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

Frederic Saint-Preux, STMicroelectronics, Grenoble, France

Introduction (1)

- Use of UPF2.x (IEEE 1801™) to cop with increasing complexity of power structures in multi-voltage and multi-domains SOCs
  - Bias modeling with supply sets
  - Physical constraints modeling with repeaters and supply availability
  - Improvement of the reconciliation of the UPF against a new version of RTL with supply *filters* when defining strategies
- For higher level of concurrent engineering, Hierarchical Implementation
  - Block implemented separately from the top level
  - Block Low Power interface model for top level Implementation ("interface UPF"/Liberty)
- For top level Verification, all the design seen
  - Block Low Power interface model not used

Introduction (2)

- Couldn't there be differences between the Flat Verification and the Hierarchical Implementation?
  - Since *filters* may not see the same supplies at the interface of the blocks: actual supplies in top level Verification, UPF supplies or *Liberty* supply attributes in top level Implementation
- Solution in IEEE1801™-2015: *soft macro* concept
- How to cope with partial support of *soft macro* in tools?
- Couldn't there be electrical errors when integrating at top level the netlist of the block that has been implemented separately?
  - Since the block Low Power interface model may not be correct

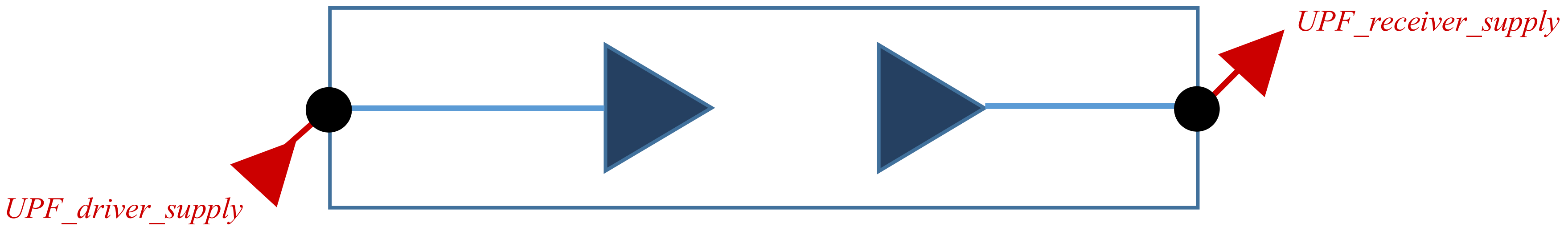
Agenda

- UPF for Hierarchical Implementation
- UPF strategy *filters*
- Driver/Receiver supply analysis for strategy application
- Discrepancies in strategy application: Example
- IEEE1801™ *Soft Macro*
- New check required for reporting discrepancies
- Check of block Low Power interface model
  - UPF supply attributes versus block actual supplies
  - Liberty supply attributes versus block actual supplies

UPF for Hierarchical Implementation

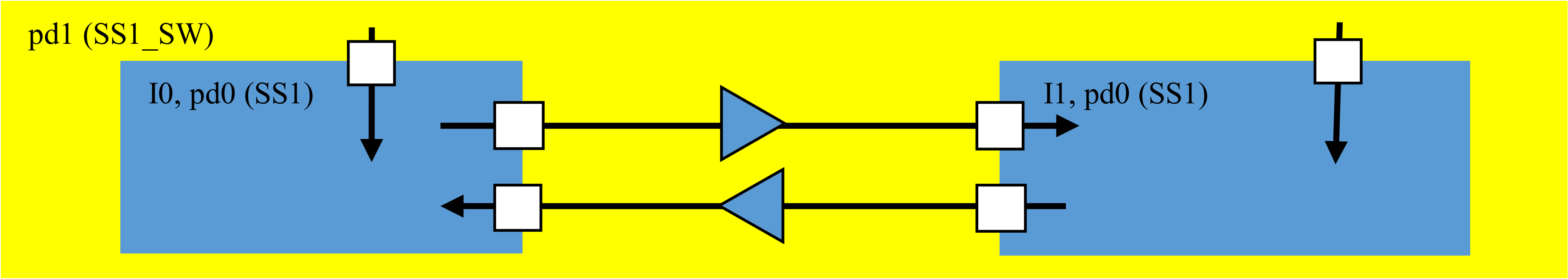
- Block self-contained UPF
  - All the information required for standalone Verification and Implementation
- External world to be modelled by *UPF_driver_supply* and *UPF_receiver_supply* attributes set by:
  - set_port_attributes –driver_supply <supply_set_ref> for block primary inputs
  - set_port_attributes –receiver_supply <supply_set_ref> for block primary outputs

