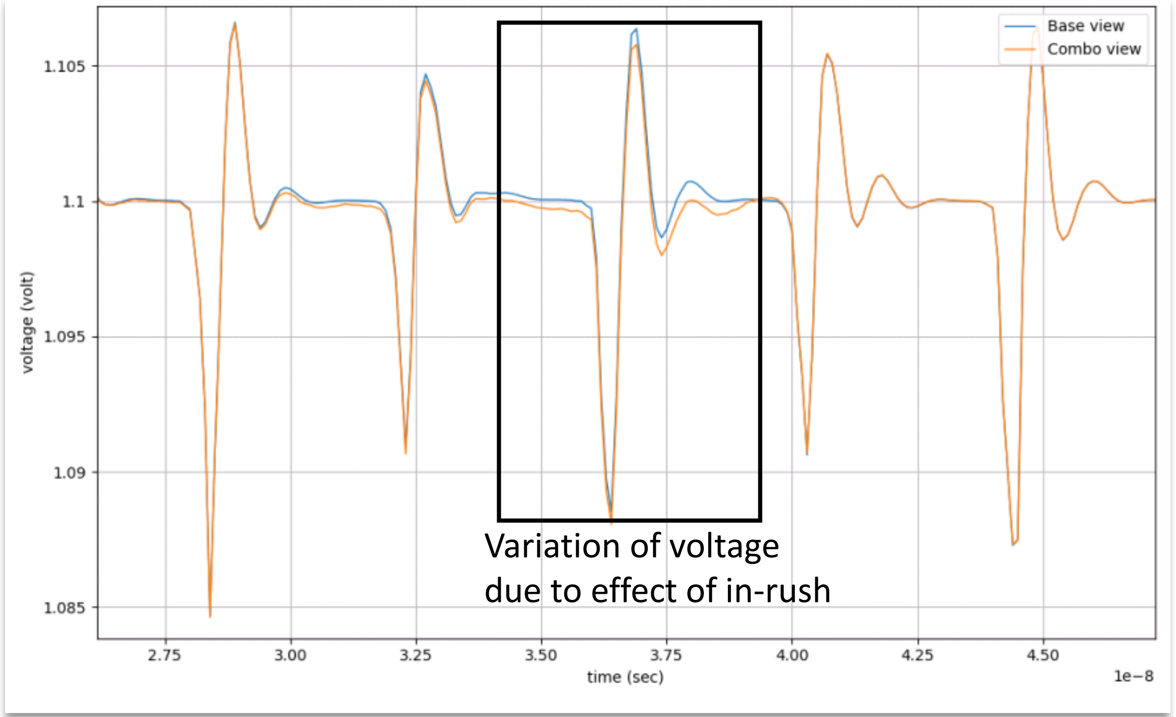


Effects on Neighbouring Instances

near_inst is an instance on the always-on domain close to the rampup block

- `av_always_on.get_voltage(near_inst, Pin(<supply_pin>))`
- `av_combo.get_voltage(near_inst, Pin(<supply_pin>))`

