2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

UPF: How to avoid traps in a Hierarchical Implementation Low Power flow?

Frederic Saint-Preux, STMicroelectronics, Grenoble, France (*frederic.saint-preux@st.com*)

***Abstract*—The increasing complexity of Low Power technology has obliged to adopt new functionalities introduced in UPF2.0 (IEEE 1801™-2009). For example, the *filters* for strategies (-source/-sink/-diff_supply_only) ease the designer work but raise serious concerns in a Hierarchical Implementation flow where blocks are implemented separately from the top level. How to be sure that there are no differences between the Flat Verification and the Hierarchical Implementation? How to be sure that there will be no electrical error when integrating at top level the netlist of the block that has been implemented separately? The UPF3.0 (2015) addresses this challenge by introducing the concept of *soft macro*. However, this concept is not supported by all tools. This paper will detail the concern, the solution brought by the UPF3.0, the approach to cope with the partial support in the tools and the need to check the black box model used when implementing the top level.**

Keywords—IEEE1801, UPF, soft macro, Liberty, supply attributes

I. INTRODUCTION

The introduction of complex power structures in multi-voltage and multi-domains SOCs such as active body biasing and the constraints brought by the floorplan have led to consider UPF2.0 (IEEE 1801™-2009) which extends the features of UPF1.0. UPF2.0 can also drastically improve the reconciliation of the power intent description against a new version of the RTL. Instead of dealing with explicit elements at the interface of the power domains to define isolation and level shifting strategies as in UPF1.0, the user can specify *filters* which will ensure that the strategies will always apply to the right elements in regards to the driver and receiver supplies.

To allow higher level of concurrent engineering, it could be wise to implement blocks/IPs separately from the top level. This Hierarchical Implementation flow requires for use at top level a Low Power interface model for such a block; this model could be an "interface UPF" or a *Liberty* file with Low Power attributes. In most cases, the Verification does not see these models as the Verification often runs flat with all the design elements visible (HDL + "full UPF" for blocks/IPs). Thus, there could be some discrepancies between the Verification and the Implementation because the UPF *filters* may not see the same supplies at the interface of the blocks: actual supplies in Verification, UPF supply attributes or *Liberty* supply attributes in Implementation. This paper will first describe what a Hierarchical Implementation flow implies in terms of UPF power intent definition (hierarchical composition) and will show some examples where the design seen in the Verification is different from the one seen in the Implementation. The paper will then describe the solution brought by the IEEE 1801™ standard: the attribute *UPF_is_soft_macro* and its associated semantics. Since this solution is currently not supported by all tools, it is crucial to implement a new check in the Low Power static checkers, explained in this paper. Furthermore, even with the *soft macro* feature, another trap could happen in case a *Liberty* model is used as the Low Power interface model. The supply attributes in the *Liberty* model (*related_power_pin/related_ground_pin*) may come from a separate user specification beside the UPF; how to anticipate any issue that could occur when integrating the netlists of the blocks at top level, i.e. when switching from the *Liberty* supply attributes to the actual netlist supplies? The paper will discuss this.

II. UPF FOR HIERARCHICAL IMPLEMENTATION

A Hierarchical Implementation requires a self-contained UPF for the block that is implemented separately from the top level. This block UPF must contain all the information needed for standalone Verification and

Implementation. Particularly, the external constraints must be modelled by the *UPF_driver_supply* attribute (set by `set_port_attributes -driver_supply`) for the primary inputs and by the *UPF_receiver_supply* attribute (set by `set_port_attributes -receiver_supply`) for primary outputs of the block. In the following, we will name these attributes the *external supply attributes*. This UPF is loaded in the top UPF through `load_upf -scope` and the integration is done by connecting the supply ports:

```
load_upf -scope blk_inst block.upf
connect_supply_net -ports blk_inst/supply_port_name supply_net_name
```

The `load_upf` command can be conditioned by a TCL variable to load either the block UPF for the Verification or the *Liberty* model for the Implementation. In both cases, the supply ports of the block are explicitly connected (`connect_supply_net` above): either connection to the supply ports created in the block UPF for the Verification or connection to the *pg_pins* of the *Liberty* model for the Implementation.

