Blackboxing Techniques for Improving VC-LP Throughput

Parag Mandrekar (AMD, Sunnyvale CA)

Joseph Gutierrez (AMD, Austin TX)

Hank Lin (AMD, Shanghai PRC)

Vishwanath Sundararaman (Synopsys, Austin TX)

www.amd.com

ABSTRACT

Today's designs are growing in size and in Low Power complexity, driving a need to find methods to increase the throughput of Low Power (LP) checks. In traditional LP Flows, UPF files are created for the full instance hierarchy of the design RTL/netlist. The commands/constructs in the UPF reference some of the hierarchical design objects inside the instances. In today's SoC design environment, however, the full-hierarchy of design instances may not be available at the same time. LP checks at SoC-level do not have to be deferred until all blocks' netlists are completed. Black-boxing of the not-completed instances have been effective in running LP checks at SoC-level. Also, black-boxing instances previously verified saves run-time and avoids re-analyzing issues already known at block-level. Black-boxing is also useful in analyzing top-level port connectivity between blocks. This paper discusses approaches to effectively use UPFs on black-boxed objects and details its capabilities and limitations.

Table of Contents

Table of Figures

Table of Tables

1. Introduction – Static PRC (Power Rule Checks)

VC-Static-LP (VC-LP) has been widely used at AMD for performing Static Power Rule Checks (PRC) on multi-voltage designs at block-level as well as SoC-level. The tool can be used along with a UPF (Unified Power Format) power intent file, both on the RTL as well as the netlist side.

For RTL PRC, the checks are primarily associated with the UPF: at RTL stage, the UPF is checked for correctness and the presence or absence of the correct Isolation/Level-Shifting strategies, since the RTL usually does not yet have these constructs.

After Logic Synthesis , the resulting netlist has Isolation, Level-Shifters, Power Switches and other cells instantiated – VC-LP can now be used to additionally perform netlist-side checks, to ensure it confirms 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.

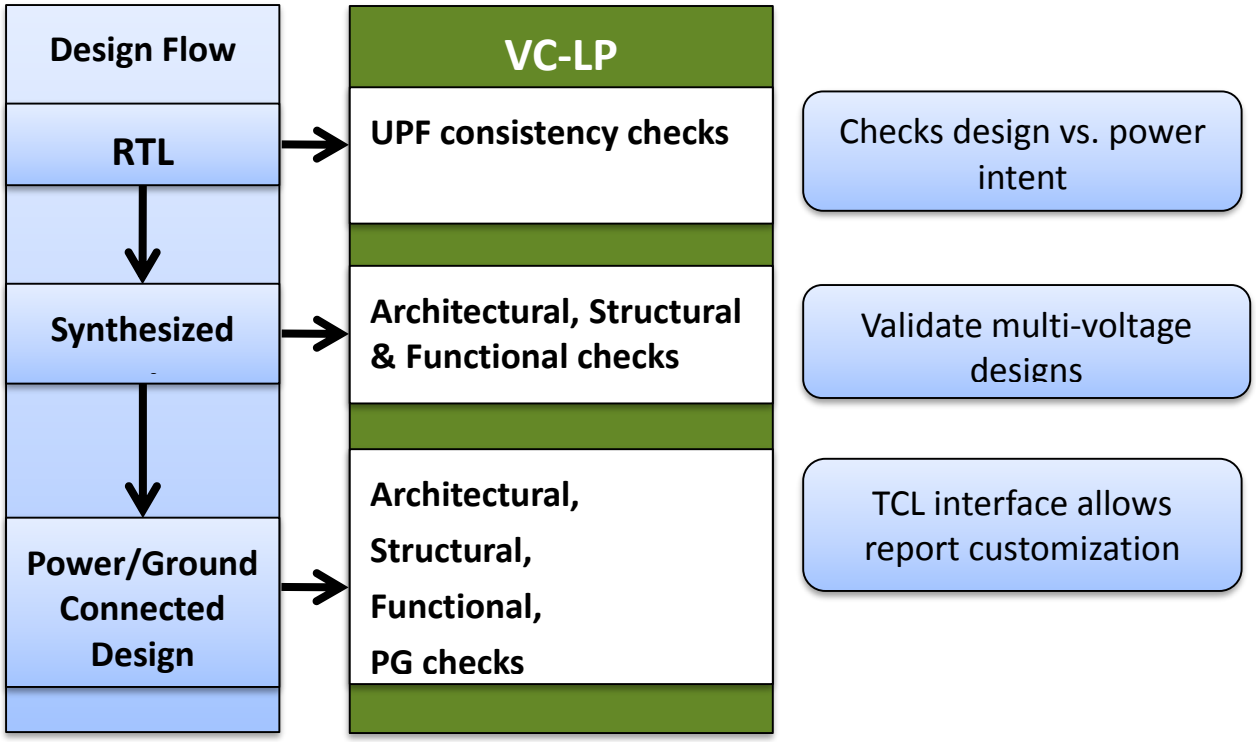

Figure 1. VC-LP Usage throughout the Design Flow

VC-LP is designed part of the VC-Static Platform (LP, FV, Lint, CDC) and runs via commands inside a tcl-based shell. It has been designed to work seamlessly with other Synopsys tools like Design Compiler (DC) and IC Compiler (ICC). Among its useful features are an all-new intuitive Graphical Interface with a Violation Viewer that can be used to cross-reference design objects with the relevant section of the Schematic, Verilog source and UPF. Other features include separate, de-coupled Build, Check and Report steps, LP database save restore that enables incremental checking without re-building the database, commands to skip, configure, filter and waive checks and access to the LP database via API calls to enable scripting.

1.1 Hierarchical vs. Black-Box runs

VC-LP generates violation reports at block-level with quick turnaround times (typically a few minutes to an hour). However, when these blocks are instantiated with full hierarchy at SoC-level, the turnaround time can drastically increase and could take significantly longer. Thus, running VSI-LP at SoC level can become CPU and memory intensive if it is run with the entire hierarchy of logic blocks exposed.

VC-LP performs its analysis and Rule Checks on "crossovers" in the design: a crossover is a path in the design that contains a domain boundary. The tool tries to identify crossovers that contain single/dual rail buffers/inverters, Isolation/Level-Shifter strategies and intermediate domain boundaries. These crossovers could hence span multiple blocks. Exposing the entire logic hierarchy thus enables the tool to see the complete crossover – it is consequently able to look for violations more accurately. Since the entire hierarchy is available for performing Low Power Checks, the analysis is usually the most complete and accurate.

