# Black-Boxing Techniques for Improving VC-LP Throughput

## (Enhanced Black-box Flow)

Parag Mandrekar
Advanced Micro Devices, Inc. (AMD)

September 29, 2016
Austin, TX

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

Propagating Design Info Via SPA (**`set_port_attribute`**)

Example runs and future enhancements to VC-LP

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

Propagating Design Info Via SPA (`set_port_attribute`)

Example runs and future enhancements to VC-LP

# Introduction
## Background

- VC Static-LP (VC-LP) – Synopsys' next generation Static Power Rule Checking (PRC) tool
  - Replaces Synopsys' MVRC – up to 10x run-time, 2x capacity improvement
  - Multi-voltage, power-gated designs
    - Checks design + UPF for "**power correctness**" – looks for protection (electrical) violations + functional violations
  - Common tcl-shell based interface, rich set of API commands
  - Intuitive Violation Viewer GUI, cross-ref with Schematic, Source, UPF Viewer
  - Separate, de-coupled Build, Check, Report steps DB checkpoint/save and future restore
  - Enables incremental checking and reporting without re-building DB
  - Violation checks – commands to skip and configure checks
  - Report generation - reduce  similar violations to a single representative violation, filter/waive violations to report, by tag/family/severity/regexp

# Introduction (Continued)

AMD

snug
Synopsys Users Group
Austin

## Functional intent specifies

- **Architecture**
  - Design hierarchy
  - Data path
  - Custom blocks
- **Application**
  - State machines
  - Combinatorial logic
  - I/Os
  - EX: CPU, DSP, Cache
- **Usage of IP**
  - Industry-standard interfaces
  - Memories
  - etc

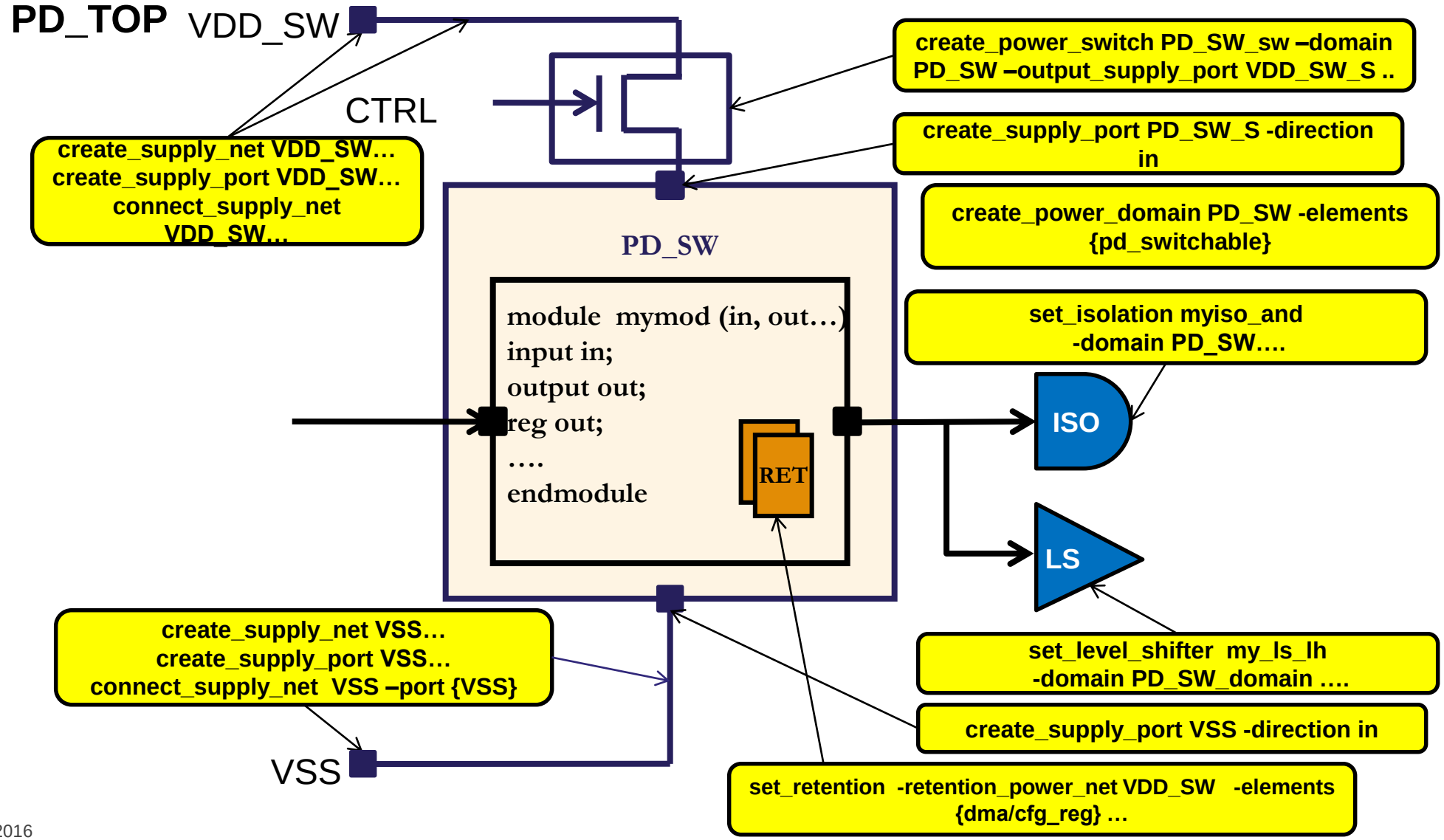
**RTL, netlist**

## Power intent specifies

- **Power distribution architecture**
  - Power domains `create_power_domain`
  - Supply rails `create_supply_net, create_supply_port`
- **Power strategy**
  - Power state tables `create_pst, add_pst_state`
  - Operating voltages `add_port_state VON -state {VDDON 1.0} -state {VDDOFF off}`
- **Usage of special cells**
  - Isolation cells, Level shifters `set_isolation, set_level_shifter`
  - Power switches `create_power_switch`
  - Retention registers `set_retention`

**UPF**

# Introduction (Continued)



# Introduction (Continued)

- VC-LP Static Power Rule Checking (PRC) usage at various stages of the design

  – RTL PRC :
    - Usually RTL has no instantiated protection cells
    - UPF consistency – LS (Level-Shifter) / ISO (Isolation) strategy presence or absence, correctness