  Named *external supply attributes*



UPF strategy *filters*

- For UPF set_isolation command
  - -diff_supply_only TRUE|FALSE
  - -source <source_supply_ref> / -sink <sink_supply_ref>



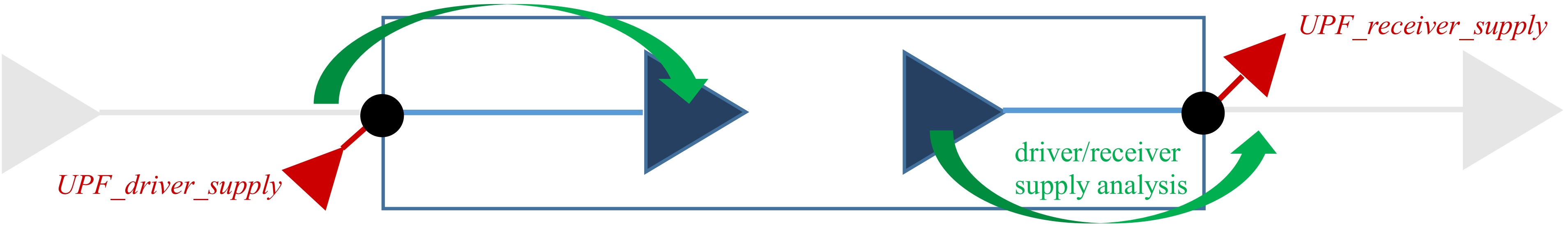
set_isolation pd0_iso_in -domain pd0 -applies_to inputs -isolation_supply_set SS1 \
-diff_supply_only TRUE

or

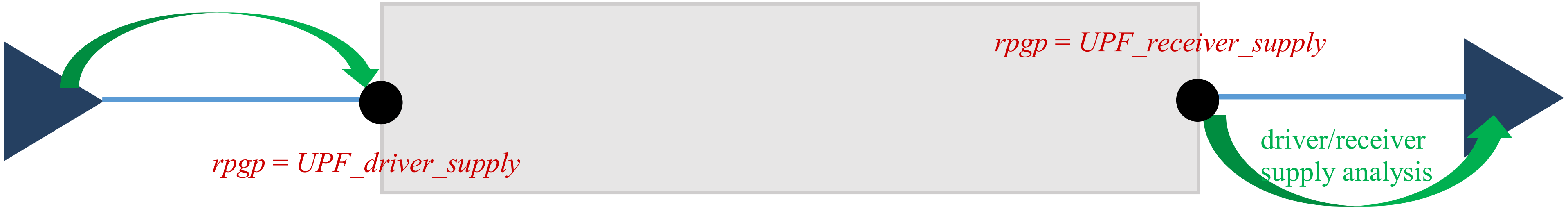
set_isolation pd0_iso_in -domain pd0 -isolation_supply_set SS1 -clamp_value 0 \
-source SS1_SW -sink SS1

Driver/Receiver supply analysis (1)

- Block standalone Verification and Implementation

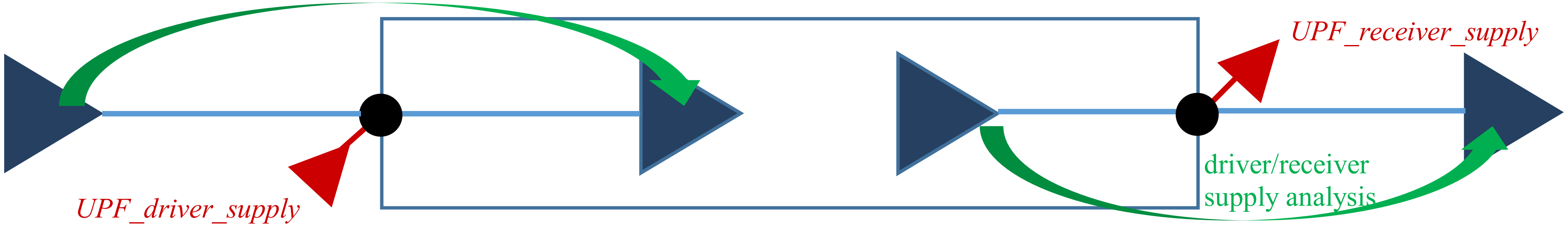


- Top level Implementation with Liberty as block model: Low Power interface modelled by *related_power_pin/related_ground_pin* attributes