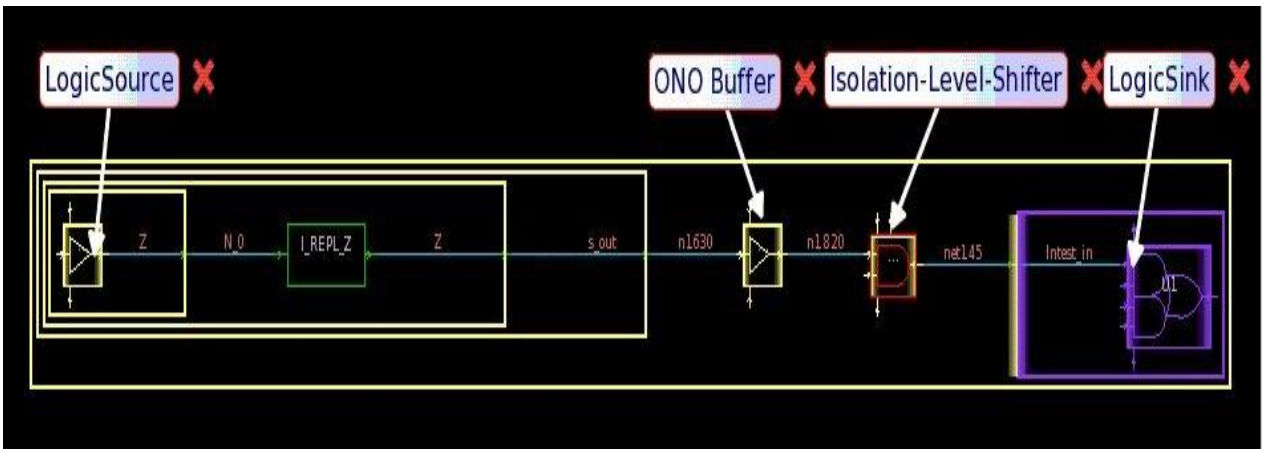

Figure 2. Example crossover spanning 2 blocks, with a domain boundary

However, in a project, all the blocks' netlists and UPFs may not be complete and available the same time. Hence, selective black-boxing approaches are used to proceed with checking at the SoC-level.

The following are some scenarios where black-boxing lower level blocks has been useful:

For blocks whose design netlists or upf files are not ready yet, black-boxing these blocks enables the checking of other blocks' low power problems at full SoC level.

For blocks that have already been verified at block-level and which do not affect other blocks - black-boxing those blocks saves run time and avoid re-analysis of issues already known at block level. Examples of these blocks are those that have their primary inputs and outputs completely protected from within the block (using ISO and LS cells) and blocks that have no ONO supplies.

If we want to only check whether there are missing Level Shifters at the SoC level, between blocks, we only check the port connectivity of those blocks with other blocks in the SoC. In this case, it is beneficial to black-box the blocks whose internal logic is not relevant to the analysis.

Black-boxing of the blocks at SoC-level also has some drawbacks:

A stub has to be created for block netlist, since VC-LP is designed to read the entire module and subsequently use only the interface.

Block upf constructs, objects and strategies inside the block hierarchy may not be applicable. VC-LP could report errors or ignore them.

As explained earlier, since the entire hierarchy is not available, VC-LP may not be able generate the full crossover. VC-LP may thus incorrectly report or miss violations or those crossovers.

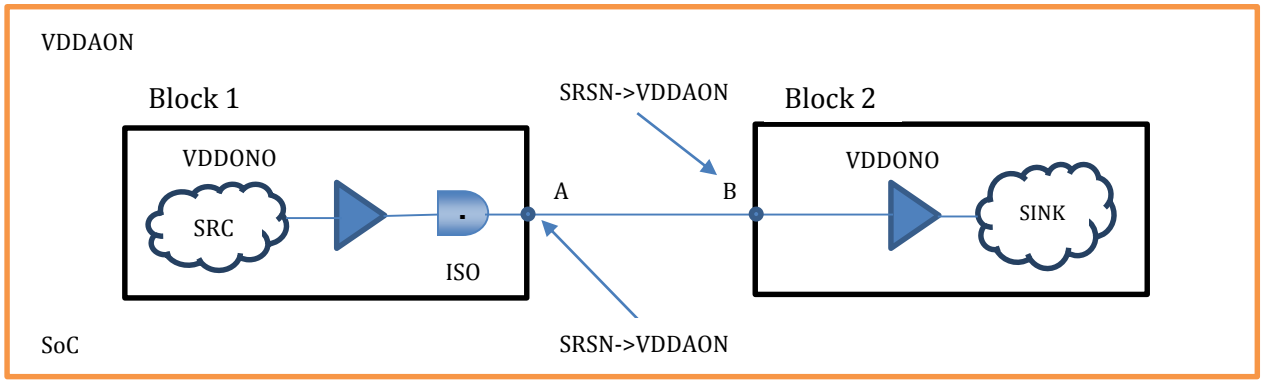

Figure 3. ISO_INST_REDUND (warning) reported in full hierarchical SoC-level run

In the above figure at Block-level, for Block 1, there is a `set_related_supply_net` (SRSN) of VDDAON (always-on supply) on output port A. Hence, an ISO has been inserted, that isolates A from logic driven by supply VDDONO (ON-OFF supply). Similarly, for Block 2, due to the SRSN of VDDAON on input port B, no ISO is necessary to protect the driven input of the buffer.

Now, when Block 1 and Block 2 are instantiated with full hierarchy at SoC-level, VC-LP sees the entire crossover from SRC to SINK and reports an ISO_INST_REDUND warning, since both SRC and SINK are driven by VDDONO. VC-LP, in this case, does not use the SRSN information.

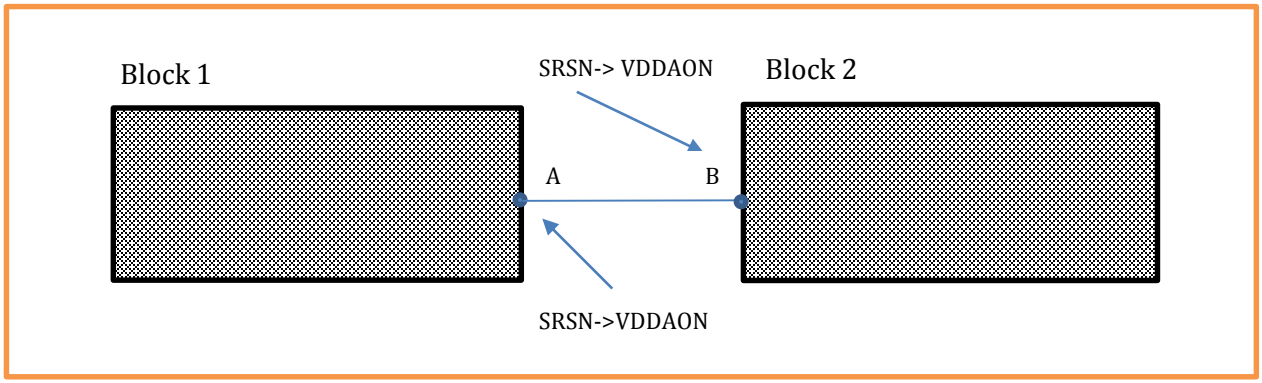

Figure 4. No violation reported at SoC-level when the blocks are black-boxed

However, when both blocks are Black-boxed at SoC-level as seen in Figure 4, VC-LP uses the SRSN information on the ports. Now, no warning is reported since port A is the SRC and port B is the SINK and their relative supplies are both VDDAON.

2. Low Power Check Methodology

2.1 Block-level runs

At the block-level, VC-LP is able to read the design information and complete the Low Power analysis with quick turnaround times, usually less than 1 hour. All block-level runs that have complete design information can be done in parallel.