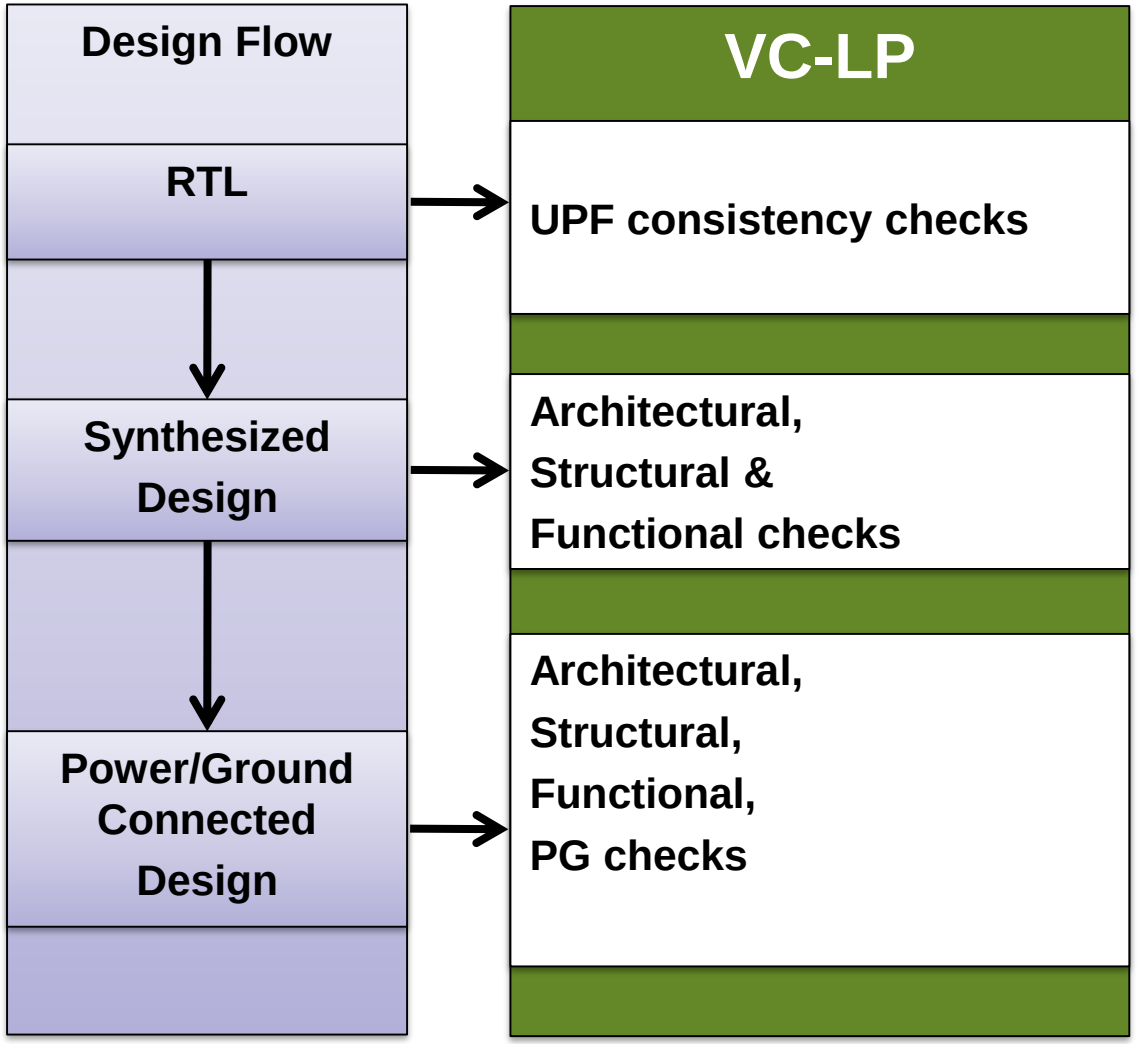
  – Netlist PRC :
    - After Logic Synthesis

      Resulting netlist has ISO, LS, PSW (Power Switch) and other cells instantiated

      VC-LP used to additionally perform netlist-side checks, to ensure it conforms to the power supply network and strategies specified in the UPF.
    - After Place and Route

      VC-LP can be used to additionally check the supply hookup in the resulting physical netlist, along with other consistency checks

# Introduction (Continued)




Checks design vs. power intent

Validate multi-voltage designs

TCL interface allows report customization

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

Propagating Design Info Via SPA (`set_port_attribute`)

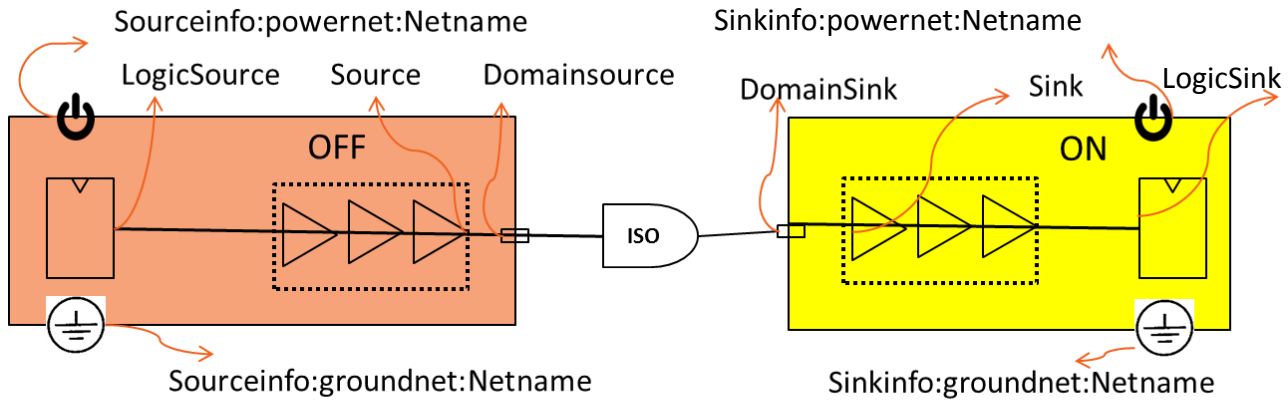
Example runs and future enhancements to VC-LP