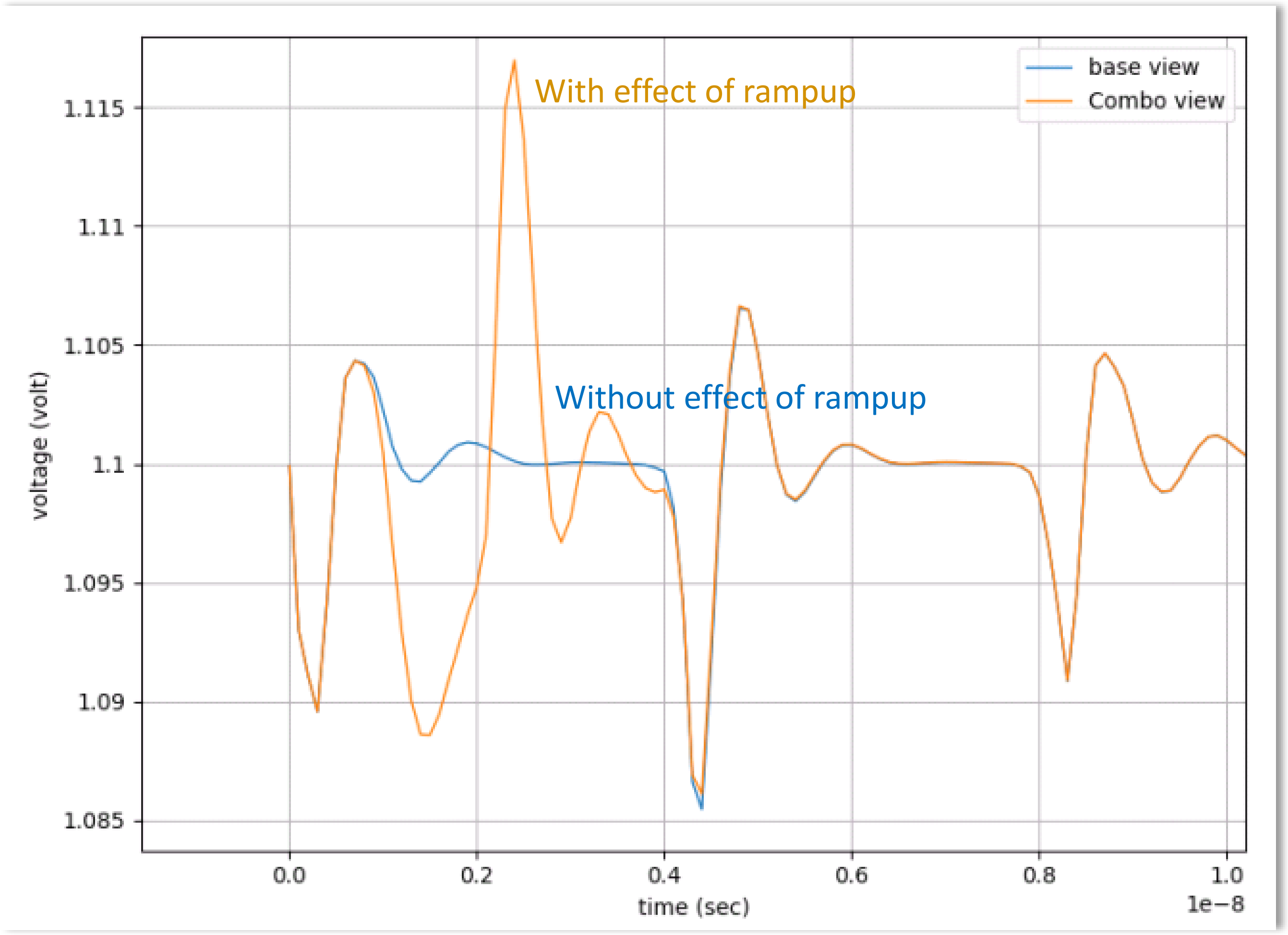
```
>>> base_wfm = av_ru_always_on.get_voltage(Instance("core0.regfile_data_memory.DFF_X1_740"),Pin('VDD’))
>>> combo_wfm = av_combo_1.get_voltage(Instance("core0.regfile_data_memory.DFF_X1_740"),Pin('VDD’))
>>> plot([base_wfm,combo_wfm] , labels = ['Base view' , 'Combo view'])
```



Effect of Changing the LSO

- Adjust the LSO timings to create different rampup scenarios
  - Possible to directly provide LSO per switch