Block-level violations reported by VC-LP on crossovers that have their source as well as sink inside the block (**Single-Block Violations**), need to be checked and fixed/waived, if that block is intended to be black-boxed at SoC-level – these violations are contained entirely within the block and usually are not seen at SoC-level.

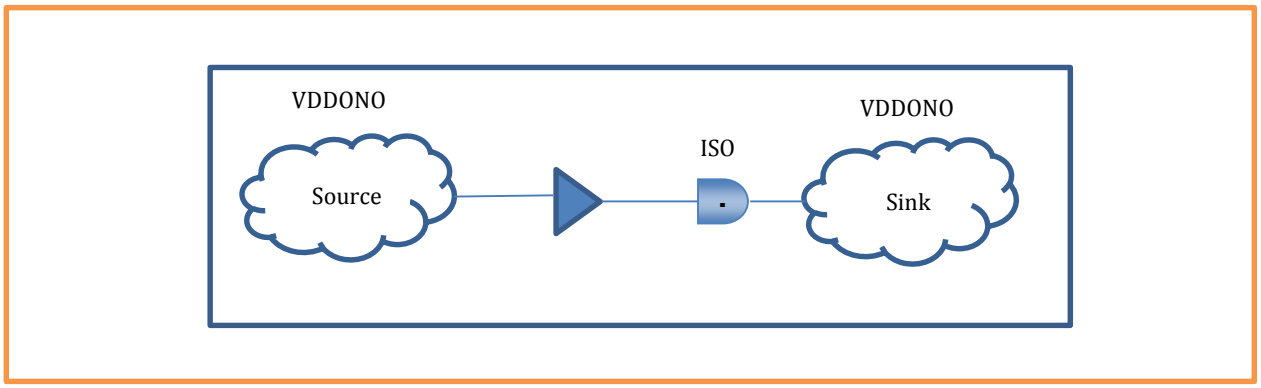

Figure 5. ISO_INST_REDUND (warning) violation inside block

In the example in Figure 5, at block-level VC-LP will report an `ISO_INST_REDUND` warning for the crossover shown, since the source as well as the sink are driven by supplies VDDONO. Since both the source and the sink are entirely inside the block, the crossover is not seen when the block is black-boxed at SoC-level, hence this violation is not reported at that level.

Also, violations specific to design instances inside the block should be checked and fixed – these violations are reported on the instance itself and not on crossovers. An example violation is `ISO_CLAMP_INVERT` (error) – where the isolation strategy has defined a clamp value whose polarity does not match the actual clamp value of the ISO instance in the design. Such errors are not caught when the block is black-boxed at SoC-level.

2.2 SoC-level runs

As explained earlier, a fully hierarchical run with all logic exposed, along with hierarchical set of UPFs gives the most accurate results. If the design contains large components (which in turn have blocks) – iso boundaries known and component separately verified, these can be black-boxed. If the components inputs are "fully protected" and outputs are fully isolated inside the component, then the component can be black-boxed.

Static Power Rule Checks at SoC-level could have significantly long run times. Due to the long turnaround time, there are a limited number of these runs that can be performed in the design cycle.

Black-boxing of blocks in the design greatly improves run-time, allowing for quick analysis and fixing/waiving of violations. After the results of the first hierarchical run have been analyzed, selective black-boxing techniques can be used to quickly iterate over fixing and waiving specific violations. For example, if the 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, 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.

2.3 Propagating design information to the block ports via SPA's

In general, VC-LP run time at SoC-level will significantly improve with black-boxing – the more blocks black-boxed, the greater the reduction in the run-time.
However, the cost of this black-boxing is reduced accuracy in reporting violations between blocks. To improve this accuracy, information from the design is abstracted to the ports via SPA's.

As detailed in the next section, propagating and adding information like driver/receiver supplies, tied inputs, unconnected inputs and outputs, iso_enable ports, ports isolated/protected inside the blocks and feedthrough ports add useful information for VC-LP to analyze crossovers that reference those block ports. With this SPA information, VC-LP is able to better identify missing or redundant ISO/LS, iso_en source and polarity, tied input conditions and other Low Power related issues, resulting in less fake violations and false negatives. On our SoC 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.

3. Steps involved in enabling Black-box Flow

3.1 Running at block-level

During Low Power Checking at block-level, the VC-LP command `write_bbox_data_model` can be used to write out information at the black-box ports, via `set_port_attributes` (SPA) statements. This writes out the SPA in a file called "**bboxAttribute.upf**", which can be concatenated to the end of the existing block.upf file.

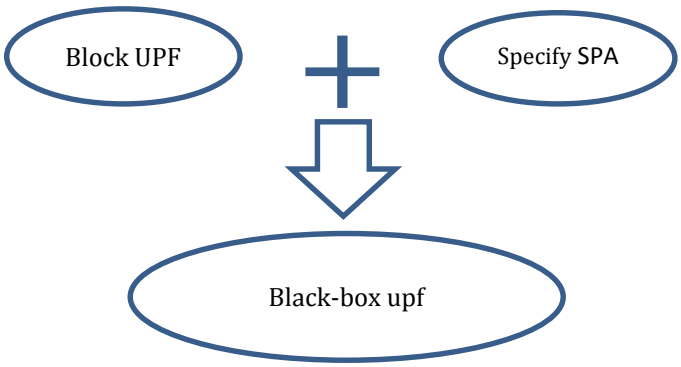

Figure 6. Creating a block.upf

VC-LP derives and propagates design information from the logic inside the block to the ports of the block. Some of these port attributes are

- 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 {0} \
-attribute SNPS_BLK_MODEL_TIED_NAME {blackbox/LOGIC1}
```

- 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_port_name>}
-attribute SNPS_BLK_MODEL_UNCONNECTED {<1/0>}
-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}
```

- 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}
```

In addition to this, a tcl script based on the VC-Static/VC-LP API is used to generate driver/receiver supply information for each port. In future versions of VC-LP this script's functionality will be handled in a native VC-LP command.

For Black-Box Input Ports

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

- **For 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 TCA_TCC7_req[1] -
driver_supply SS_PD1 -receiver_supply  SS_GLB
set_port_attributes -port TCA_TCC7_req[0] -
driver_supply SS_PD1 -receiver_supply  SS_GLB
set_port_attributes -port TCA_TCC4_hole_send -
driver_supply SS_PD1 -receiver_supply  SS_GLB
```

The `-driver_supply` for a block output port is the supply of the last (driving) cell which is connected to the output port
The `-receiver_supply` for a block input port is the supply of the first (driven) cell's which is connected to the input port

SPA's written out in this manner by VC-LP 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. In this way the block-level UPF can be used for block-level as well as SoC-level Low Power Checks.

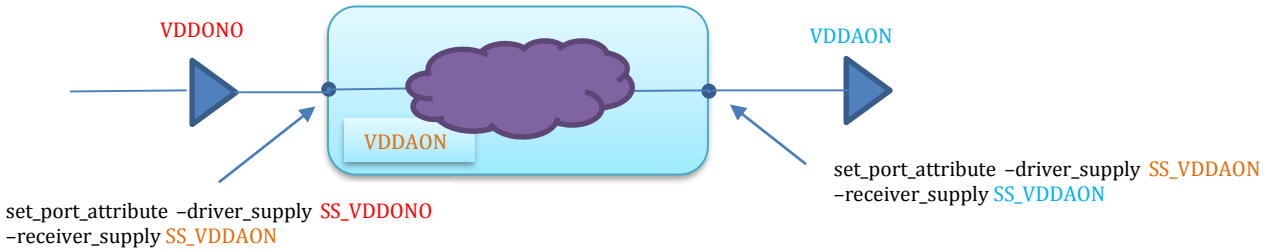

Figure 7. SPA's computed for black-boxed inputs and outputs

The following diagram details the steps involved in the Block-level VC-LP run.