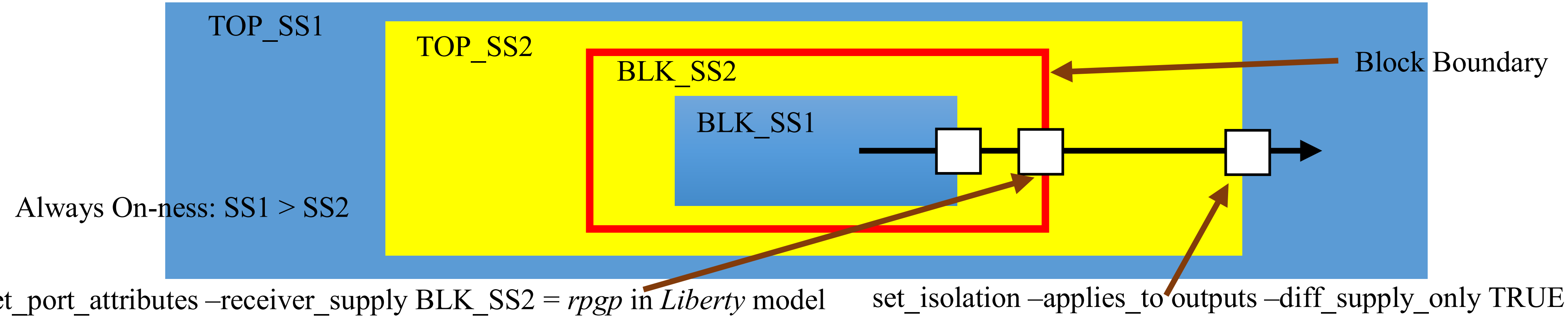


Driver/Receiver supply analysis (2)

- Top level Verification



Discrepancies in strategy application: Example



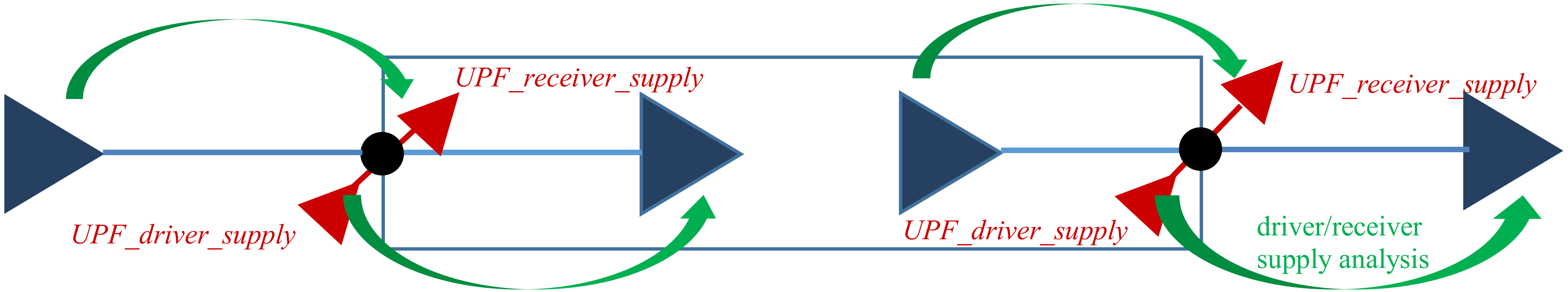
- Top level Implementation
  - source supply = SS2 (*rpgp*)
  - sink supply = SS1

  → the strategy applies and an isolation cell is inferred
- Top level Verification
  - source supply = SS1
  - sink supply = SS1

