2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

Introduction of IEEE 1801-2024 (UPF4.0) improvements for the specification and verification of low-power

John Decker, Daniel Cross – Cadence Design Systems
Amit Srivastava – Synopsys
Lakshmanan Balasubramanian - Texas Instruments

accellera
SYSTEMS INITIATIVE

Agenda

- Introduction
- Interconnect between UPF supplies and arbitrary HDL types
- Improvements in successive refinement and refinable macros
- Overview of Retention Changes
- Virtual Supply and Virtual Equivalence
- General Updates
- Beyond 4.0
- Q/A

Introductions/Acknowledgements

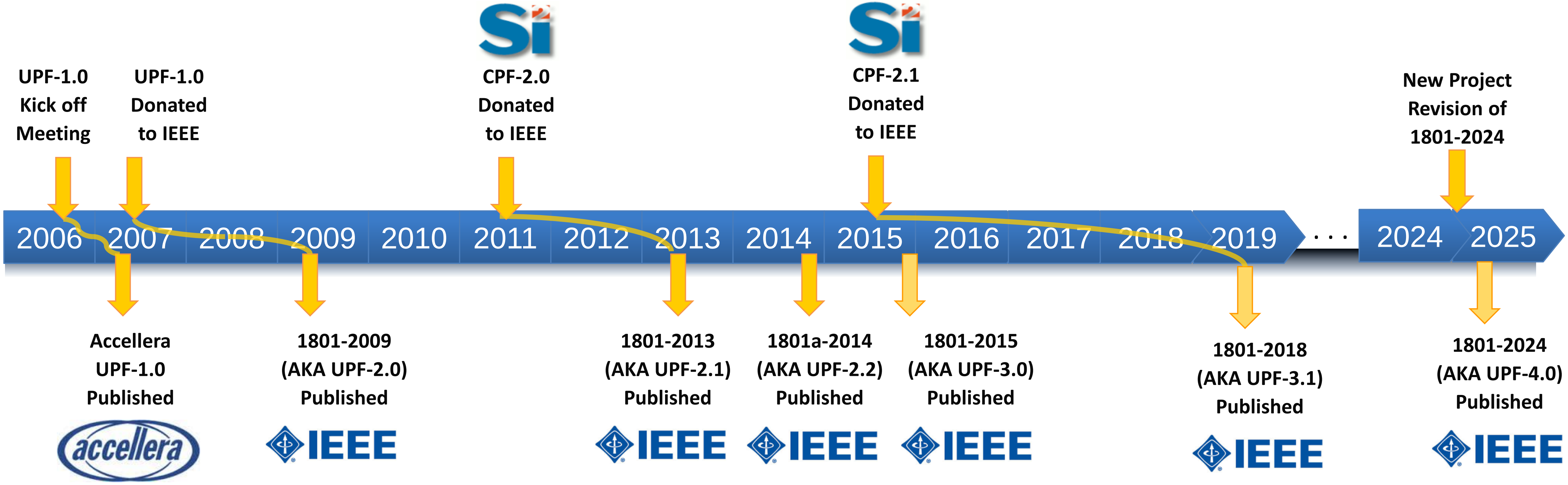
- Presenters
  - John Decker : 1801 WG Chair - Cadence Design Systems
  - Amit Srivastava : 1801 WG Vice-Chair - Synopsys
  - Daniel Cross : Cadence Design Systems
- Contributors
  - Lakshmanan Balasubramanian : 1801 WG secretary, Texas Instruments
  - Marcelo Glusman – Cadence, Paul Bailey - Nordic Semiconductor,
  - Rick Koster - Siemens EDA, Progyna Khondkar - Cadence
- Special thanks to the 1801 WG
  - Over 40 members representing 16 companies
  - Former members John Biggs(previous Chair), Phil Giangarra, David Cheng
- Thanks to IEEE Standards Association and Accellera
  - This presentation solely represents the views of the author(s), and does not necessarily represent a position of either the IEEE P1801 Working Group, the IEEE Design Automation Standards Committee, IEEE or the IEEE Standards Association.
  - Design Automation Standards Committee of the IEEE Computer Society, “IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems”, IEEE Std. 1801™-2024