Block-level verification steps	
Description	Commands Flow
Read in block design netlist, UPF and verification setup (via tcl *app_vars*)	`read_file -netlist …..` `read_upf block.upf`

Check violations	`check_lp -stage upf`
	`check_lp -stage design`
	`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`
	`# Script to generate driver/receiver supply info`
	`source $FLOWDIR/gen_drv_recv_spa.tcl`

Save the database / quit	`save_session` `quit -f`

Figure 8. Block-level verification steps

For VC-LP SoC-level runs, the bboxAttribute.upf is concatenated to the end of the block.upf file.
All the upf statements are scoped at block-level and are loaded in the UPF at SoC-Level using "`load_upf -scope`"

3.2 Creating the block.upf file for for use as black-box

The block.upf file is originally intended for use for block-level VC-LP Low Power Checks. The same upf, however, can be used for the black-boxed block at SoC-level. Care has to be taken to correctly specify upf constructs like Power State Tables (PST's) and UPF objects like isolation strategies, and `connect_supply_net` statements.

Some of the requirements are as follows:

- Each 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:
    i. Top power domain of the block.
    ii. Supplies for the top power domain of the block.
    iii. Power State Table (PST) to describe the relationship between the supplies, based on only the supplies defined in the top power domain. Any supplies referenced inside the block hierarchy are ignored when parsing the PST.

iv. Any design references, strategies, and connect_supply_net statements inside the black-box hierarchy are ignored by VC-LP.

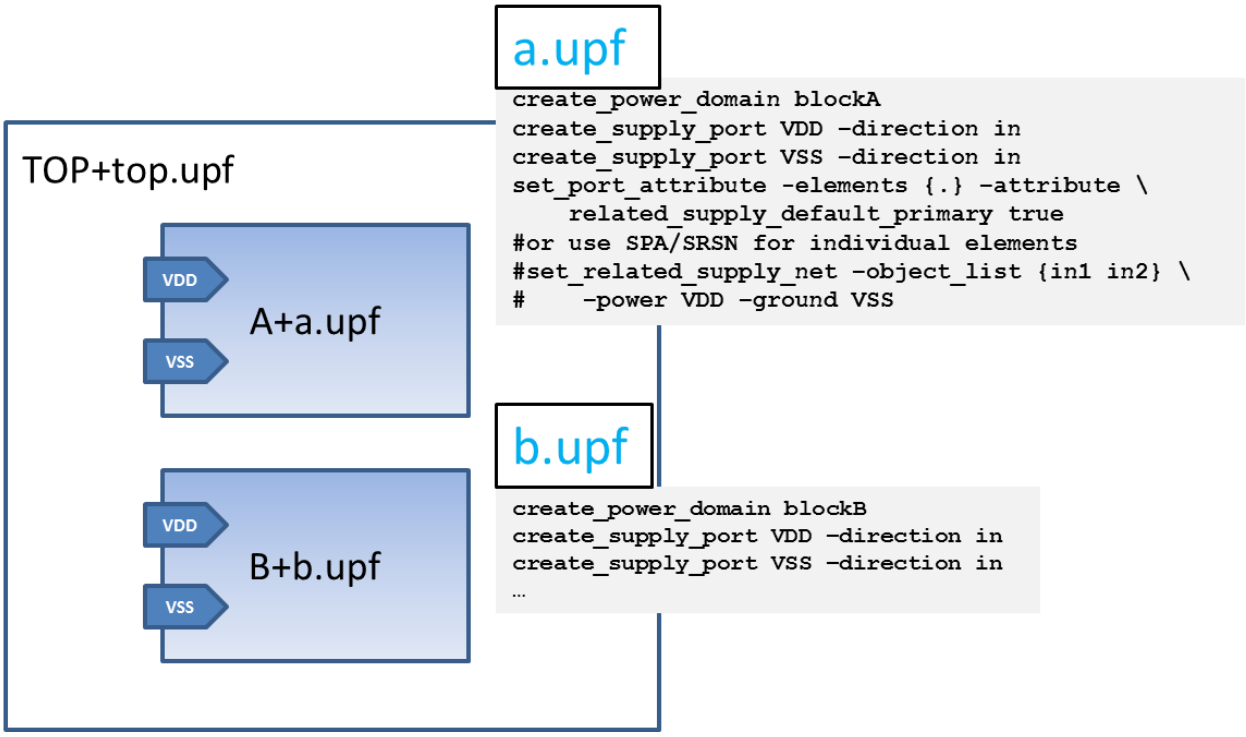

Figure 9. block.upf showing supply definitions and SPA's at block boundary

The following table taken from the VC-LP black-box flow App Note details how VC-LP interprets UPF constructs in a block.upf.

Table 1. Black-box UPF - Power Domains

`create_power_domain bb_pd`	Processed
`create_power_domain bb_pd -supply {primary ss1} -update`	Processed
`create_power_domain <Nested_hierarchy>/power_domain -element {sub}`	Read & Ignored (references hierarchy inside block)
`create_power_domain bb_pd -include_scope -element {sub/u1}`	Element sub/u1 is filtered out
`set_domain_supply_net bb_pd -primary_power_net vdd1 \ -primary_ground_net vss1`	Processed

`set_domain_supply_net <Nested_hierarchy>/power_domain –primary_power_net sub/vdd_n1 –primary_ground_net sub/vss_n1`	Read & Ignored (references hierarchy inside block)

3.2.1 Supply Ports

The commands `create_supply_port`, `create_supply_set`, and `create_supply_net` will be honored if specified at black-box scope. If specified, the supply port construct will be created at the interface of the black-box scope. Any of the above commands specified on nested black-box scopes will be ignored.

Table 2. Black-box UPF - Supply Ports

`create_supply_port VDD –domain bb_pd`	Processed
`create_supply_port VDD_n1 –domain <Nested_hierarchy>/power_domain`	Read & Ignored (references hierarchy inside block)
`create_supply_net vdd1 –domain bb_pd`	Processed
`create_supply_net vdd_n1 –domain <Nested_hierarchy>/power_domain`	Read & Ignored (references hierarchy inside block)
`create_supply_set ss1 –function {power vdd1} –function {ground vss1}`	Processed
`create_supply_set ss_n1 –function {power vdd_n1} –function {ground vss_n1}`	Read & Ignored (references hierarchy inside block)

3.2.2 Supply Connections

Explicit supply connections to existing supply ports or black-box PG pins through `connect_supply_net` are honored. The command `associate_supply_set` is honored if all the supply objects are scoped at the black-box scope.