# Hierarchical vs Black-box runs (Cont'd)

## VC-LP Crossover Analysis

- VC-LP analyzes "crossover" paths
  - Crossover – any path that contains a domain boundary
  - Start at a **LogicSource**, include buffers/inverters, protection cells, intermediate domain boundaries
  - End at **LogicSink**
  - Crossovers may span > 1 block
  - Exposing entire hierarchy allows VC-LP to see the complete crossover
    - More accurate analysis
    - Exposing all blocks' hierarchy means more logic to analyze – longer runs

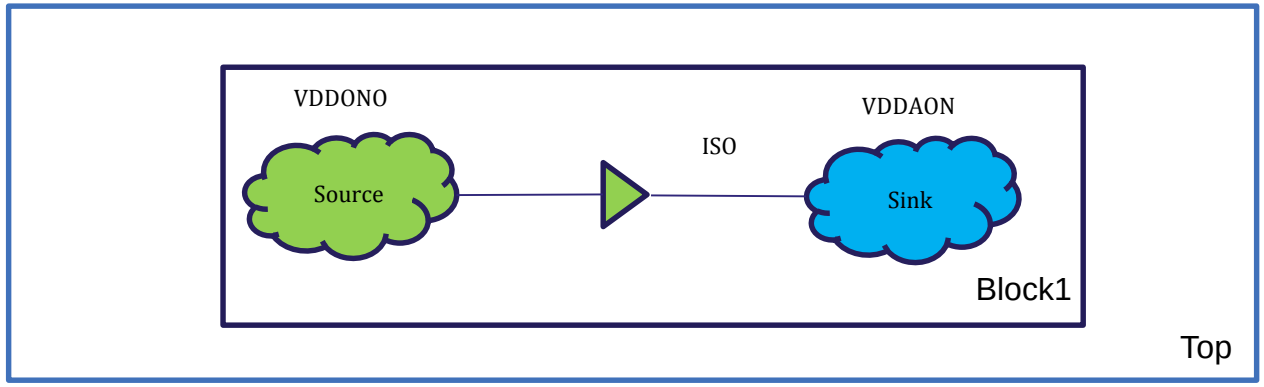
## Crossover Fields Terminology




| Field | Details |
|---|---|
| LogicSource | Pin on the first non-buffer(AOB/single rail),inv,ISO/LS cell driving the PD boundary port |
| Source | Pin on the first cell driving the PD boundary port |
| DomainSource | Driver Port on the Domain boundary cell |
| LogicSink | Pin on the first non-buffer(AOB/single rail),inv,ISO/LS driven by the PD boundary port |
| Sink | Pin of the first cell driven by the boundary port |
| DomainSink | Driven port on the Domain boundary cell |

# Hierarchical vs Black-box runs

- Block-level runs
  - VC-LP can be run with full hierarchy exposed (Hierarchical Mode)
    - Fast turnaround time, complete violation reports at block-level




ISO_INST_MISSING

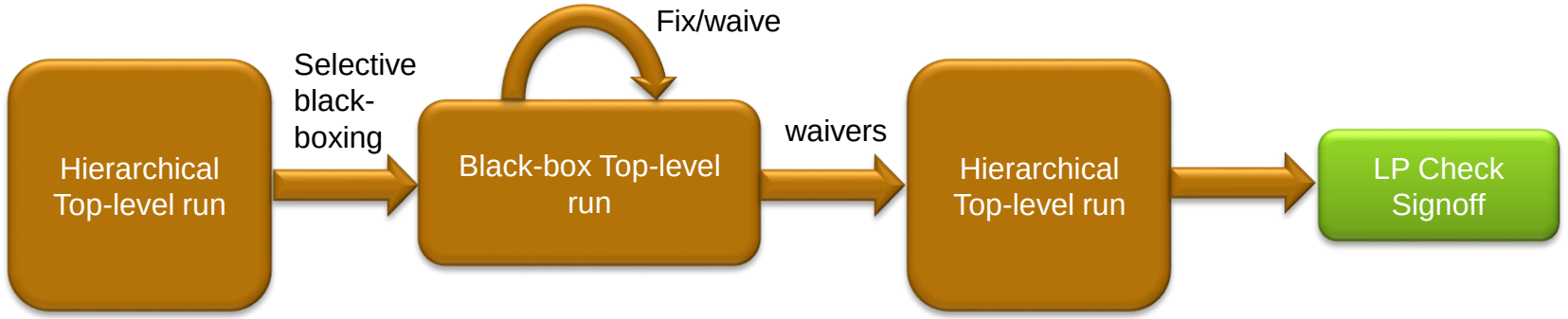
    - All block-level runs that have the complete design information can be done in parallel
    - Violations with LogicSource and LogicSink inside block need to be checked and fixed/waived

# Hierarchical vs Black-box runs (Cont'd)

- Top-level runs
  - VC-LP can be run with full hierarchy exposed (Hierarchical Mode)
    - More accurate results, longer run times
    - If design size is large, limited number of runs can be performed in a design cycle
  - Black-boxing is an effective strategy
    - Blocks already verified at block-level and not affecting other blocks can be black-boxed (avoid re-analyzing inside the block, save run-time)
    - Blocks not completed (upf, netlist not ready) for the Top-level run can be black-boxed
    - Any block or component (containing blocks) that have fully protected inputs and fully isolated outputs can be black-boxed
    - Also useful in verifying power-ground (PG) connectivity between blocks at Top-level
    - Enables quick iterations of Top-level runs and clean up of LP violations
    - Reduces run-time, but results may be inaccurate/incomplete for crossovers spanning blocks

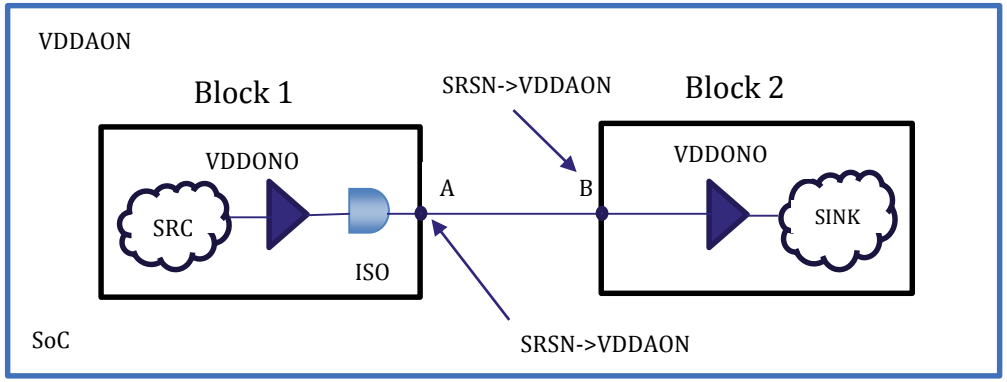
# Hierarchical vs Black-box runs (Cont'd)