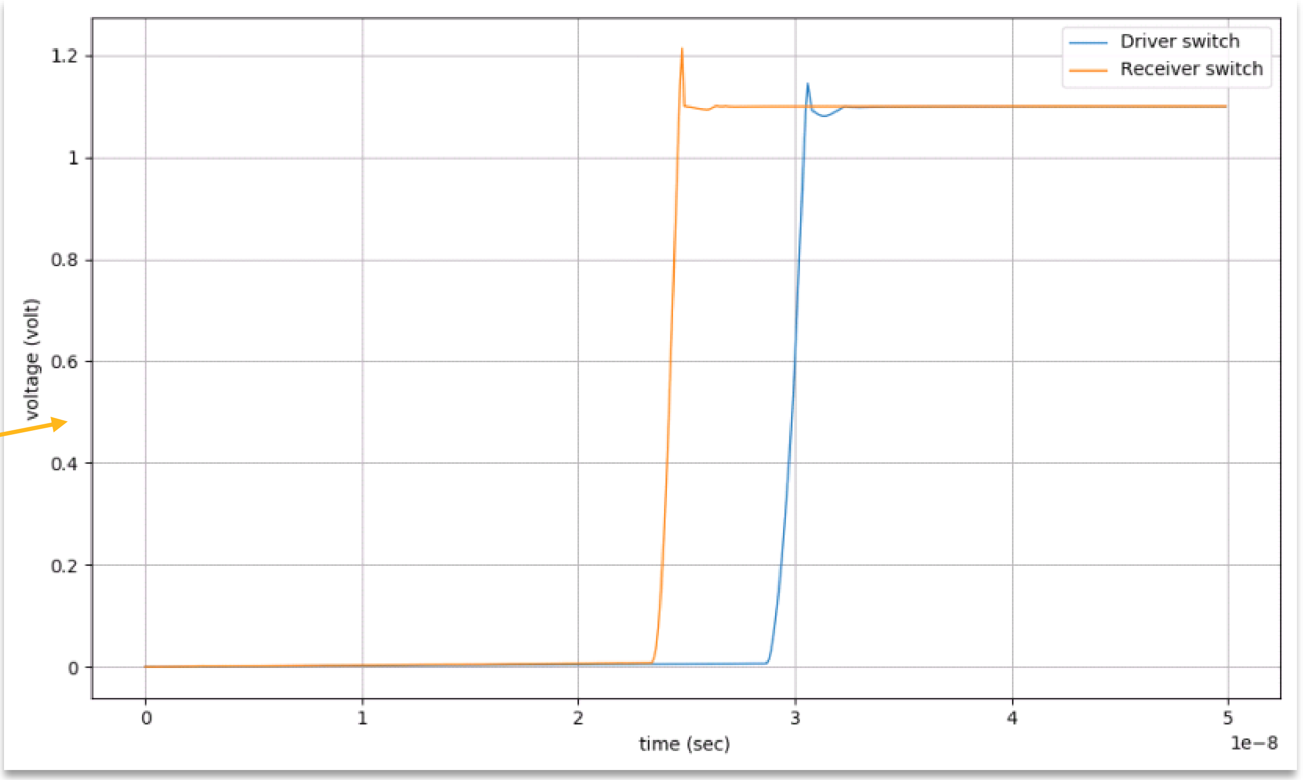
- What happens if
  - All my switches are turned on at once? (shown here)
  - Per switch delay is modified?
  - A few switches are not turned on?



Differential Voltage Report

- Find regions where receiver ramps up before driver
  - Crowbar currents

```
# Differential Voltage Report for Net('core3/VDD_INT')
#
# <DriverSwitch> <ReceiverSwitch> <DifferentialVoltage>
# - <DriverInst1> <ReceiverInst1>
# - <DriverInst2> <ReceiverInst2>

core3/inst_sw_S2_929 core3/inst_sw_S2_754 1.208
 - core3/HFSINV_792_1928
core3/regfile_program_memory.MUX2_X1_3678
 - core3/HFSINV_792_1928
core3/regfile_program_memory.MUX2_X1_3680
 - core3/HFSINV_846_2579
core3/regfile_program_memory.MUX2_X1_2693

core3/inst_sw_S1_977 core3/inst_sw_S2_754 1.207
 - core3/ZBUF_2_inst_25980 core3/ZBUF_17_inst_19280
```



Differential Voltage Report Code Snippet

- Use help(rampup_get_differential_report.dump_differential_voltage_report) for a list of all arguments

```
rampup_get_differential_report.dump_differential_voltage_report(
    filename     = '<net_name>_differential_voltage.rpt',
    av           = av_char,
    internal_net = Net('<internal_net>'))
```

- Generating the differential voltage report

```
>>> from thpkgs.ae_utils import rampup_get_differential_report

>>> rampup_get_differential_report.dump_differential_voltage_report(
        filename     = 'VDD_INT_differential_voltage.rpt',
        av           = av_ru_quiet,
        internal_net = Net('core3/VDD_INT'))
```

Sanity Checks for Rampup

- Verify the APLSW model using IV curves

- Verify switch control signal propagation
  - Sense of the LSO
  - Check logic signal at random switches
  - Check switch turn-on heatmap for white spots

- Verify accurate off-state voltage calculation
  - Leakage comes from pwcap
  - Find voltage in off-state IV curve to meet leakage