Table 3. Black-box UPF - Supply Connections

`connect_supply_net vdd1 –ports VDD`	Processed
`connect_supply_net vdd_n1 –ports VDD_n1`	Read & Ignored (references hierarchy inside block)
`associate_supply_set ss1 –handle bb_pd.primary`	Processed
`associate_supply_set ss_n1 –handle <Nested_hierarchy>/power_domain.primary`	Read & Ignored (references hierarchy inside block)

3.2.3 Related Supplies for Ports

`set_port_attributes` will be allowed on black-box scope. `set_port_attributes –repeater_supply` will be honored if the objects specified are at the black-box scope. Otherwise, it is ignored. For -receiver_supply/–driver_supply, it will be ignored by implementation tools, but will be checked by verification tools.

Command set_related_supply_net will be honored if the objects specified are at the BBOX scope. Otherwise, it is ignored.

Table 4. Black-box UPF - Related Supplies for Ports

`set_port_attributes –elements {.} –applies_to inputs –repeater_supply rs1`	Processed
`set_port_attributes –elements {.} –applies_to inputs -driver_supply ss2 –receiver_supply ss1`	Read & Ignored (references hierarchy inside block)
`set_related_supply_net –object_list [get_ports INPUT]–power vdd1 –receiver_supply vss1`	Processed
`set_related_supply_net –object_list [get_ports sub/INPUT]–power vdd_n1 –receiver_supply vss_n1`	Read & Ignored (references hierarchy inside block)

3.2.4 Power Switches

Command `create_power_switch` will be honored at the black-box scope. If specified, a power switch construct will be created at the black-box scope. Options refering to invisible objects will be ignored. Any power switch command specified on the nested black-box domains will be ignored.

Table 5. Black-box UPF - Power Switches

`create_power_switch sw1 -domain bb_pd \`
`-output_supply_port {vout VO1} \`
`-input_supply_port {vin VI1} \`
`-control_port {ctrl_small ON1} \`
`-on_state {full_s vin {ctr_small}}`
	Processed

`create_power_switch sw_n1 -domain <Nested_hierarchy>/power_domain \`
`-output_supply_port {vout_n VO_N1} \`
`-input_supply_port {vin_n VI_N1} \`
`-control_port {ctrl_small_n ON_N1} \`
`-on_state {full_s vin_n {ctr_small_n}}`
	Read & Ignored (references hierarchy inside block)

3.2.5 Power State Tables

Power state commands `add_power_state`, `add_pst_state`, `add_port_state`, and `create_pst` are honored if they use supplies at the black-box scope. Otherwise, they will be ignored.

Table 6. Black-box UPF - Power State Tables

`add_power_state bb_pd.primary -state HVp \ {-supply_expr {power == `{FULL_ON, X, Y, Z }}}`	Processed
`add_power_state bb_pd.primary -state HPg \ {-supply_expr {ground == `{OFF}}}`	Processed

3.2.6 Isolation Strategies

Location parent isolation strategies are defined on the power domain scoped on a black-box are honored, other all are ignored. For `set_isolation_control`, strategies other than location parent will be ignored during top level verification. If an isolation strategy is honored, the corresponding `map_isolation` command will be honored, or else if the isolation strategy is ignored, then the corresponding map_isolation will be ignored

Table 7. Black-box UPF - ISO Strategies

`set_isolation iso1 -domain bb_pd -elements special_port1 -isolation_supply_set iso_ss`	iso1 will be read and stored
`set_isolation iso2 -domain bb_pd -elements special_port2 -isolation_supply_set iso_ss`	iso2 will be read and stored
`set_isolation_control iso2 -domain bb_pd -location self - isolation_signal en -isolation_sense high`	iso2 will be ignored at the top level
`set_isolation iso3 -domain bb_pd -elements sub/special_port1 -isolation_supply_set iso_ss`	ignored at the top level
`set_isolation iso4 -domain bb_pd -elements special_port3 -no_isolation`	Command will be Processed

As an aside, a "`-location parent`" policy at black-box top scope is also allowed from TOP scope.

`set_isolation top_iso1 -domain BBOX/bb_pd -elements BBOX/special_port1 -isolation_supply_set BBOX/iso_ss`	iso1 will be read and stored
`set_isolation_control top_iso1 -domain BBOX/bb_pd -location parent -isolation_signal`	iso 1 valid at the top level

`BBOX/en –isolation_sense high`	

3.2.7 Level-Shifter Strategies

Level-shifter strategies (`set_level_shifter`) on black-boxes will be honored if `–location parent` is specified. Otherwise, it is ignored.

Table 8. Black-box UPF - LS Strategies

`set_level_shifter ls1 –domain bb_pd –elements special_port1 -location parent`	ls1 valid at SoC-level
`set_level_shifter ls2 –domain bb_pd –elements special_port2 -location self`	ls2 Ignored at SoC-level
`set_level_shifter ls3 –domain bb_pd –elements special_port3 -no_shift`	Processed

As an aside, a "`–location parent`" policy at black-box top scope is also allowed from TOP scope

`set_level_shifter top_ls1 –domain BBOX/bb_pd –elements BBOX/special_port1 -location parent`	ls1 will be implemented at the top level

3.3 Creating Stub modules for the block

Also at block-level, Stub modules should be created for black-box use. When the module is black-boxed at SoC-level, VC-LP reads the entire hierarchical block netlist and keeps only the interface (in/out/bidir port) information.

To avoid spending time reading the entire hierarchy, the Stub module should be created beforehand. This can be achieved with a simple script that parses the top module in the block netlist and writes out a Verilog netlist with just the interface information (empty module with only port information).

3.4 Running at SoC-level

For SoC-level runs, the tcl command below is used to specify the list of block modules or instances to be black-boxed.

- set_blackbox
  [-cells <cell>]
  (List of instances to be black-boxed, separated by " ")
  [-designs <design>]
  (List of modules to be black-boxed, separated by " ")

VC-LP refers to these blocks as UBB (**User Defined Black-boxes**) and considers all the specified modules as black-boxes even if the design file has a complete module definition. All the internal logic of these modules is ignored during elaboration/crossover generation.

This VC-LP command must be used before reading the design.

- Example

```
set_blackbox -cells "u1"
# using Instance name
```

or

```
set_blackbox -designs { block_stub_a  block_stub_b }
# using module name
```

In the vc_static_shell interactive tcl shell, the "report_link" command can be used to verify the type of black-box (Stub, unresolved module, UserBB, etc.)

```
vc_static_shell> report_link
Module            Instances    Reason
------            ---------    ------
block_stub_a      1            UserBB
```

The read_upf -quietBbox VC-LP command is used to read in the block upf at the SoC-level.

This command parses the upf and terminates with a hard error only for incorrect syntax. Any errors inside the hierarchy of a black-boxed block scope are ignored. This capability enables usage of the same upf file for block-level as well as SoC-level runs.

During parsing, the following messages are seen.

```
[Info] SM_EMPTY_SKIP: Marking empty Module/Entity as blackbox
    Module/Entity 'moduleX' is marked as a blackbox because it is
empty

[Info] SM_EMPTY_SKIP: Marking empty Module/Entity as blackbox
    Module/Entity 'moduleY' is marked as a blackbox because it is
empty

[Warning] SM_BB_LIST: Blackbox module list
    List of modules in design which are blackboxed by tool/user (2):
moduleX moduleY
```

These messages indicate which modules were parsed as black-boxes by the user (via set_blackbox) or the tool (unresolved module, stub, etc.).