– Example use-model – Top-level runs
  - Limited number of hierarchical runs possible for very large SoC designs
  - After 1st hierarchical run results have been analyzed, selective black-boxing techniques can be used to quickly iterate over fixing and waiving specific violations
  - If full hierarchical run's violation report indicates that most violations are concentrated to 1-2 blocks, and the violations indicate that the Source and Sink are localized to those 2 blocks (inter-block crossovers), the remaining blocks can be black-boxed for subsequent runs
  - Significant speedup of the VC-LP run can be achieved with this, resulting in faster time to signoff

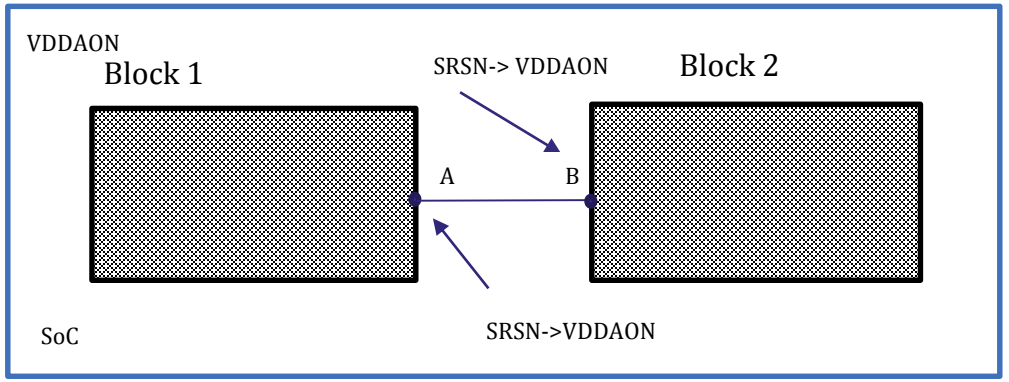


# Hierarchical vs Black-box runs (Cont'd)

- Limitations – Black-boxing at Top-level
  - Stub(s) have to be created for block netlists – VC-LP always reads the entire module logic, then uses only the interface definition
  - Block UPF – constructs, objects, and strategies inside hierarchy may not be applicable
    - VC-LP could ignore or report errors
  - Since entire hierarchy is not visible, full crossovers may not be generated
    - VC-LP could miss violations




**ISO_INST_REDUND (warning) reported in full hierarchical Top-level run**




**No violation reported at Top-level when the blocks are black-boxed**

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

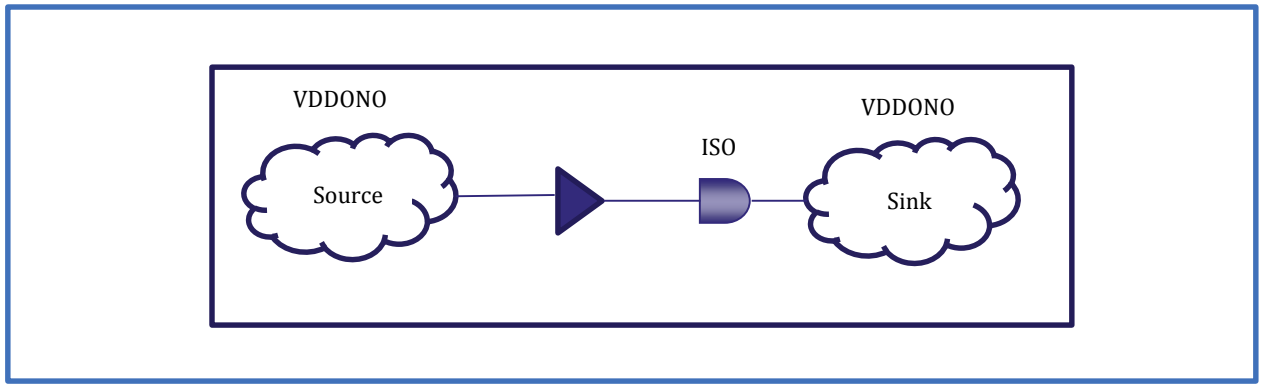
Propagating Design Info Via SPA (`set_port_attribute`)

Example runs and future enhancements to VC-LP

# Black-box Flow Steps

## Block-level runs

- VC-LP can read design/upf and analyze with quick turnaround time
- All block-level VC-LP runs with complete design info **can be done in parallel**
- Violations on crossovers with Source and Sink inside the block need to be checked and waived
  - These violations are not seen if block is black-boxed at Top-level
- Violations reported specific to design instances or protection devices should also be checked and fixed (e.g., ISO_CLAMP_INVERT error)

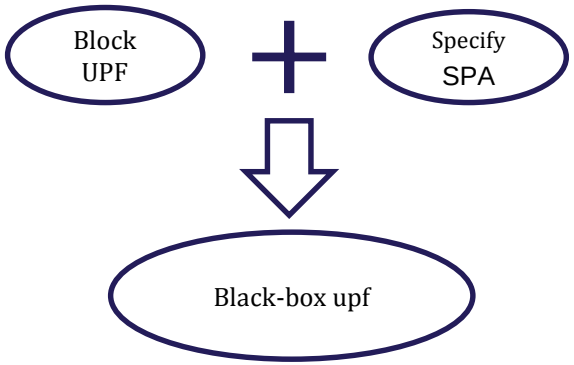



**ISO_INST_REDUND (warning) violation inside block**

# Black-box Flow Steps (Continued)

## Block-level runs

- VC-LP command `write_bbox_data_model` can be used to write relevant design information at the black-box ports, via `set_port_attribute` (SPA) statements
- The resulting **BBOX_ATT/bboxAttribute.upf** file can be concatenated with the block upf file and user at the Top-level




| Block-level verification steps | |
|---|---|
| **Description** | Commands Flow |
| **Read in block design netlist, UPF and verification setup (via tcl app_vars)** | `read_file -netlist …..`<br>`read_upf block.upf` |
| **Check violations** | `check_lp -stage upf`<br>`check_lp -stage design`<br>`check_lp -stage pg` |
| **Write violation reports** | `report_lp -verbose -file <rpt file>` |
| **Write out black-box SPA information in BBOX_ATT/bboxAttribute.upf** | **`write_bbox_data_model -path BBOX_ATT`** |
| **Save the database / quit** | `# Script to generate driver/receiver supply info`<br>`source`<br>**`$FLOWDIR/gen_drv_recv_spa.tcl`**<br>`save_session`<br>`quit -f` |