Unified Power Format (UPF)

- IEEE Standard for expressing Power Intent
  - To define power architecture and power management control
  - To minimize power consumption
  - Enables a consistent representation of power intent across all aspects of the design and verification flow
  - Enables early verification of power intent
- An Evolving Standard
  - 6 versions of UPF over ~18 years
  - Donations from Accellera UPF1.0 and Si2 CPF 2.0 and 2.1
  - 1801-2024 had contributions from more than 20 chip design and EDA companies

- Based upon Tcl
  - Tcl syntax and semantics
- And HDLs
  - SystemVerilog, VHDL, SystemC
- For Verification
  - Simulation, Emulation, Static/Formal
- For Implementation
  - Synthesis, DFT, P&R, etc.
- And for System Level Power Modeling
  - Abstract power models with power_expr

Evolution of the Standard



UPF-1.0 Kick off Meeting

UPF-1.0 Donated to IEEE

CPF-2.0 Donated to IEEE

CPF-2.1 Donated to IEEE

New Project Revision of 1801-2024

Accellera UPF-1.0 Published

1801-2009 (AKA UPF-2.0) Published

1801-2013 (AKA UPF-2.1) Published

1801a-2014 (AKA UPF-2.2) Published

1801-2015 (AKA UPF-3.0) Published

1801-2018 (AKA UPF-3.1) Published

1801-2024 (AKA UPF-4.0) Published

UPF 4.0 – Major Goals

- Enable low power simulation with mixed-signal features like real number modeling
- Enable accurate modeling of Retention
- Enhance IP design reuse with refinable macros
- Improve successive refinement flow
- Address over 200 Mantis items tracking improvement requests.
    - Enable Virtual Supplies
    - Updates/Clarifications to semantics
    - Ease-of-use features

Summary of Change Topics

- New Concepts
  - Refinable Macro
  - Implementation UPF
  - Virtual nets/ports/sets/equivalence
  - Tunneling
  - Connections to real (VCM)
- Major Updates
  - Power distribution section 4.5.1
  - Simulation of state retention (9.7)
  - Annex I – VCM usage examples
  - Supply equivalence
- Clarifications (partial list)
  - Major improvements to Definitions
  - Resolved elements list
  - Literal supply
- Open SA repository

- Precedence
  - SPA, retention, composite types, Macros
- Naming Related
  - Rooted vs Simple name clarifications
  - Escaped naming styles
  - Generate block delimiter
  - Library name (5.3.3.2)
- Complete update of Annex E (example)
- Command Updates
  - **`set_port_attributes –is_analog`** allowed on instance pins
  - **`set_port_attributes –feedthrough`** improvements
  - **`set_design_attributes`** with no object creates a “UPF“ wide attribute
  - **`set_repeater –repeater_supply`** mandatory

Command/Option Change Summary

New Options
set_isolation –async_set_reset -async_clamp_value
connect_supply_net –vcm -tunneling
set_port_attribute -is_refinable_macro -async_clamp_value
load_upf -implementation
define_power_model –update -implementation -complete
create_supply_net -virtual
create_supply_port –virtual
create_supply_set -virtual
set_retention -applies_to {latch ff both} –restore_period_condition -powerdown_period_condition -restore_event_condition -save_event_condition
find_objects –expand_to_bits

New Commands
create_vcm
create_upf_library
load_upf_library
use_upf_library
map_retention_clamp_cell
create_abstract_power_source

Legacy/Deprecated
set_isolation –applies_to_sink_clamp -applies_to_source_clamp
create_supply_net -reuse -domain
create_power_switch -domain
create_supply_port -domain
set_port_attribute –sink_off_clamp -source_off_clamp
create_upf2hdl_vct
create_hdl2upf_vct
set_retention_elements -transitive
set_retention –save_condition -restore_condition –retention_condition

Where to get the IEEE 1801-2024 Spec

- 1801-2024 spec is available from the IEEE GET program

- IEEE SA open repository
  - Select examples and packages from the 1801-2024 specification
  - Planned community space to provide comments, advice, additional examples
  - Available at: https://opensource.ieee.org/upf