A sample tcl command file for a VC-LP black-box run looks like the following:

```
set search_path …
set link_library "…"
set_app_var …
set_blackbox -designs { moduleX moduleY }
read_file -top soc -netlist { soc.v moduleXStub.v moduleYStub.v … }
catch { read_upf soc.upf } read_upf_error
if { $read_upf_error == 0 } {
  puts "ERROR : Problem(s) during Upf parsing. Please check log file.
Exiting\n" ;
  quit -force
}
check_lp -stage upf
check_lp -stage design
check_lp –stage pg
report_lp -limit 0 -verbose -file soc.rpt
save_session
quit –f
```

The command `infer_source` is **not** used for these runs, since Signal corruption checks are not applicable for SoC-level Black-box runs.

4. Example runs with Enhanced Black-boxing Methodology

The enhanced black-box approach was tried on two SoC designs at AMD. Using netlist stubs, block.upf's created with the relevant constructs visible in black-box scope and SPA's generated from write_bbox_data_model/API script, the VC-LP run-time to build the data, perform checks and generate reports was hugely reduced. While Single-Block Violations were not reported for the black-boxed instances, most others were correctly identified and reported.

The faster turnaround time enabled the team to rerun multiple times at SoC-level and fix or waive off potential issues with each successive iteration.

For SoC #1:

- The initial run was Full Hierarchical – and completed in ~26 hours. The run had >80 block modules. ~50 of these 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 tiles.
- The violations from this 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.
- The SPA's reflected useful information regarding the black-boxed block's internal logic.
- The black-box run(s) completed in ~2 hours.

- 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 SoC #2:

- Same scenarios as SoC #1, except:
    - The initial Full Hierarchical run completed in ~4 hours.
    - All ~40 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.

4.1 Consistency checks reported when using SoC-level Black-box Flow

At block-level, the design is verified based on the supply that is connected to external logic driving the block port input and also the supply that is connected to the external logic being driven by the block port output. These external supplies are reflected at the block inputs and outputs via SRSNs or SPAs (`-driver_supply` for inputs and `-receiver_supply` for outputs).

After the block-level has been signed off, if these blocks are black-boxed at SoC-level, VC-LP performs 2 consistency checks.

`UPF_DRIVERSUPPLY_MISMATCH` : This warning is reported if the `-driver_supply` specified for the driving block's output port does not match the `-driver_supply` of the input port of the driven block.

`UPF_RECEIVERSUPPLY_MISMATCH` : This warning is reported if the `-receiver_supply` specified for the driven block's input port does not match the `-receiver_supply` of the output port of the driving block.

In the below example, a `UPF_DRIVERSUPPLY_MISMATCH` is reported, since BlackBox2's input port has `-driver_supply` VDDONO2 (expected supply) but BlackBox1's output port has `-driver_supply` VDDONO1 which is the actual supply of the driving logic.

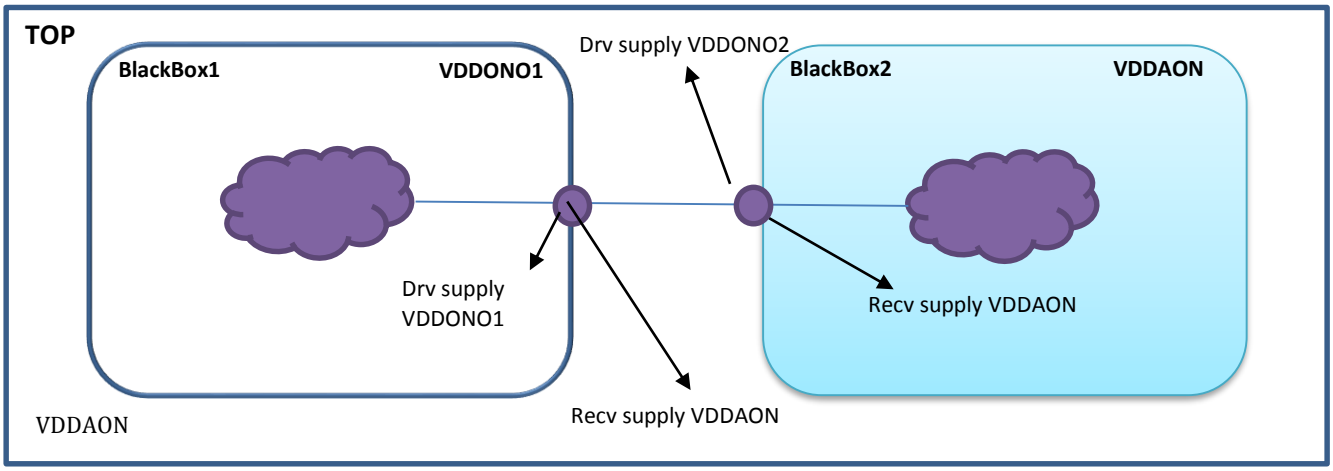

Figure 10. Consistency Checks for driver and receiver supplies of black-boxes

Tags Introduced:

Status	Severity	Issue
UPF_DRIVERSUPPLY_MISMATCH	WARNING	power or ground of Driver supply is inconsistent with the actual design driver
UPF_RECEIVERSUPPLY_MISMATCH	WARNING	power or ground of Receiver supply is inconsistent with the actual design receiver

4.2 PG checks reported when using SoC-level Black-box Flow

All the supply ports of a black-box are considered as PG ports. All PG checks that are applicable on hierarchical cells, will be applicable to the ports of the black-box as well.

VC-LP will add one additional debug field "**BlackBoxScope : True**" for any violation issued related to any black box pins.

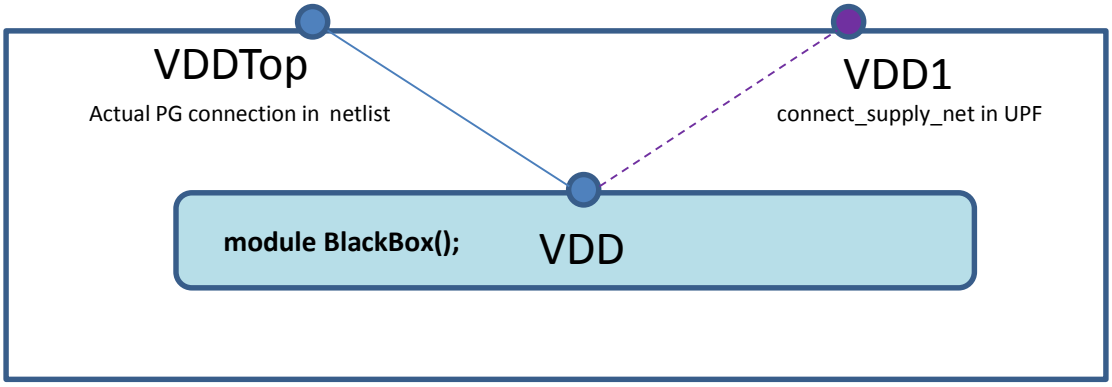

Figure 11. PG checks reported when using SoC-level black-box Flow

```
Tag           : PG_CSN_CONN
 Description   : UPF connect_supply_net [UPFSupply] does not match
design supply net [DesignNet] on pin [CellPin] of [CellType]
instance [Instance] ([Cell])

 UPFSupply     : VDD1
 DesignNet
     NetName     : VDDTop
     NetType     : Design/UPF
 CellPin       : VDD
 Instance    : i_BlackBox
 Cell           : BlackBox
 BlackBoxScope : True
```

4.3 Warning Messages from VC-LP during UPF parsing

1. `[Warning] USERBBOX_PORT_UNCONSTRAINED: User Black box port has no supply constraint`
   - This warning message is issued for any User black-boxed module, stub module or unresolved module non-pg pin that does not have SPA or SRSN supply constraints.