# Black-box Flow Steps (Continued)

## Top-level runs

- Create stubs for each block that is being black-boxed
  - Module definition and input/output/inout, etc. declarations
  - If the hierarchical block module is used, VC-LP reads the entire module and uses only the module interface
- Use the **`set_blackbox`** command, to black-box selected blocks
  - **`set_blackbox –cells “u1” # using instance name`**
  - **`set_blackbox –designs { block_stub_x block_stub_y } #using module name`**

```
set search_path …
set link_library "…”
set_app_var …
set_blackbox -designs { modX modY }
read_file -top soc -netlist { soc.v
modXStub.v modYStub.v … }
catch { read_upf -quietBbox soc.upf }
read_upf_error
if { $read_upf_error == 0 } {
  puts "ERROR : Problem(s) during Upf
parsing. Please check log file\n" ;
  quit -force
}
# No infer_source in bb flow
check_lp -stage upf
check_lp -stage design
check_lp -stage pg
report_lp -limit 0 -verbose -file
soc.rpt
quit
```

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

Propagating Design Info Via SPA (`set_port_attribute`)

Example runs and future enhancements to VC-LP

# Creating block.upf for use as black-box

- **`block.upf`** file originally intended for use for block-level VC-LP Low Power Checks
- Can be used **without modification** for the black-boxed block in Top-level flow
- Requirements of **`block.upf`**
  - Each power-aware block that is black-boxed must have its own UPF.
  - For each port of the black-box, both the **`driver_supply`** & **`receiver_supply`** should be defined.
  - The following information must be present in the block UPF:
    - Top **power domain** of the block
    - **Supplies** for the top power domain of the block
    - Power State Table (**PST**) based on only the supplies defined in the top power domain. Any supplies referenced inside the block hierarchy are ignored when parsing the PST
    - Any design references, strategies, and **`connect_supply_net`** statements inside the black-box hierarchy are ignored by VC-LP.

# Creating block.upf for use as black-box

- VC-LP only processes objects that exist at the black-box boundary, it ignores objects that do not exist inside the black-box
  - e.g., **`create_power_domain bb_pd -include_scope -element {sub/u1}`**
  - sub/u1 inside a black-boxed block is ignored

- Error messages related to objects not found inside black-box are suppressed

- Applies to
  - **`create_power_domain, create_supply_port, create_supply_net, create_supply_set , connect_supply_net, associate_supply_set`**
  - **`set_port_attributes, set_related_supply_net, create_power_switch, add_power_state, add_pst_state, add_port_state, create_pst, set_isolation, set_isolation_control, map_isolation, set_level_shifter`**

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

Propagating Design Info Via SPA (**`set_port_attribute`**)

Example runs and future enhancements to VC-LP

# Propagating Design Info Via SPA's

## `set_port_attribute`

- When a block is black-boxed, inter-block crossover information is lost
- SPA's provide a way to propagate and add information regarding the internal logic, to the ports of the block
  - Driver/receiver supplies, tied inputs, unconnected inputs and outputs, iso_enable ports, ports isolated/protected inside the blocks, feedthrough ports
  - **Adds useful information for VC-LP** to analyze crossovers that reference those block ports
  - Using SPA information, VC-LP can identify **missing or redundant ISO/LS**, iso_en source and polarity, tied input conditions and other LP related issues
  - Less fake violations and false negatives
  - On our Top-level designs analyzed by VC-LP that use this enhanced SPA information during black-boxing, up to **85% violations were correctly identified** compared to fully hierarchical VC-LP runs

# SPA's derived at block-level

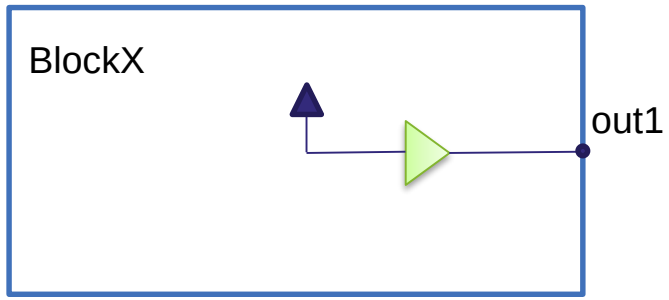
- VC-LP derives and propagates design information from the logic inside the block to the ports of the block (**`write_bbox_data_model`**)

- Ports driven by constant 0/1 (Tie-Low/Tie-High, 1`b0/1`b1, supply0/supply1)
  - Syntax

```
set_port_attribute -model <bbox_name> -ports {<bbox_port_name>}
-attribute SNPS_BLK_MODEL_TIED_VALUE {<1/0>}
-attribute SNPS_BLK_MODEL_TIED_NAME {hier_name_of_const}
```

  - Example

```
set_port_attributes -model core1 -ports {out1} \
-attribute SNPS_BLK_MODEL_TIED_VALUE {1} \
-attribute SNPS_BLK_MODEL_TIED_NAME {blackbox/LOGIC1}
```



# SPA's derived at block-level (Continued)

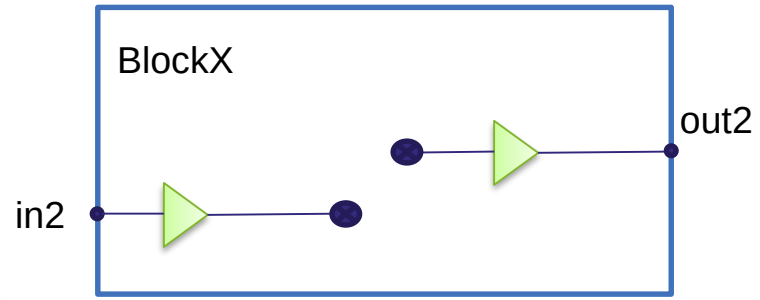
- VC-LP derives and propagates design information from the logic inside the block to the ports of the block (**`write_bbox_data_model`**)