2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

New feature for UPF 4.0: interconnect between UPF supplies and arbitrary HDL types

Daniel Cross – Cadence Design Systems

accellera
SYSTEMS INITIATIVE

Motivation for adding to the standard

- An increasing number of designs are mixed signal in nature and have significant analog and mixed signal content
- Co-verification of analog and mixed signal design elements with purely digital components has increased in importance
- Analog and digital portions of designs increasingly share power supplies
- Use of Real Number Modeling (RNM) to represent analog functions for verification has proliferated
- Synchronization of analog and UPF representations of the power supply network has become critical

New concepts

- **VCM – Value Conversion Method**

  extends and enhances VCT (Value Conversion Table)

- **Tunneling**

  allows analog connections to be made via UPF

- **UPF Library**

  helps avoid name collisions between otherwise global objects

- **Automatic VCM selection by nettype and data type**

  allows successive refinement by supporting multiple representations with a single UPF

VCMs

- VCMs provide a richer and more flexible mechanism to translate between UPF supply nets and HDL
  - Improved simple table conversions vs VCT
  - Enable advanced conversions of more complex types by using user defined Modules and Functions
  - Ability to contain a list of other VCMs to enable automated type conversions
- VCTs are now legacy
  - Their functionality is a subset of VCMs
  - As legacy they continue to work but 1801 encourages migration
- Examples of each type provided at end of this presentation

Syntax:

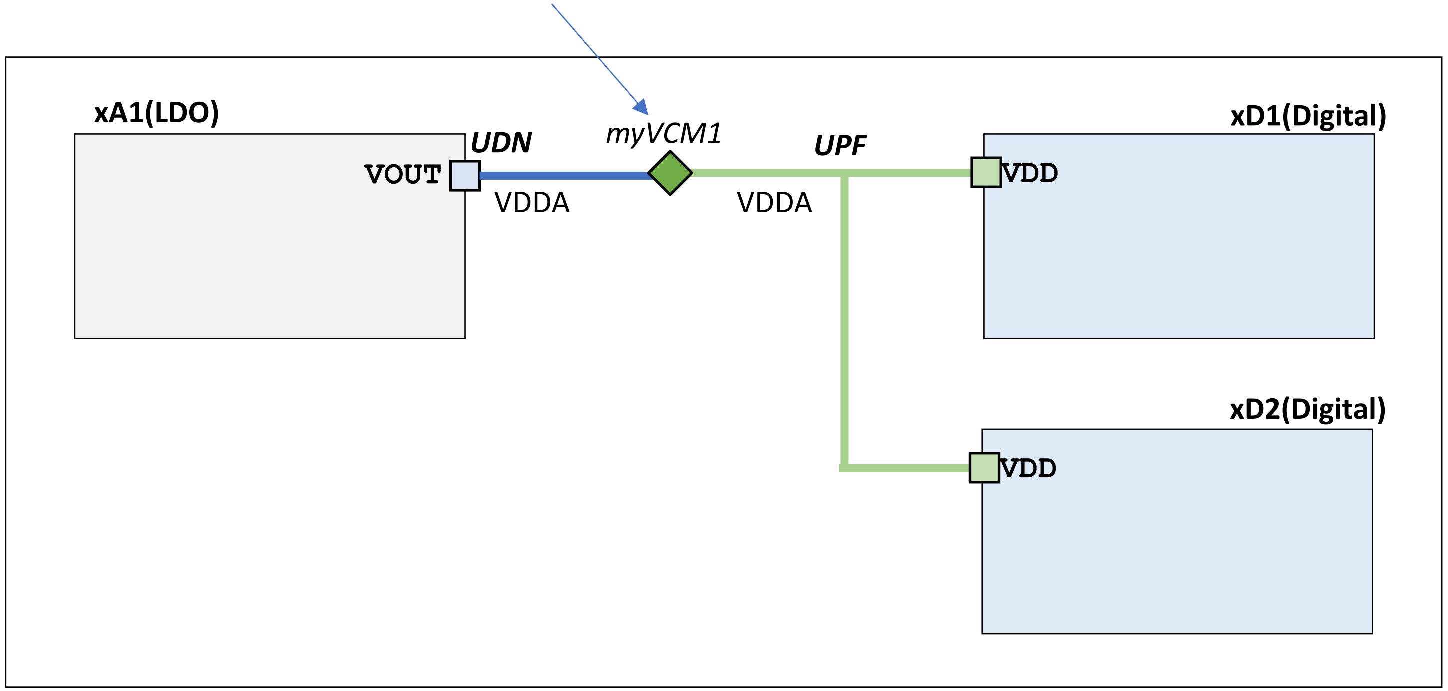
create_vcm *vcm_name*

And one of the following option sets:

- **-table** {{*from_value to_value*}*}
  - -hdl_type {<vhdl | sv> [*HDL_typename*]}
  - -conversion_direction <hdl2upf | upf2hdl>
  - -field *field_name*
- **-function** *hdl_package::function_name*
- **-model** *module_name*
  - -parameters {{*param_name param_value* }*}
- **-vcms** *ext_vcm_list*

Example Situation in which to use VCMs

Convert between a UDN and a UPF supply net

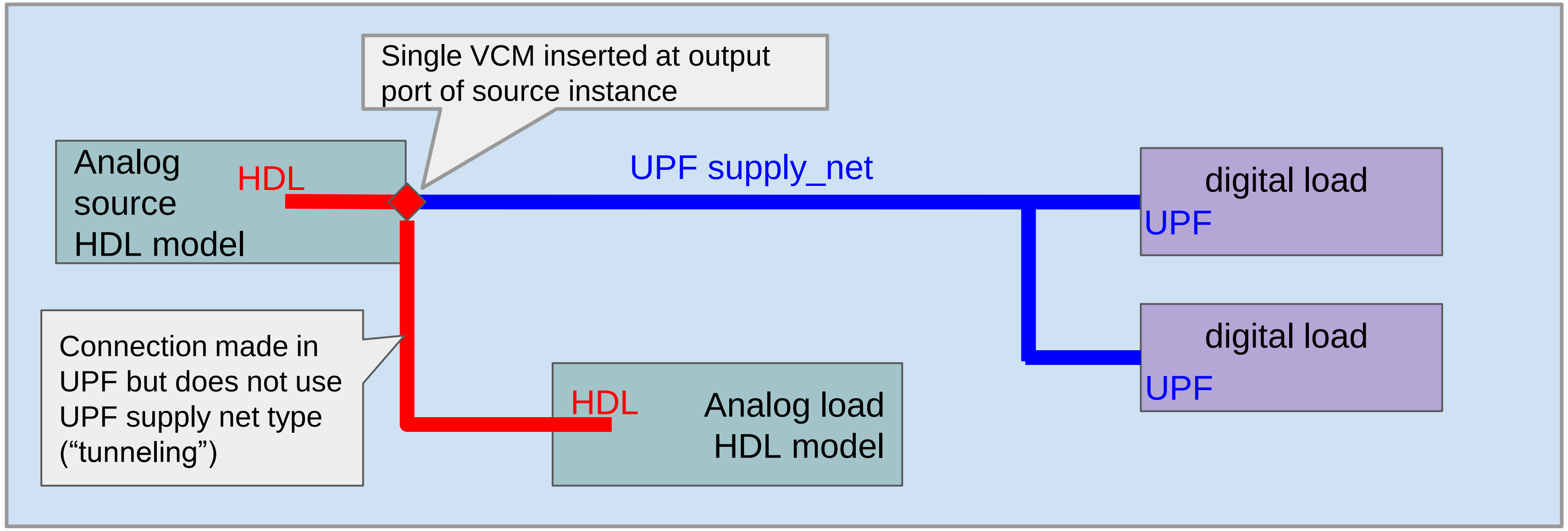


```
connect_supply_net VDDA -ports {xA1/VOUT xD1/VDD xD2/VDD} \
       -vcm myVCM1
```