Example of setting of the *external supply attributes* in the block.upf:

```
set_port_attributes -elements {.} -applies_to inputs -driver_supply SS1
set_port_attributes -elements {.} -applies_to outputs -receiver_supply SS1
```

Once the block UPF is loaded at top level, these *external supply attributes* apply on hierarchical pins and are ignored in Verification since the actual source and sink are visible at top level.

III. UPF STRATEGY *FILTERS*

UPF2.0 allows improving the time efficiency for writing and reconciling the UPF against a new RTL release by introducing *filters* which ease the writing of the strategies. The first *filter* is `diff_supply_only` for the `set_isolation` command: `-diff_supply_only TRUE|FALSE`.

"**-diff_supply_only TRUE** is satisfied by any port for which the driving logic and receiving logic are powered by supply sets that do not match, or for which either driving or receiving or both supply sets cannot be determined. **-diff_supply_only FALSE** is satisfied by any port."

This construct is very useful to handle Always-On feedthroughs for disjoint power domains (see Figure 1)

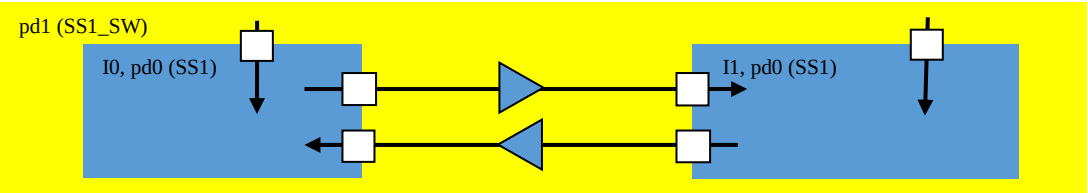

Figure 1: Disjoint power domains

The strategy to isolate all the inputs of pd0 except those for signals in-between I0 and I1 is very simple:

```
set_isolation pd0_iso_in -domain pd0 -applies_to inputs -isolation_supply_set SS1 \
-diff_supply_only TRUE
```

In UPF1.0, the user would have written two strategies: a default one with `-applies_to inputs` and a specific one with `-no_isolation` for the signals in-between I0 and I1 to avoid redundant isolation.

Furthermore, instead of dealing with elements at the boundary interface of a power domain and hard coding port names as the argument of `-elements` of `set_isolation` and `set_level_shifter`, as in UPF1.0, the user can consider the driver supply at the source and the receiver supply at the sink and specify *filters* which will select elements according to the driver and the receiver supplies, ensuring that the electrical constraints in terms of isolation, level shifting and repeater insertions are satisfied on the different RTL/netlists all along the flow. The semantics of this second filtering feature for `set_isolation`, `set_level_shifter` and `set_repeater` is:

```
-source <source_domain_name | source_supply_ref>  The name of a supply set or power domain
-sink <sink_domain_name | sink_supply_ref>        The name of a supply set or power domain
```

For example for Figure 1:

```
set_isolation pd0_iso_in -domain pd0 -isolation_supply_set SS1 \
-clamp_value 0 -source SS1_SW -sink SS1
```

IV. DRIVER/RECEIVER SUPPLY ANALYSIS FOR STRATEGY APPLICATION

Figure 2 describes the driver/receiver supply analysis into the block when it is implemented in standalone (the external context in grey colour is not seen): the *UPF_driver_supply* attribute is used as a source supply for a block primary input whereas the *UPF_receiver_supply* attribute is used as a sink supply for a block primary output.