- Ports not connected to any internal logic
  - If input port, it means there is no logic within the model driven by the port
  - If output port, it means there is no logic within the model driving the port
  - Syntax

```
set_port_attribute  -model <bbox_name> -ports {<bbox_output_port_name>}
-attribute SNPS_BLK_MODEL_UNCONNECTED {<true/false>}
-attribute SNPS_BLK_MODEL_UNCONNECTED_NAME {hier_name}
```

  - Example

```
set_port_attributes -model bbox_top -ports {out2} \
-attribute SNPS_BLK_MODEL_UNCONNECTED {true} \
-attribute SNPS_BLK_MODEL_UNCONNECTED_NAME {sInst22/out}
```



# SPA's derived at block-level (Continued)

- VC-LP derives and propagates design information from the logic inside the block to the ports of the block (**`write_bbox_data_model`**)
- Ports that are iso_enable inputs
  - Syntax

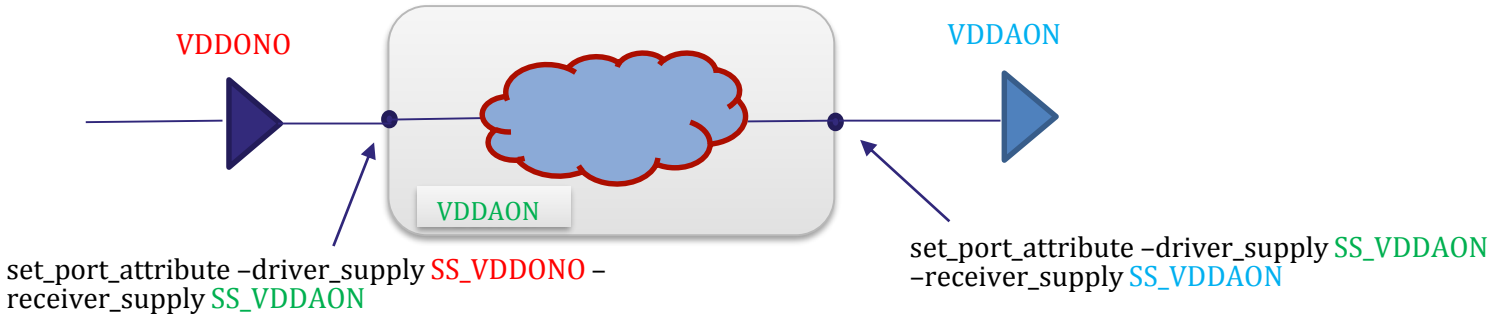
```
set_port_attributes -model <bbox_name> -ports {<bbox_port_name>}  \
-attribute SNPS_BLK_MODEL_DEVICE_PORT {ISOLATION device port name }   \
-attribute SNPS_BLK_MODEL_DEVICE_CLAMP {<device clamp value>} \
-attribute SNPS_BLK_MODEL_DEVICE_POLICY {<Iso policy at block>} \
-attribute SNPS_BLK_MODEL_POLICY_CLAMP {<Iso policy clamp value at block>}
```

  - Example

```
set_port_attributes -model bbox_top -ports {iso} \
-attribute SNPS_BLK_MODEL_DEVICE_PORT {iso6/B}   \
-attribute SNPS_BLK_MODEL_DEVICE_CLAMP {LOGIC_0} \
-attribute SNPS_BLK_MODEL_DEVICE_POLICY {PD_SUB/isoP2}
-attribute SNPS_BLK_MODEL_POLICY_CLAMP {LOGIC_0}
```

# SPA's derived at block-level (Continued)

- SPA's can specify both the external supply of the driving/driven logic of the port as well as the internal supply of logic that drives/is driven by the port, in the same statement
  - **`-driver_supply`** (block output port): the supply of the last (driving) cell connected to the output port
  - **`-receiver_supply`** (block input port): the supply of the first (driven) cell connected to the input port
- In this way, the block-level UPF can be used for block-level as well as Top-level Low Power Checks



# SPA's derived at block-level (Continued)

- VC-LP **API-based tcl script** used to generate driver/receiver supply information for each port
  - Black-Box Input Ports

```
set_port_attribute -port {list of all input ports} \
-driver_supply external_supply_set \
-receiver_supply internal_supply_set
```

  - Black-Box output Ports

```
set_port_attribute -port {list of all output ports}  \
-driver_supply internal_supply_set \
-receiver_supply external_supply_set
```

  - Example

```
set_port_attributes -port XYZ7_req[1] -driver_supply SS_PD1 -receiver_supply SS_GLB
set_port_attributes -port XYZ7_req[0] -driver_supply SS_PD1 -receiver_supply SS_GLB
set_port_attributes -port XYZ4_send -driver_supply SS_PD1 -receiver_supply SS_GLB
```

# Agenda

Introduction

Hierarchical vs black-box runs

Black-box flow steps

Black-box UPF creation and scope

Propagating Design Info Via SPA (`set_port_attribute`)

Example runs and future enhancements to VC-LP

# Example runs with Enhanced Black-box Methodology

- Used on 2 large Full-chip designs. Stubs for black-boxes, original upf, block SPA's used
- Violations at block-level analyzed and fixed separately
- Remaining inter-block violation counts close to hierarchical run
  - For Top-level (Full-Chip) Design #1:
    - Initial run **Full Hierarchical** – completed in **~26 hours**
    - Violations from Hierarchical run were analyzed and **only those blocks that did not have both Source and Sink in the same violation** were chosen to be black-boxed
    - **62.5%** of the blocks were black-boxed
    - Signal Corruption (Architectural) functional checks were NOT enabled for either the Full Hierarchical run or the black-box runs. This was due to the long crossovers of the scan_en, iso_enable, clk, reset and clk_en Control Roots that spanned multiple blocks
    - The SPA's reflected useful information regarding the black-boxed block's internal logic
    - The black-box run(s) completed in **~2 hours**