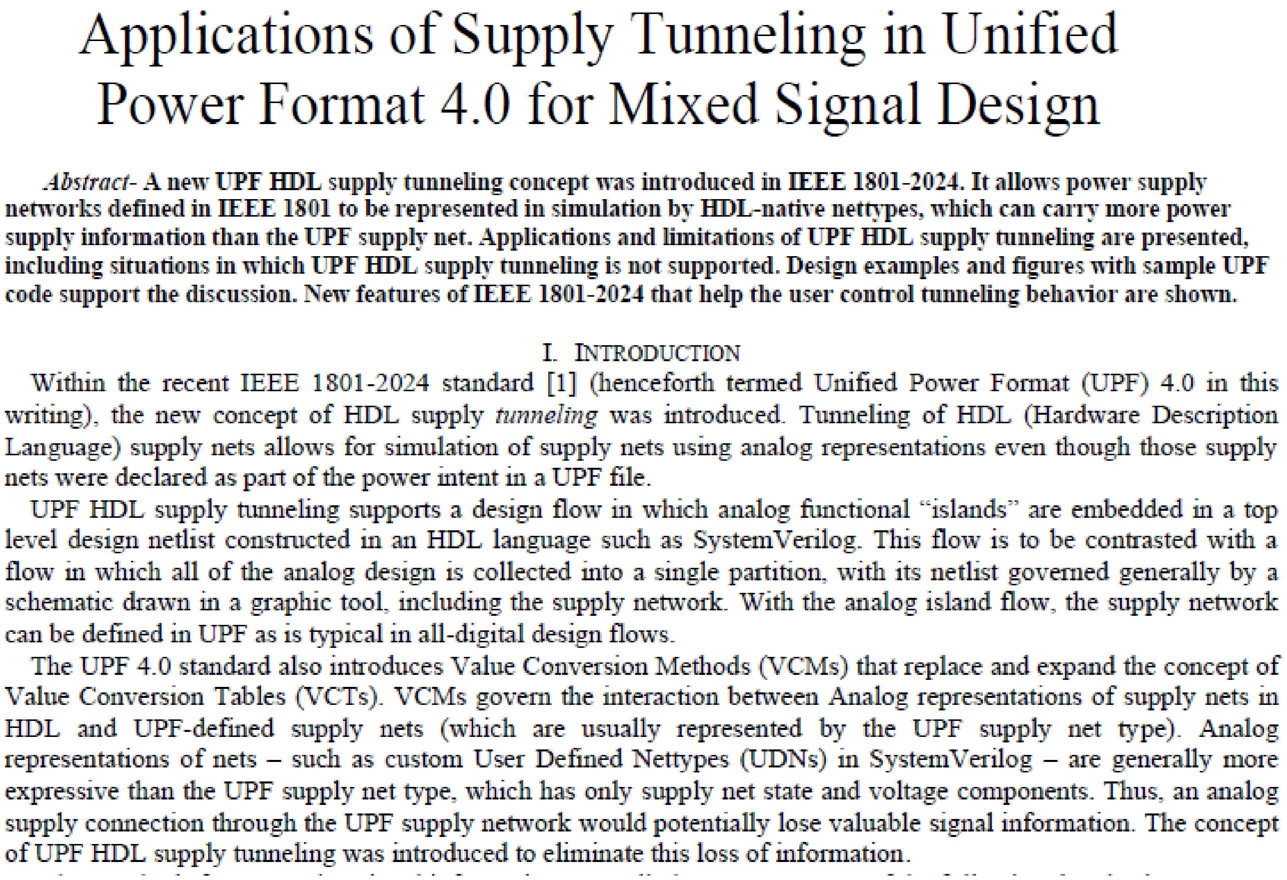
- In addition to single-bit logic, the VCM allows connecting UPF supply nets to:
  - integer
  - enum
  - real
  - User-Defined Nettype (UDN)
  - UDN with struct
  - record
- Only UPF supply nets (not logic nets or supply sets) can be connected with these methods.