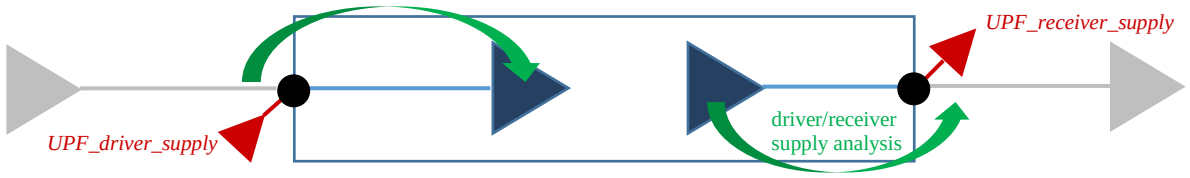

Figure 2: Driver/receiver supply analysis into the block

Figure 3 describes the driver/receiver supply analysis at top level when the block is modelled by a *Liberty* model (grey box). Here we assume that the *Liberty* supply attributes *related_power_pin/related_ground_pin* (named *rpgp*) are in line with the UPF *external supply attributes*; we will come back to this point later in the paper.

Figure 3: Driver/receiver supply analysis at top level with black-boxed block

Figure 4 describes the driver/receiver supply analysis at top level when all the design is seen. As previously said, the *external supply attributes* for the block, now applied on hierarchical pins, are ignored and the driver/receiver analysis is performed using the actual source and sink supplies in the design.

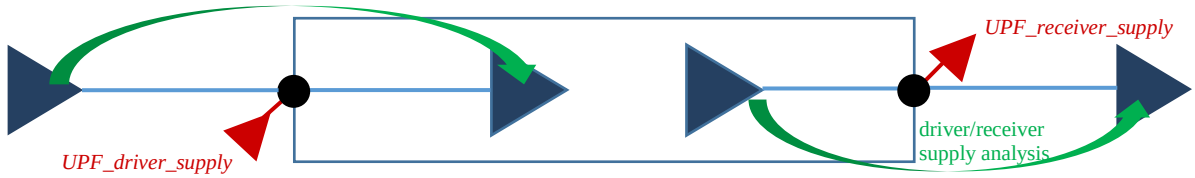

Figure 4: Driver/receiver supply analysis within the whole design

The driver/receiver supply analysis differs between the two last schemes and this may lead to discrepancies in the strategy application as we will see in the following examples.

V. DISCREPANCIES IN STRATEGY APPLICATION

Figure 5 exhibits a first example where the strategy application is different between the Verification and the Implementation.

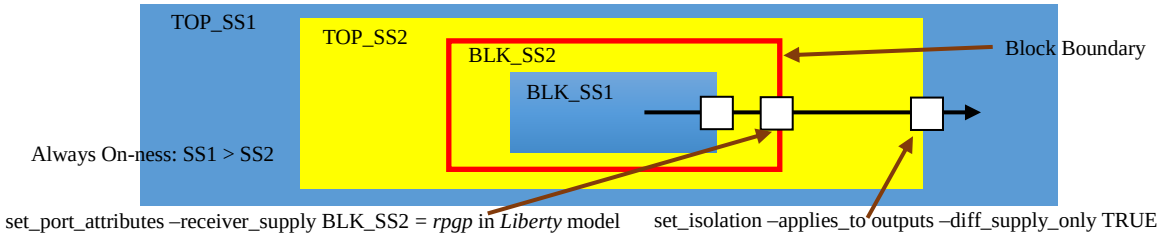

Figure 5: Strategy application discrepancy: Example 1

Applying the schemes of driver/receiver supply analysis in Figure 3 and Figure 4 for the isolation strategy:
Top level Implementation: using a *Liberty* model, the source supply is SS2 (*rpgp*) and the sink supply is SS1; the `diff_supply_only` *filter* is satisfied, the strategy applies and an isolation cell is inferred.

Verification: as the whole design is seen, the source supply is SS1 and the sink supply is SS1; the `diff_supply_only` *filter* is not satisfied, the strategy does not apply and no isolation cell is inferred.

An isolation cell is inferred during the Implementation whereas it is not simulated at RTL level. If the power intent for this path is to get an isolation cell and possibly buffers supplied by SS2 in the yellow power domain at top level, then it may be an issue since the simulation at RTL level is "best case" in the sense that the path is considered as always on (i.e. not broken by an isolation cell). Of course, an issue may be caught during the simulation of the *pg netlist* (netlist containing the power/ground nets and the power network). A `set_repeater` directive inferring a buffer powered by SS2 in the yellow power domain at top level could fix this discrepancy (see Figure 6 where, for Verification, the source becomes SS2 and so is different from the sink SS1), but the user has to know where to adequately put the `set_repeater` directive.