Sanity Checks: Offstate Voltage Calculation Code Snippet

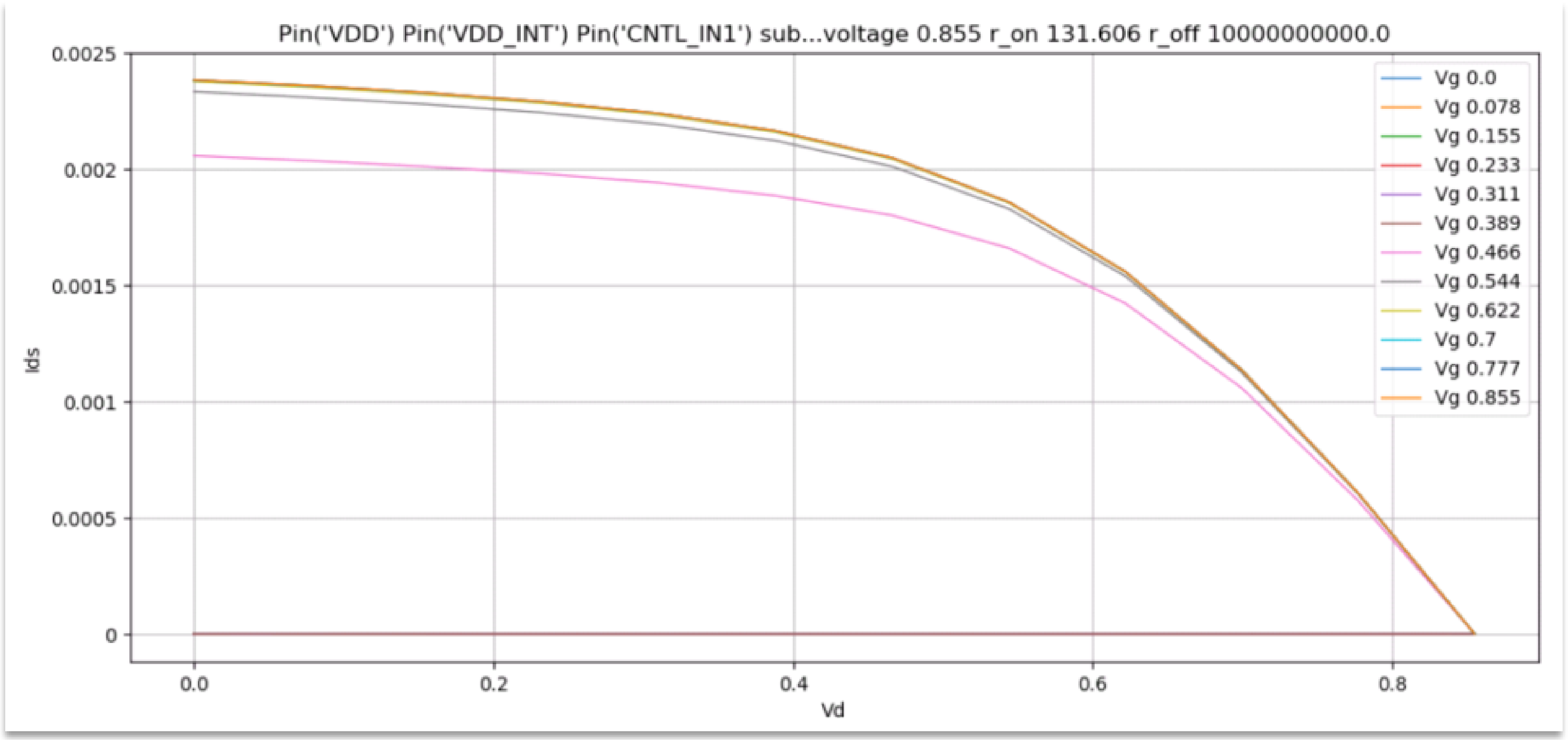
```
av_char.get_quiet_state(Net('<internal_net>')]))
  # Returns a dict with the following keys
  # 'status'                : 0 if successful
  # 'dc_state'              : calculated off state voltage
  # 'leakage'               : leakage current used for calculation
```

```
>>> av_ru_quiet.get_quiet_state(Net('core3/VDD_INT'))

{'status': 0, 'dc_state': 0.0, 'leakage': 0}
```

Checking offstate voltage

Sanity Checks: IV Curve Visualization Code Snippet

```
>>> from design_utils import get_power_gate_iv_curves
>>> get_power_gate_iv_curves(<DesignView>, Cell(<switch_cell_name>))
```



Thank You

Ansys