2. `[Warning] UPF_IGNORE_CMD: UPF Command Ignored in BlackBox/Libcell UPF`
   - This warning message is issued for commands that are inside the hierarchy of a Black-box.

3. `[Warning] UPF_BB_IGNORE_OBJ: UPF Object Ignored in BlackBox/Libcell UPF`
   - This warning is issue for any UPF Object (port, net, supply, PST state, etc.) that references hierarchy inside the block.

4. `[Warning] SM_BB_LIST: Blackbox module list - List of modules in design which are blackboxed by tool/user (2): moduleX moduleY`
   - This warning lists all the modules parsed as black-boxes by the user (via `set_blackbox`) or the tool (unresolved module, stub, etc.).

4.4 User Black-box Debug Field in VC-LP violation

- The following field has now been added to violations in the VC-LP report to identify a block instance at SoC-level as a User Black-box:
  i. **`BlackBoxScope : True`**

5. Future Work and Cases

The SPA's generated at block-level provide useful information abstracted from block logic to the black-box ports and enable VC-LP to perform a more accurate analysis during black-box runs, while providing huge run-time speedup.

Some cases, however, require additional helpful information that VC-LP can use in order to report the exact same violations for the crossovers as seen in the Full Hierarchical runs.

Some of these Cases are:

1. In a Full Hierarchical SoC-level run reports an ISO_INST_REDUND for the below crossover, since the Source and the Sink for the crossover are both powered by VDDONO.
   During SPA generation at block-level, VC-LP generates an SPA with `-driver_supply` VDDAON for the output port Block1/A, due to the supply of the driving ISO cell – and an SPA with `-receiver_supply` VDDONO for the input port Block2/B, as seen in Figure 12.

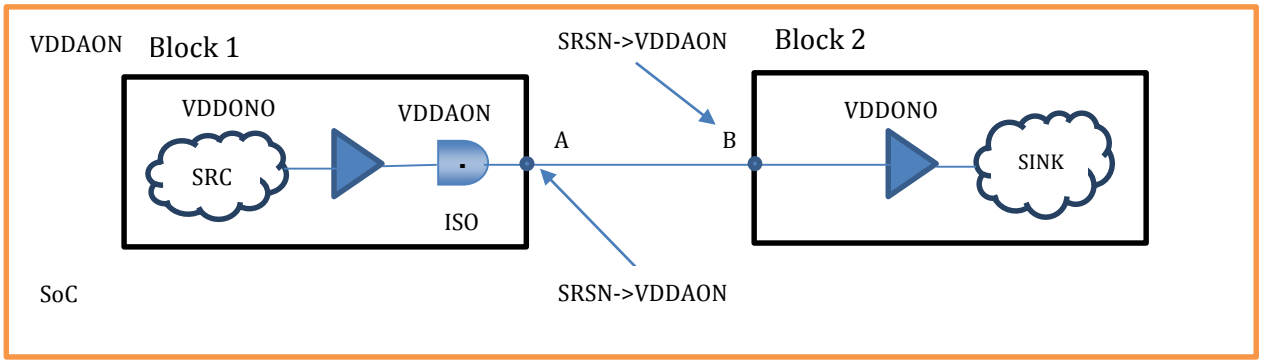

Figure 12. ISO_INST_REDUND (warning) reported in full hierarchical SoC-level run

When the 2 blocks are black-boxed, however, the ISO_INST_REDUND is not reported, since the crossover is now from Block1/A to Block2/B and, which is a VDDAON->VDDONO crossover, as seen in Figure 13.

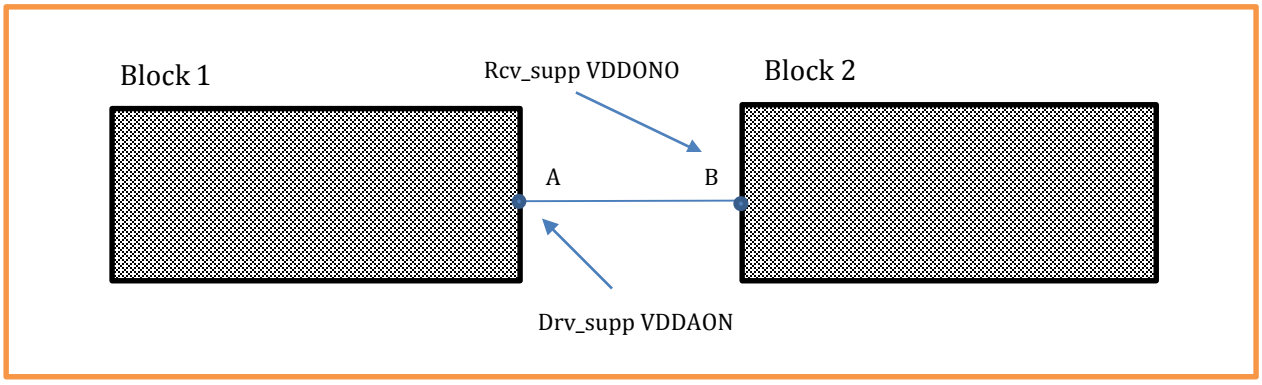

Figure 13. No violation reported in black-box SoC-level run

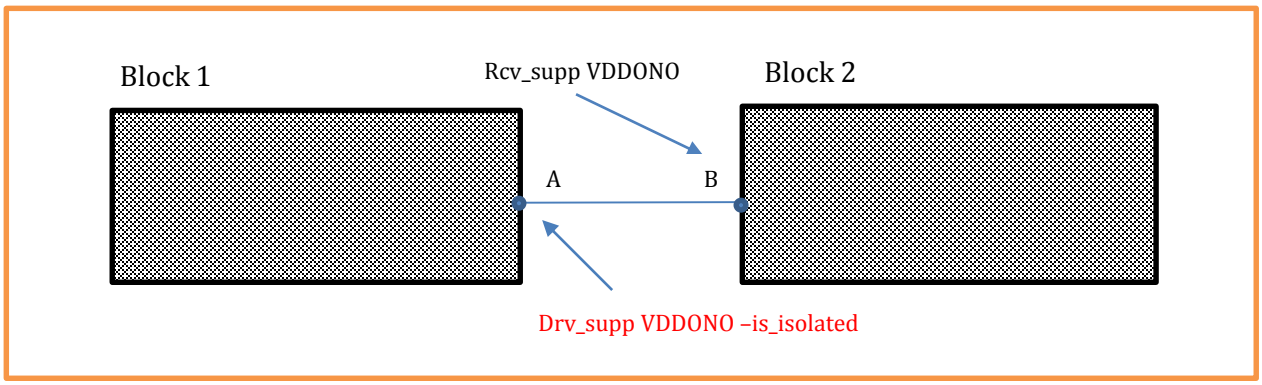

Figure 14. ISO_INST_REDUND reported in the same black-box SoC-level run after –is_isolated enhancement

If additional SPA information "**`is_isolated`**" is provided on output port Block1/A when VC-LP sees an ISO, then it can trace backwards through the ISO till it hits the driver of the ISO – and generate its supply VDDONO as `-driver_supply` instead of the ISO supply. This then makes the very same crossover at SoC level, from black-box port Block1/A to black-box port Block2/B as VDDONO->VDDONO.
Now VC-LP sees an "isolated" path (implying there is an ISO inside the block) from A to B and will report an ISO_INST_REDUND violation, exactly like the Full Hierarchical SoC-level run.