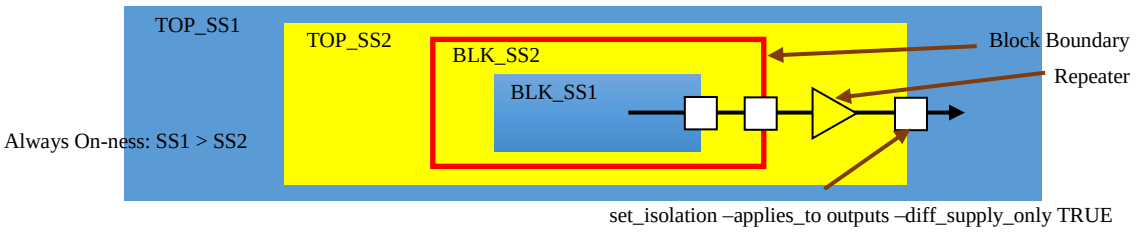

Figure 6: Example 1: repeater to fix the issue

Figure 7 shows another example where the strategy application is different between the Verification and the Implementation.

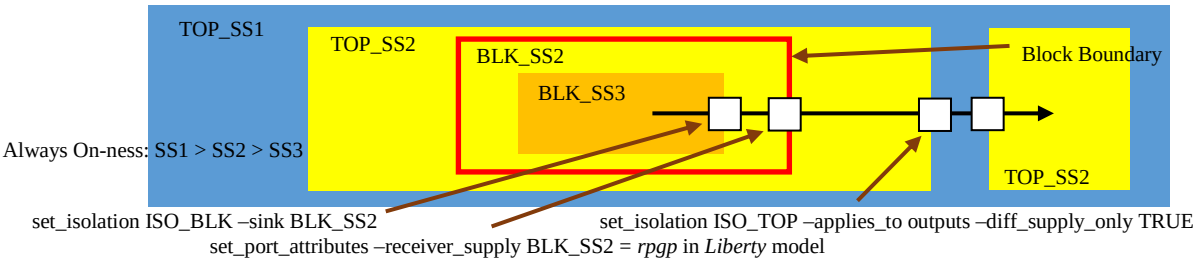

Figure 7: Strategy application discrepancy: Example 2

Applying the schemes of driver/receiver supply analysis in Figure 2, Figure 3 and Figure 4 for the two strategies:
Block level Implementation: the sink supply is SS2 (*UPF_receiver_supply*); the `sink` *filter* is satisfied, the strategy ISO_BLK applies and an isolation cell is inferred.
Top level Implementation: the source supply is SS2 (*rpgp*) and the sink supply is SS2; the `diff_supply_only` *filter* is not satisfied, the strategy ISO_TOP does not apply and no isolation cell is inferred.
Verification: as the whole design is seen, for ISO_TOP, the source supply is SS3 and the sink supply is SS2; the `diff_supply_only` *filter* is satisfied, the strategy applies and an isolation cell is inferred. For ISO_BLK strategy, the sink is SS2; the `sink` *filter* is satisfied, the strategy applies and an isolation cell is inferred.

Again, the Verification and the Implementation do not see the same logic: only one isolation cell is implemented (ISO_BLK) whereas two are simulated at RTL level. If the power intent for this path is to get only one isolation cell, the one into the block, then the simulation at RTL level is "worst case" (path from SS2 to SS2 broken by an additional isolation cell) and can catch the issue… or not. A `set_repeater` directive inferring a buffer powered by SS2 in the yellow power domain at top level could fix this discrepancy. To fix it, the user must know the issue.

As a conclusion, in a Hierarchical Implementation flow, the way the UPF is written in terms of *filters* could lead to discrepancies between the Verification and the Implementation. How to detect them at an early stage to fix them? Or how to avoid potential differences in the strategy application between the Verification and the Implementation? The IEEE1801-2015 answers the second question by the introduction of the concept of *soft macro*.