Tunneling

- Tunneling allows a connection between same type of HDL net to preserve the full extent of that type
  - Normal UPF conversions result in loss of information because the UPF supply type only has an integer voltage and supply_state
  - The HDL type may contain a richer set of values that the load HDL model may take advantage of


Single VCM inserted at output port of source instance
Analog source HDL model
HDL
UPF supply_net
digital load
UPF
digital load
UPF
Connection made in UPF but does not use UPF supply net type ("tunneling")
HDL
Analog load HDL model

VCMs and HDL Tunneling Paper



Applications of Supply Tunneling in Unified Power Format 4.0 for Mixed Signal Design

Abstract- **A new UPF HDL supply tunneling concept was introduced in IEEE 1801-2024. It allows power supply networks defined in IEEE 1801 to be represented in simulation by HDL-native nettypes, which can carry more power supply information than the UPF supply net. Applications and limitations of UPF HDL supply tunneling are presented, including situations in which UPF HDL supply tunneling is not supported. Design examples and figures with sample UPF code support the discussion. New features of IEEE 1801-2024 that help the user control tunneling behavior are shown.**

I. Introduction

Within the recent IEEE 1801-2024 standard [1] (henceforth termed Unified Power Format (UPF) 4.0 in this writing), the new concept of HDL supply *tunneling* was introduced. Tunneling of HDL (Hardware Description Language) supply nets allows for simulation of supply nets using analog representations even though those supply nets were declared as part of the power intent in a UPF file.

UPF HDL supply tunneling supports a design flow in which analog functional "islands" are embedded in a top level design netlist constructed in an HDL language such as SystemVerilog. This flow is to be contrasted with a flow in which all of the analog design is collected into a single partition, with its netlist governed generally by a schematic drawn in a graphic tool, including the supply network. With the analog island flow, the supply network can be defined in UPF as is typical in all-digital design flows.

The UPF 4.0 standard also introduces Value Conversion Methods (VCMs) that replace and expand the concept of Value Conversion Tables (VCTs). VCMs govern the interaction between Analog representations of supply nets in HDL and UPF-defined supply nets (which are usually represented by the UPF supply net type). Analog representations of nets – such as custom User Defined Nettypes (UDNs) in SystemVerilog – are generally more expressive than the UPF supply net type, which has only supply net state and voltage components. Thus, an analog supply connection through the UPF supply network would potentially lose valuable signal information. The concept of UPF HDL supply tunneling was introduced to eliminate this loss of information.

Tuesday Session 1
Low Power UPF
Paper 1127

Monterey Carmel
9 am

continued

UPF Library

- The UPF library was introduced to provide a way to avoid name collisions between VCM definitions, which are otherwise global in scope.
  - Using a UPF library, a third-party design contribution (Intellectual Property or IP) can define VCMs for use in simulating the IP, without risking name collisions with VCMs defined for the SoC or other IPs.
- New commands which support UPF library use:

```
create_upf_library upf_library_name \
   -contents { \
      upf commands \
   }
```

```
use_upf_library upf_library_name
```

```
load_upf_library upf_library_file
```

Automatic Selection of VCM by net/data type

- Motivation:
  - UPF supply nets may connect to multiple HDL types
  - The –vcms option allows the specification of a list of VCM’s
  - Tools can choose the matching VCM based on the HDL type and do the proper conversion

```
create_vcm vcm_bundle \
   -vcms {sv_logic2upf sv_real2upf sv_udn2upf}
```

Commands and Option Changes

- New Commands and options
  - create_vcm
  - create_supply_net -tunneling
  - connect_supply_net -vcm
  - create_upf_library
  - use_upf_library
  - load_upf_library

- Legacy Commands and options
  - The commands **`create_hdl2upf_vct`** and **`create_upf2hdl_vct`** are legacy.
    - They can still be supported alongside VCMs by redirecting calls to these commands to **`create_vcm`**, supplying the necessary **`–conversion_direction`**
  - The **`–vct`** option in **`connect_supply_net`** is also legacy.
    - Tools may continue to support –**`vct`**

Table VCM Examples

- Consider a SystemVerilog package with a UDN defined as follows:

```
package ldo_net_pkg;
typedef struct {
           real     volts;
           real     current;
} ldo_struct_t ;
nettype ldo_struct_t ldo_supply_net;
endpackage
```