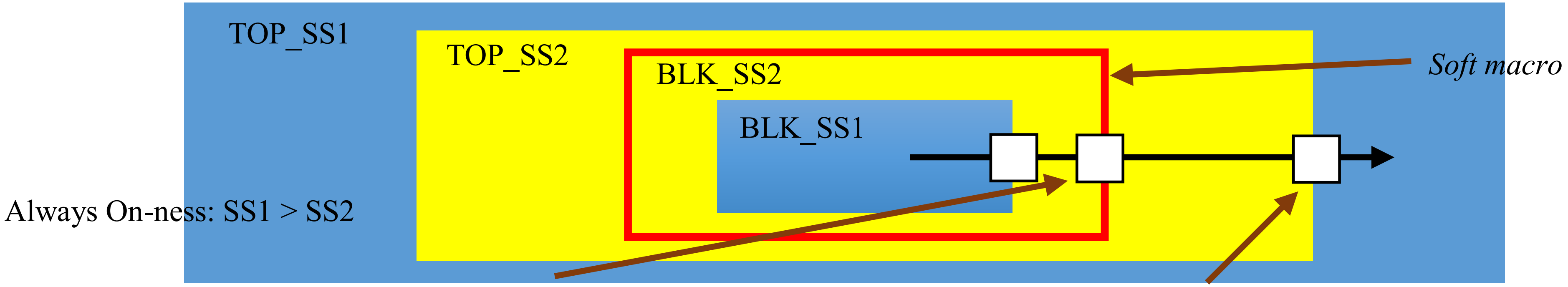
  → the strategy does not apply and no isolation cell is inferred

IEEE1801™ Soft Macro

- Attribute *UPF_is_soft_macro* specified on instances or models
- Specifies a “terminal boundary”: to stop at/start from the block boundary
- In the block UPF, in addition to *external supply attributes*, definition of *internal supply attributes* for a *soft macro*: *UPF_driver_supply* and *UPF_receiver_supply* set by:
  - set_port_attributes –receiver_supply <supply_set_ref> for block primary inputs
  - set_port_attributes –driver_supply <supply_set_ref> for block primary outputs



Example with block as Soft Macro



set_port_attributes –driver_supply BLK_SS2 = *rpgp* in *Liberty* model
set_port_attributes –receiver_supply BLK_SS2

set_isolation –applies_to outputs –diff_supply_only TRUE

- Top level Implementation
  - source supply = SS2 (*rgpg*)
  - sink supply = SS1

  → the strategy applies and an isolation cell is inferred
- Top level Verification
  - source supply = SS2 (*UPF_driver_supply*)
  - sink supply = SS1

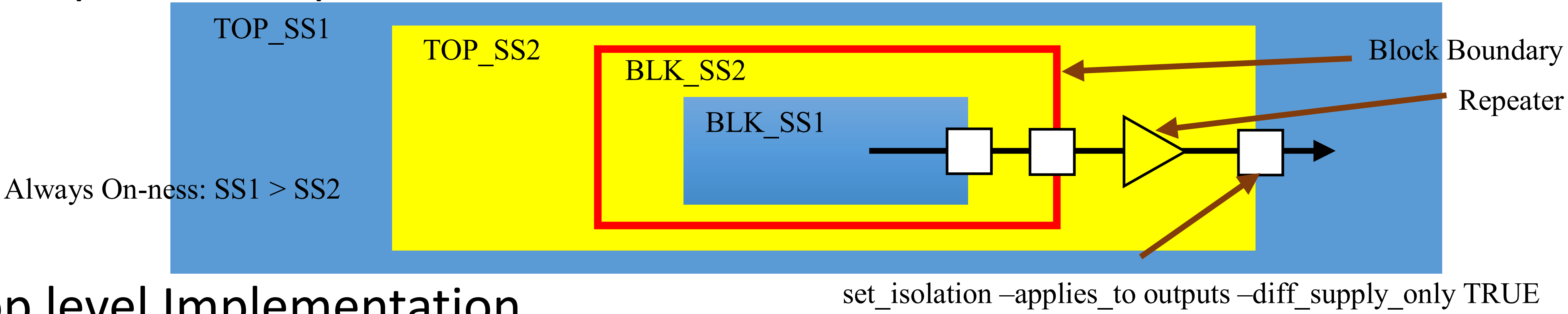
  → the strategy applies and an isolation cell is inferred

New check required

- Required if all the tools do not support *soft macro*
- Check to run with all the design seen
- What–if analysis
  - 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?
  - Are there any differences?
- If differences, to fix them. How?