VI. IEEE 1801™ *SOFT MACRO*

The definition of a *soft macro* in the standard is:

"An **instance** that is represented by the original register transfer level (RTL) and Unified Power Format (UPF) from which its implementation is (or will be) derived. Additionally, **ancestor** power intent objects are not available for use within the **scope** of the **instance**."

"A soft macro instance is considered to have a terminal boundary that restricts the scope of the object. As such, power intent objects expressed in an ancestor (such as domains and global supply sets) are not available to the block and therefore the power intent must be supplied explicitly (self-contained UPF)".

The attribute *UPF_is_soft_macro* identifies a *soft macro*; this attribute is set either by:

```
set_design_attributes -models blk_inst -is_soft_macro
```
or by:
```
set_design_attributes -models blk_inst -attribute {UPF_is_soft_macro TRUE}
```

In addition to the *external supply attributes*, new attributes must be defined for a *soft macro*:
-the *UPF_receiver_supply* attribute (set by `set_port_attributes -receiver_supply`) for the primary inputs
-the *UPF_driver_supply* attribute (set by `set_port_attributes -driver_supply`) for the primary outputs.

These attributes will characterize the sink and source supplies for top level paths to/from the ports of the *soft macro*. In the following, we will name these attributes the *internal supply attributes*. The *external and internal supply attributes* must be defined for each block primary port.

Example of setting of the *internal supply attributes* in the block.upf:

```
set_port_attributes -elements {.} -applies_to inputs -receiver_supply SS1
set_port_attributes -elements {.} -applies_to outputs -driver_supply SS1
```

The definition of these *internal supply attributes* will resolve the issue of discrepancy in the strategy application between the Verification and the Implementation.

Figure 8 describes the driver/receiver supply analysis for a *soft macro* within the whole design. For the strategy application within the block, the *UPF_driver_supply* attribute is still used as the source supply for the block primary inputs, whereas the *UPF_receiver_supply* attribute is still used as the sink supply for the block primary outputs. However, now, for the strategy application at top level, the *UPF_receiver_supply* attribute is used as the sink supply (block primary inputs), whereas the *UPF_driver_supply* attribute is used as the source supply (block primary outputs).

Thus, in both Verification and Implementation, the strategy application stops at the boundary of the *soft macro*, which was not the case before (see Figure 4 where the UPF supply attributes are ignored). This ensure the consistency in the strategy application.

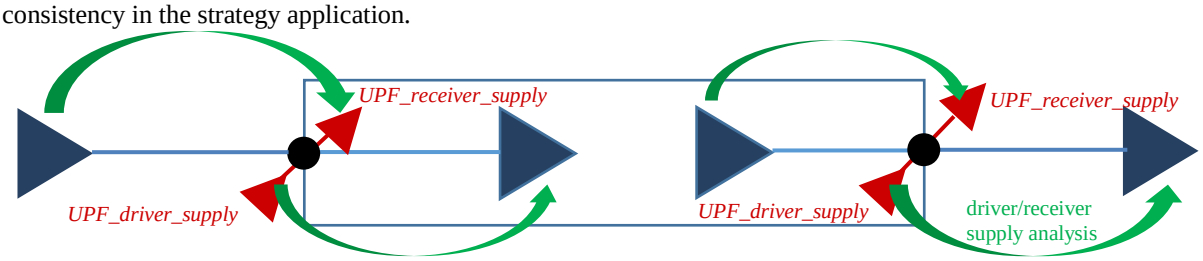

Figure 8: Driver/receiver supply analysis for a *soft macro*

Let us review the examples where the block has the attribute *UPF_is_soft_macro* and where both *external and internal supply attributes* are defined for each block primary port.