# Example runs with Enhanced Black-box Methodology (Continued)

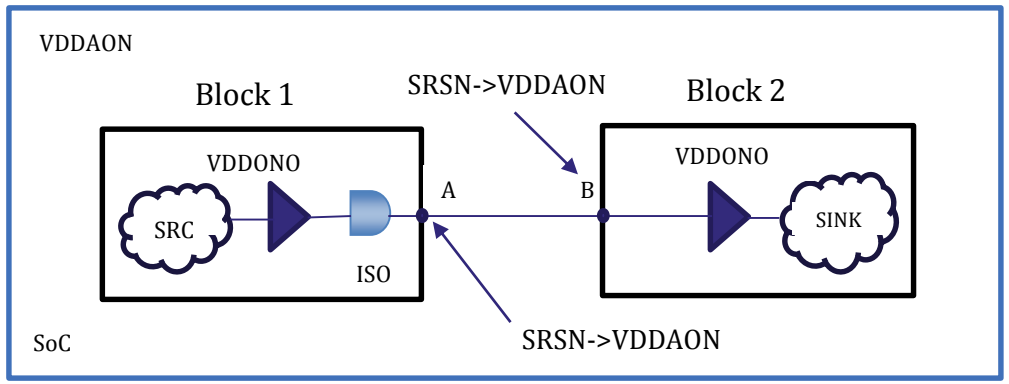
- The black-box run(s) correctly reported ~85% violations that were reported in the Full Hierarchical run.
- Most of the 15% unreported violations were Single-Block Violations.

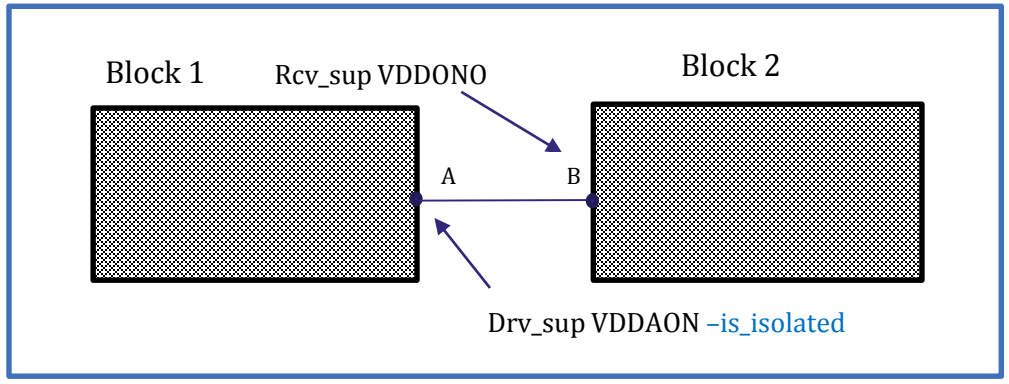
– For Top-level (Full-Chip) Design #2: Same scenarios as #1, except -
  - The initial Full Hierarchical run completed in **~4 hours**
  - All (**100%**) blocks were black-boxed
  - The black-box runs completed in **~45 mins**
  - The black-box run(s) correctly reported **~70%** violations that were reported in the Full Hierarchical run

# Future Work For Additional Accuracy

- Support for additional SPA “protection” attributes

```
set_port_attribute  -model <bbox_name> -port {<bbox_port_name>}
-attribute SNPS_BLK_PROTECTION is_isolated/is_protected
```