To fix a discrepancy: Example

- To modify UPF strategies or to add repeaters (UPF set_repeater command)
- Example with repeater



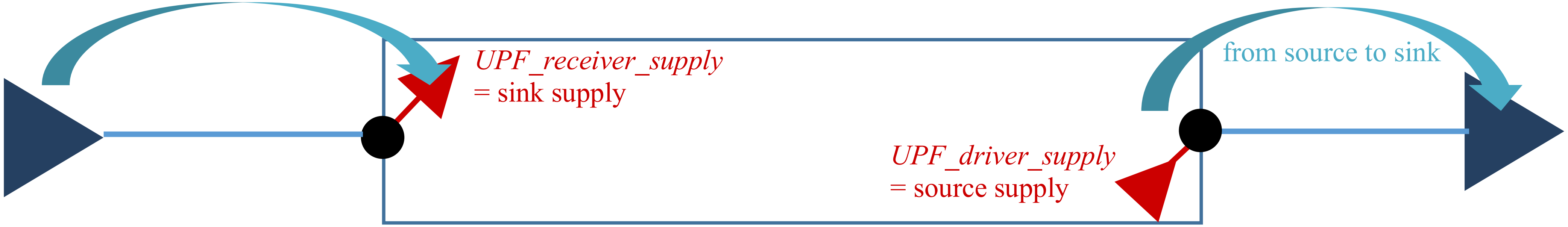
- Top level Implementation
  - source supply = SS2 (*repeater_supply*)
  - sink supply = SS1

  → the strategy applies and an isolation cell is inferred
- Top level Verification
  - source supply = SS2 (*repeater_supply*)
  - sink supply = SS1

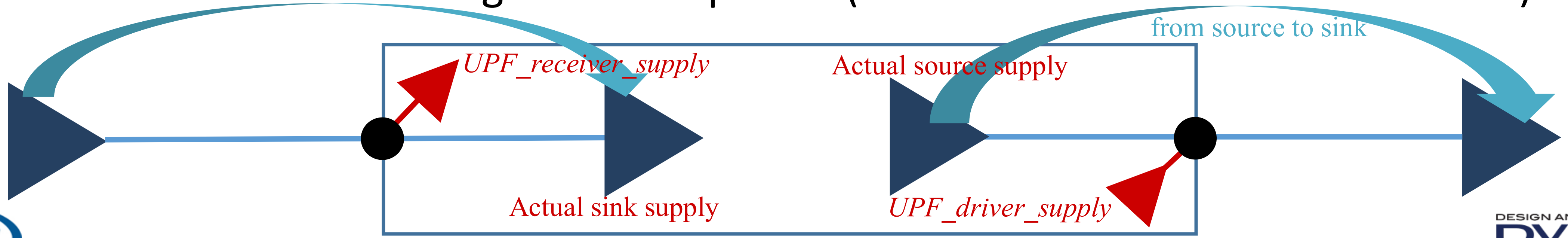
  → the strategy applies and an isolation cell is inferred

Internal supply attributes vs block actual supplies(1)

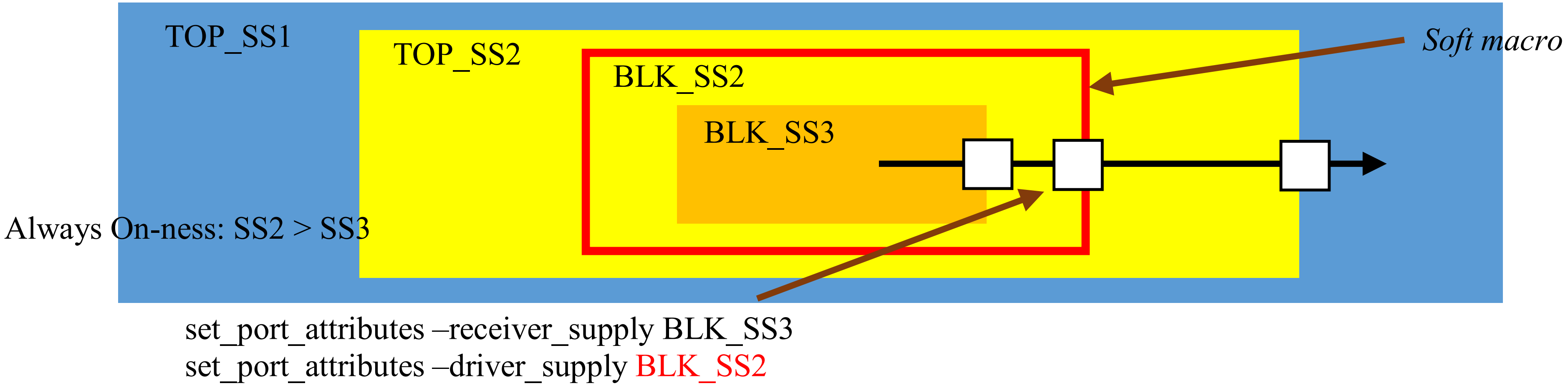
- *Internal supply attributes* in block "interface UPF" only used when implementing the top level (buffering):
  - Act as source/sink supplies for paths from/to the block