Example 1:

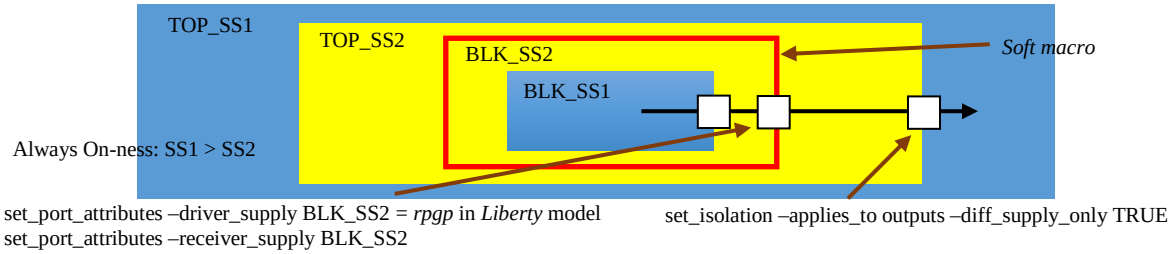

Figure 9: Example 1 with the block as a *soft macro*

Applying the new scheme of driver/receiver supply analysis in Figure 8:

Top level Implementation: using a *Liberty* model, the source supply is SS2 (*rpgp*) and the sink supply is SS1; the `diff_supply_only` *filter* is satisfied, the strategy applies and an isolation cell is inferred.

Verification: the source supply is now SS2 (*UPF_driver_supply*) and the sink supply is SS1; the `diff_supply_only` *filter* is satisfied, the strategy applies and an isolation cell is inferred.

An isolation cell is inferred in both Verification and Implementation.

Example 2:

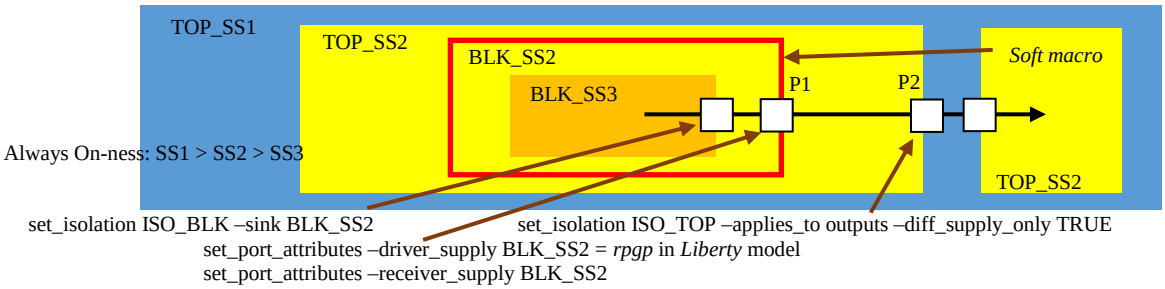

Figure 10: Example 2 with the block as a *soft macro*

Applying the new scheme of driver/receiver supply analysis in Figure 8:

Block level Implementation: the sink supply is SS2 (*UPF_receiver_supply*); the `sink` *filter* is satisfied, the strategy ISO_BLK applies and an isolation cell is inferred.

Top level Implementation: the source supply is SS2 (*rpgp*) and the sink supply is SS2; the `diff_supply_only` *filter* is not satisfied, the strategy ISO_TOP does not apply and no isolation cell is inferred.

Verification: for ISO_TOP, the source supply is now SS2 (*UPF_driver_supply*) and the sink supply is SS2; the `diff_supply_only` *filter* is not satisfied, the strategy does not apply and no isolation cell is inferred. For ISO_BLK strategy, the sink is SS2; the `sink` *filter* is satisfied, the strategy applies and an isolation cell is inferred.

Only one isolation cell is inferred in both Verification and Implementation.

As a conclusion, the *soft macro* concept introduced in the IEEE 1801™-2015 answers the problem of potential discrepancies in the strategy application. Unfortunately this concept is not supported by all tools. A solution could be to check that the UPF strategies apply in the same way in the Verification and Implementation tools. It would require to generate for each tool a "strategy application" report, which could not be straightforward. Thus the preferred way would be to flag any discrepancies in the strategy application at an early stage. The ideal candidate to catch them is the Low Power Static Checker. Of course, the discrepancies must be fixed by either modifying the UPF *filters* (if possible) or adding repeaters as seen in the examples.