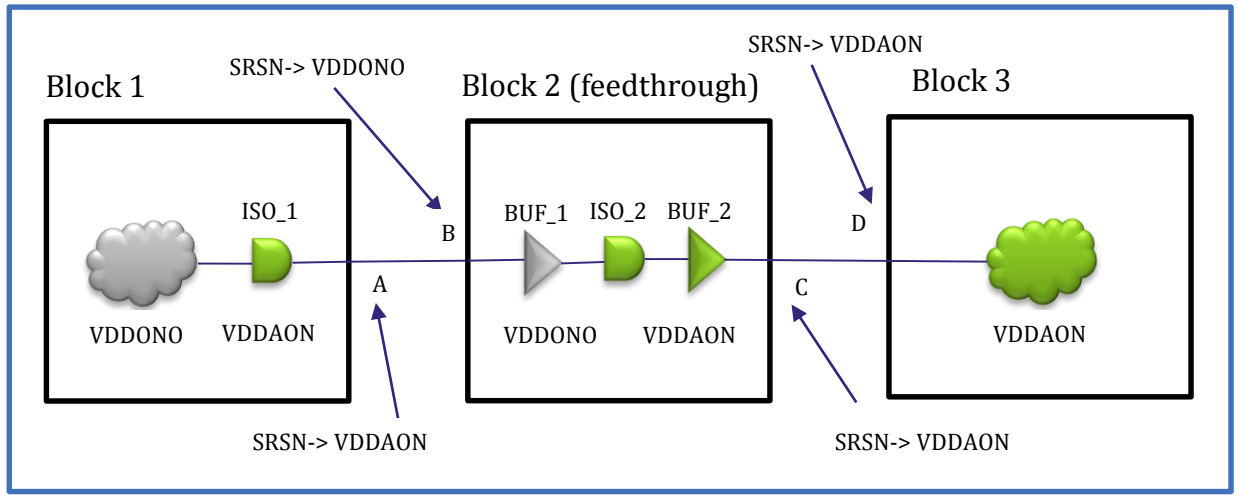


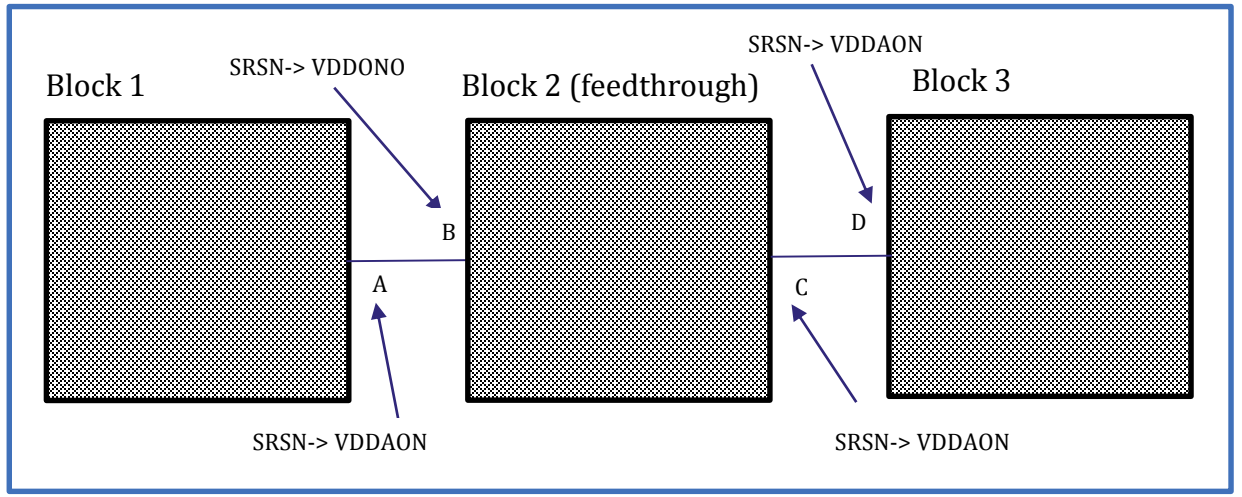



# Future Work For Additional Accuracy

- Support for additional SPA “feedthrough” attribute

```
set_port_attributes –model BlockB -feedthrough –ports { B C }
```



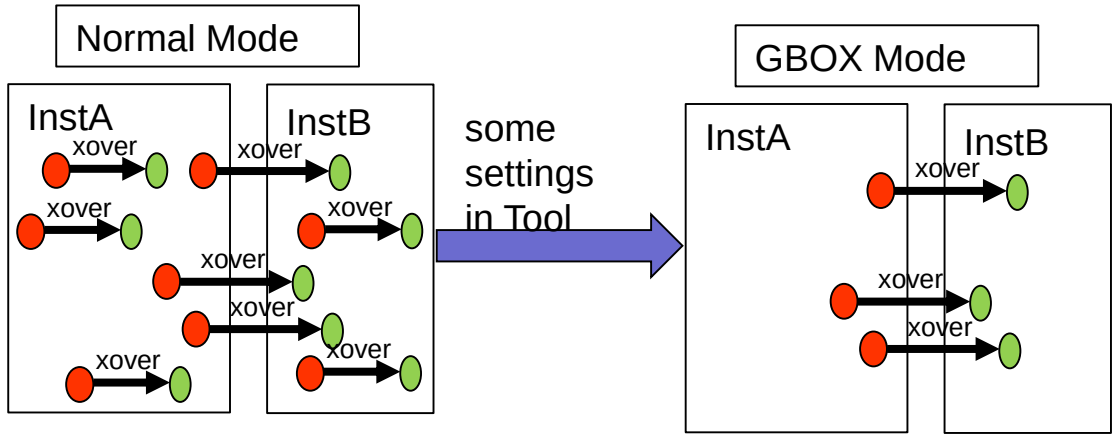




- When a crossover passes through a feedthrough block, VC-LP sees a redundant ISO_2 + incorrectly connected power rails for BUF_1 in hierarchical run (single crossover)
- When these are black-boxed crossovers are broken up - A->B and C->D and separately analyzed. VC-LP sees no violation

# New Ideas To Improve Accuracy (Synopsys)

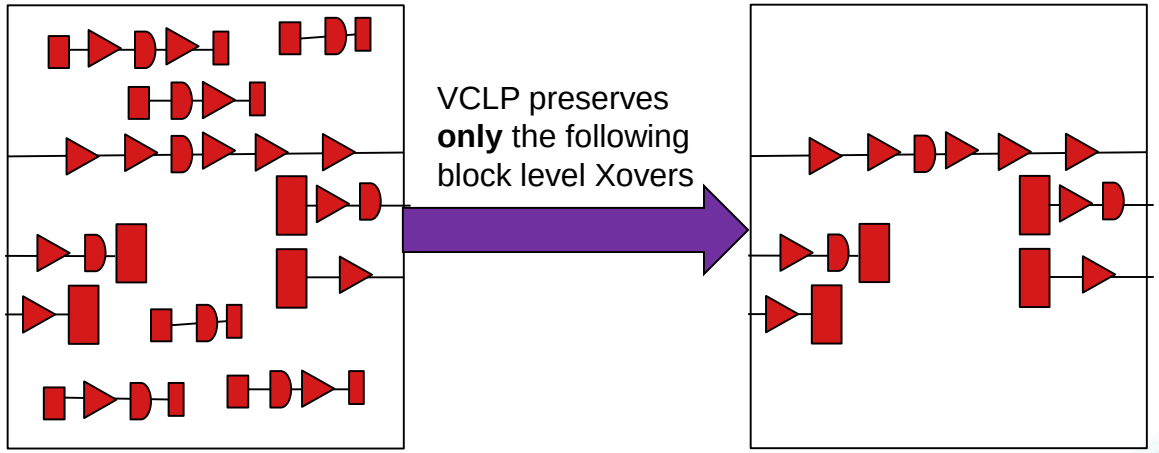
## Crossover generation after loading Hierarchical Design and upf

- Full Hierarchical netlist / upf is read in Top
- VC-LP only creates inter-block crossovers between User-Specified “glass”-boxes
  - More accurate and complete than SPA
  - No reduction in design parse, elaborate
  - Potential huge reduction in crossover gen + overall time




## Glass-box Model pre-gen

- Glass-box models for blocks are pre-generated with logic and supply networks that may contribute to inter-block crossovers
- Reduced logic glass-boxes are read in for Top-level
  - More accurate and complete than SPA
  - Huge reduction in parse, elaborate, overall time



# Conclusion / Summary

- Block-level VC-LP runs
  - Short turnaround times, use less memory
  - Can be run with full hierarchy
- Top-level / Full Chip VC-LP runs
  - Run-times, memory can drastically increase when run fully hierarchical
  - Difficult to have multiple iterative runs
  - **Selective Black-boxing** is an effective method to reduce run-time, but at the expense of accuracy of results
  - Propagating supply, connectivity info to the boundary ports of black-boxes via **`set_port_attribute`** greatly improves accuracy of results
  - Complete coverage for “protection” checks possible using black-boxing, while reducing run-time

AMD

snug
Synopsys Users Group
Austin

# Thank You

## Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.
The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.
AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.
AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
ATTRIBUTIONS