  - Are not used when implementing the block in standalone and are ignored once the block netlist is integrated at top level (use of actual drivers and receivers)



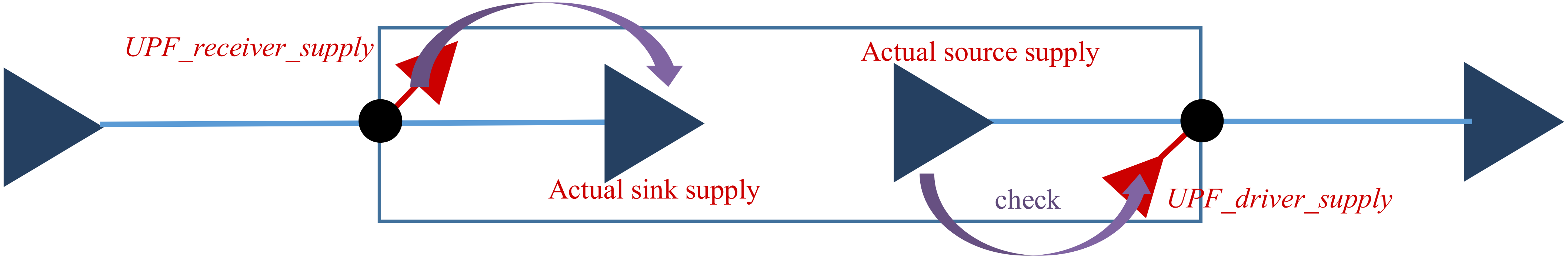
Internal supply attributes vs block actual supplies(2)



- Block power intent: direct path from "SS3 block" to the output port
- … but UPF_driver_supply wrongly set to SS2
  - During top level Implementation, buffers supplied by SS2 inserted from the block output
  - Once the block netlist is integrated, from SS3 (block actual supply) to SS2 (buffer supply) without any isolation cell

Internal supply attributes vs block actual supplies(3)

- Need to check the consistency between the *internal supply attributes* and the block actual supplies



- Consistency checks defined in IEEE1801™

Liberty supply attributes vs block actual supplies

- *Liberty* model often used as the black box model in a Hierarchical Implementation flow
- *related_power_pin/related_ground_pin* attributes (*rpgp*)
  - Can be added in the *Liberty* model from a separate user specification beside the UPF
  - Must be consistent with the block actual supplies
- Consistency check required
  - Can be composed of two checks using the *internal supply attributes* as pivots:
    - *rgpg* attributes against *internal supply attributes*
    - *Internal supply attributes* against actual supplies (IEEE1801™ consistency checks)

Conclusion

- To avoid traps in a Hierarchical Implementation Low Power flow
  - Implementation and Verification tools must see the same objects at the interface of a block which is implemented separately from the top level
  - These objects must be consistent with the actual objects after the block Implementation
- To ensure this, block specified as *soft macro* + three kinds of checks
  - If *soft macro* not supported by all tools, reporting of any discrepancies in the strategy application considering or not the *internal* and *external supply attributes*
  - Reporting of any inconsistencies between *internal supply attributes* and block actual supplies
  - If Liberty model used for top level Implementation, reporting of any inconsistencies between the *Liberty* attributes and the *internal supply attributes*

Questions

Which tool for implementing these checks?

When to run these checks?