VII. NEW CHECK TO CATCH DISCREPANCIES IN STRATEGY APPLICATION

For this check, the block must be identified as a *soft macro* or by a specific design attribute so that a what-if driver/receiver supply analysis is performed at the boundary of the block. What happens if the *internal supply attributes* are considered for the strategy application at top level and if the *external supply attributes* are considered for the strategy application within the block? What happens if the *internal supply attributes* and the *external supply attributes* are NOT considered for the strategy application? Is there a difference?

The reported message could be for **Example 2** (see Figure 10):

The isolation strategy ISO_TOP does NOT apply on port P2 if the set_port_attributes –driver_supply SS2 on the port P1 is considered.

This check is crucial as long as the strategy application with *soft macro* is not supported by all tools.

VIII. *INTERNAL SUPPLY ATTRIBUTES* VERSUS ACTUAL SUPPLIES

At top level, the *internal supply attributes* specified in the block "interface UPF" act as source supplies for the paths from the block outputs and as sink supplies for the paths to the block primary inputs, as represented in Figure 11:

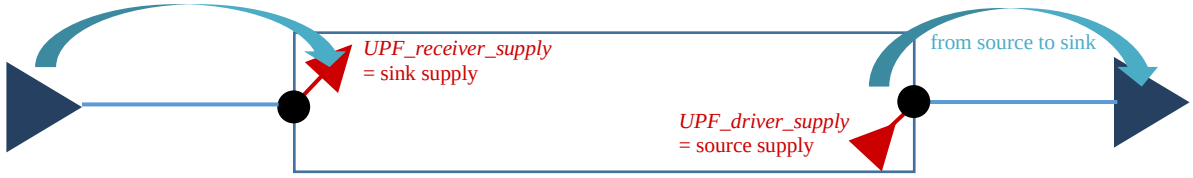

Figure 11: From source to sink to/from the black-boxed block

Beyond their use for the strategy application, these attributes are also, for example, considered for the buffering done at top level by the Implementation tools when the block is black-boxed.

It should be noted that these *internal supply attributes* are not used, and so not checked, when implementing the block separately, contrary to the *external supply attributes* which define the power intent of the block (source supplies for the primary inputs and sink supplies for the primary outputs). It is then mandatory to check the consistency of the *internal supply attributes* against the actual supplies within the block to avoid any issue when integrating at top level the netlist of the block that has been implemented separately.

Figure 12 shows a case where the attribute *UPF_driver_supply* for an output port is wrongly set and where an electrical issue occurs when switching from the black-box model with the block "interface UPF" to the netlist with the block UPF, i.e. when switching from the block *internal supply attributes* to the block actual supplies.

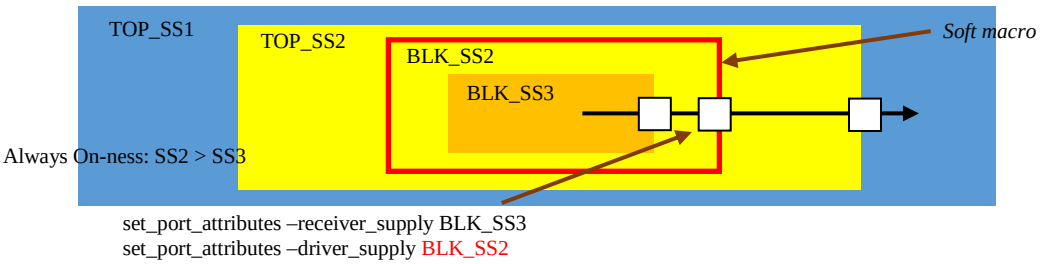

Figure 12: *UPF_driver_supply* wrongly set

In all the previous examples, for the given port, the UPF *external and internal supply attributes* are the same supply, but nothing precludes using different supplies for such attributes and it may be correct not to have same supplies. Here the attribute *UPF_receiver_supply* at SS3 defines the power intent within the block as a "direct path from BLK_SS3 to the primary output port". The *UPF_driver_supply* attribute is wrongly defined to SS2. Hence the Implementation tool at top level may insert buffers powered by SS2 from this output port. When integrating the netlist of the block at top level and switching to actual supplies (see Figure 13, where the supply attributes are

ignored since they do not have real existence after the Implementation), it may end up with an electrical violation: from SS3 to SS2 without any isolation cell.