AMD is working with Synopsys to further enhance the accuracy the black-box flow to support this "`is_isolated`" SPA feature.

2. In a Full Hierarchical run with a feedthrough block between 2 blocks, VC-LP will analyze it as 2 separate crossovers when the feedthrough block is black-boxed.

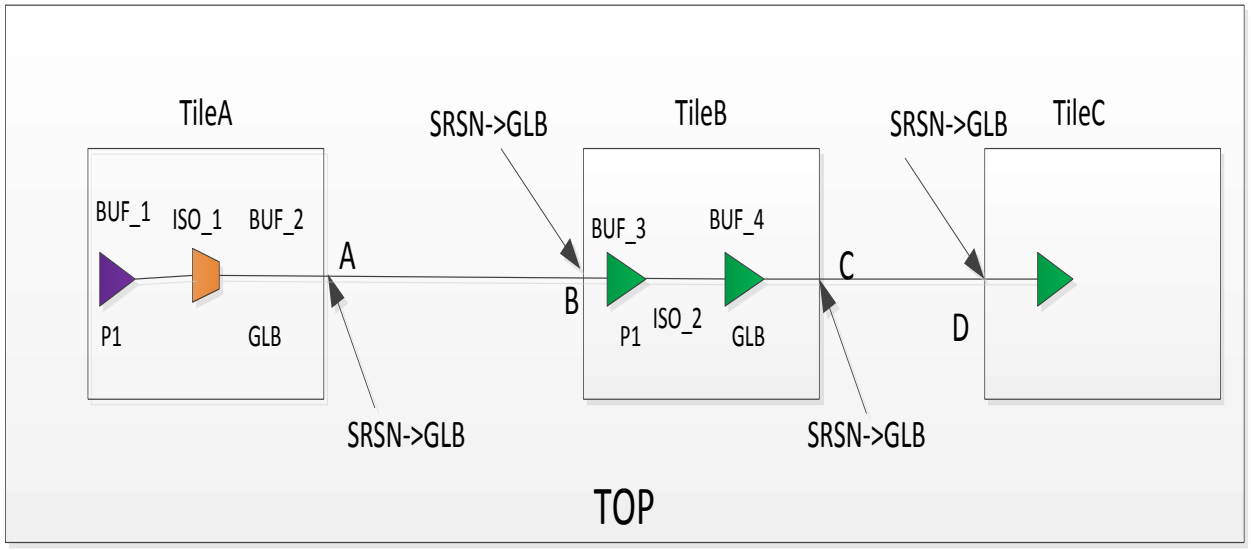

Figure 15. Feedthrough block in Full Hierarchical SoC-level run

In the above Figure, the block in the middle is a feedthrough block and can have buffers and ISO cells, but no other logic. VC-LP analyzes the long crossover from "TileA/P1-BUF_1-output" to "TileC/GLB-buf-input".

When black-boxed, VC-LP now sees 2 separate crossovers A->B and C->D as shown below. The resulting analysis could result in different violations from the Full Hierarchical run.

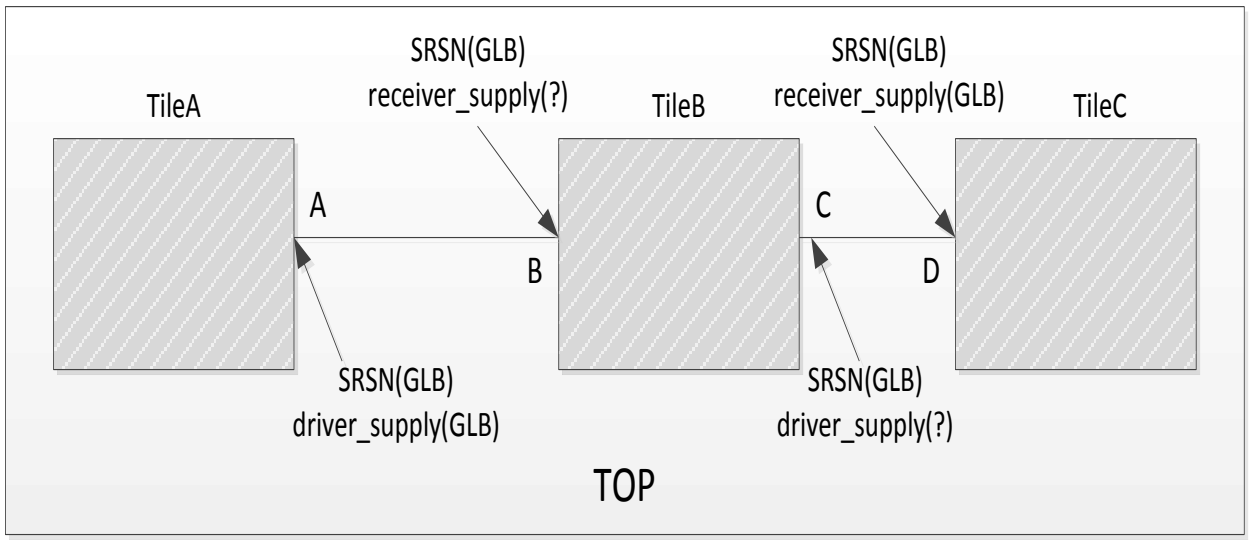

Figure 16. Feedthrough block in black-box SoC-level run

To prevent this, AMD is working with Synopsys to enhance the SPA generation to support –feedthrough connections that indicate the port feedthrough connections.

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

6. Acknowledgements

Special thanks to the Synopsys AC/AE team, including Vishwanath Sundararaman, Nishant Patel, Leo Fang, and Tareq Altakrouri for the help they provided during development of the enhanced black-box flow.

Thank you to AMD managers/colleagues Jean Fei, Ed Bender, and Tayeb Bouguerba for providing additional technical help when using this tool inside AMD.

7. References

[1] Verdi Signoff-LP User Guide, Synopsys, 2014-2016

[2] Verdi Signoff-LP Command Reference, Synopsys, 2014-16

[3] Verdi Signoff-LP Low Power Message Reference, Synopsys, 2016

[4] Application Notes and Information provided from Synopsys AE's Vishwanath Sundararaman, Nishant Patel, and Leo Fang