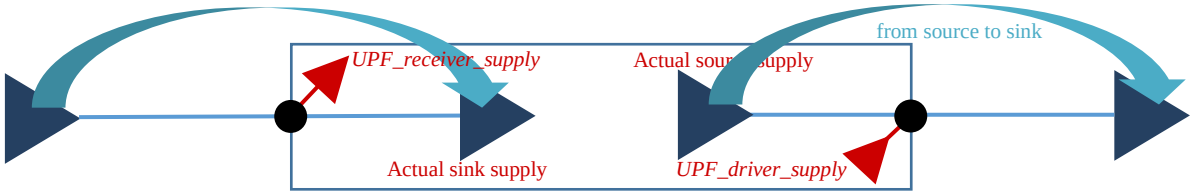

Figure 13: From source to sink at the boundary of the block within the whole design

The standard defines the check of the *internal supply attributes* (see Figure 14):

"For an output port with the attribute **UPF_driver_supply**, when that port has a single source and the driving logic is present within the model or instance whose port is being attributed, it shall be an error if the actual supply of the driving logic is not the same as, or equivalent to, the specified driver supply. The actual supply of the driving logic is the supply of the logic element driving this port after applying all strategies in the power intent, and therefore may be the supply of a retention cell, a repeater cell, an isolation cell, or a level-shifter cell inserted by such a strategy."

The same for the attribute *UPF_receiver_supply*.

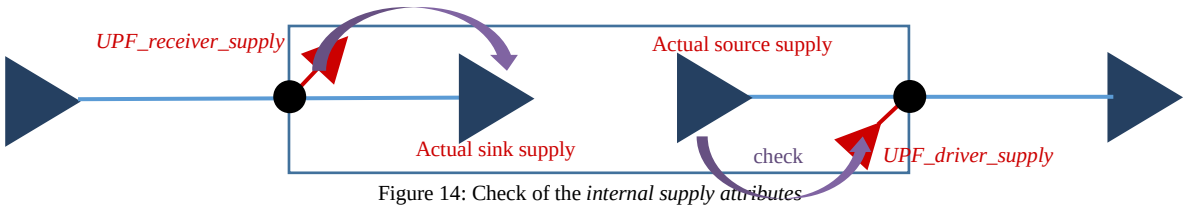

Figure 14: Check of the *internal supply attributes*

This check is mandatory at block level before implementing the top level (check at block RTL level) and before integrating the netlist of the block at top level (check at block netlist level) to anticipate any integration issue.

IX. *LIBERTY* SUPPLY ATTRIBUTES VERSUS ACTUAL SUPPLIES

The *Liberty* model is often used as the black box model in a Hierarchical Implementation flow. However, new parameters enter the picture: the *related_power_pin/related_ground_pin* attributes (*rpgp*) which must be consistent with the actual supplies within the block in the same way the UPF *internal supply attributes* must be. This is crucial in case the Low Power attributes are added in the *Liberty* model from a separate user specification beside the UPF.

This consistency check can actually be composed of two checks using the *internal supply attributes* as pivots: if the *rgpg* attributes and the *internal supply attributes* are the same and if the *internal supply attributes* and the actual supplies are the same (check defined in the IEEE 1801™ standard, as previously discussed), then the *rgpg* attributes and the actual supplies are the same. The ideal candidate for EDA tools to implement the first check (*rpgp* attributes against the *internal supply attributes*) is of course the Low Power Static Checker tool.

X. CONCLUSION

The Implementation and Verification tools must see the same objects at the interface of a block which is implemented separately from the top level, and these objects must be consistent with the actual objects after the block Implementation. Otherwise, traps such as discrepancies in the strategy application and electrical issues when integrating the netlist of the block could happen. As demonstrated, two new checks should be implemented in the Low Power Static Checker tool to avoid these issues. The first one reporting discrepancies in the strategy application will become useless once all the tools support the *soft macro* concept but is crucial in the meantime; the second one checking the *Liberty* attributes against the *internal supply attributes* is and will remain mandatory.

XI. REFERENCE

[1] IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems (IEEE 1801™-2015)