- A VCM that maps values on a HDL port of type ldo_supply_net can be declared as follows:

```
create_vcm LDONET2UPF \
-hdl_type {sv ldo_net_pkg::ldo_supply_net} \
-conversion_direction hdl2upf \
-field volts \
-table { \
            {{5.0  * } {OFF            }} \
            {{1.0 5.0} {FULL_ON    1.1 }} \
            {{0.6 1.0} {PARTIAL_ON 0.9 }} \
            {{ *  0.6} {OFF            }} \
       }
```

Function VCM Example

```
package myVCM_pkg;
   import UPF::*;
   import ldo_net_pkg::*;

   function automatic upfSupplyTypeT func_h2u_snap (ldo_struct_t hdl_in);
      upfSupplyTypeT     upf_out;

      upf_out.voltage = 0;
      upf_out.state       = UNDETERMINED;

      if (hdl_in.volts <= 0.2) begin
               upf_out.voltage   = 0;
               upf_out.state     = OFF;
      end
      else if ((hdl_in.volts > 0.2)
            && (hdl_in.volts <= 0.9)) begin
               upf_out.voltage   = 0.9;
               upf_out.state     = FULL_ON;
      end
           . . .
      return  upf_out;
   endfunction
endpackage
```

```
# Using the package in UPF
create_vcm LDONET2UPF_function \
  -function myVCM_pkg::func_h2u_snap
```

Model VCM Example

```
import UPF::*; import ldo_net_pkg::*;
module snap_volt_vcm #(
          //default parameter defs
          parameter ov_threshold                  = 5.0,
          parameter hi_snap_volts                 = 1.1,
          parameter hi_threshold                  = 1.0,
          parameter on_snap_volts                 = 0.9,
          parameter on_threshold                  = 0.2
 )   (
          input             ldo_supply_net               hdl_in,
          output            upfSupplyTypeT               upf_out
);

always @(hdl_in.volts)
   begin
      . . .
   end
endmodule
```

```
create_vcm LDONET2UPF_module \
  -model  my_module_lib.snap_volt_vcm \
  -parameters { \
      {ov_threshold    5.0} \
      {hi_snap_volts   1.5} \
      {hi_threshold    1.4} \
      {on_snap_volts   1.0} \
      {on_threshold    0.5} }
```

Advantages of Functions over Modules

- **Functions are less resource intensive.**
  - They exist for one event and then disappear
  - Modules are instantiated in the netlist, and exist for the entire simulation, occupying memory even when nothing happens to them

- **Functions can be imported from other languages (e.g. C++)**
  - Modules must be HDL

Advantages of Modules over Functions

- Modules can be easily parameterized
  - One module description can be used as the basis for many VCMs (using create_vcm –parameters)
- Modules can model time delay effects
  - Since they are static objects, they can respond to stimuli over several event times
  - More complex behaviors can be modeled

List VCM Example

```
create_vcm vcm_bundle \
    -vcms {sv_logic2upf sv_real2upf sv_udn2upf}
```

- You can include VCMs for both directions in a list, as long as the ports you connect with it are **input** or **output** (but not **inout**).
- You can only include one VCM with a given HDL type and direction in the list.
- All of the VCMs in the list have to be previously defined with a **create_vcm** command.
- The list can be a mix of table VCMs, function VCMs, module VCMs, or even other list VCMs.
- Use the name of the list VCM when you make UPF supply net connections to HDL ports.

Making Connections with VCMs

- Apply your VCMs to make connections between HDL ports and UPF supply nets by using the **`connect_supply_net`** command with a new option: **`-vcm`**

- For example:

```
connect_supply_net vdd -ports {u1/vdd_1v0} -vcm vcm_bundle
```

- Some things to be aware of:
  - Expect an error if the VCM you specify does not match the HDL type and port direction you are connecting to. If you specify a list, then exactly one of the VCMs in the list has to match.
  - If any ports listed in **`-ports {}`** are UPF supply ports (and therefore do not need conversion), **`-vcm`** will be ignored for those ports.
  - You can also use **`connect_supply_net`** with **`–pg_type`** to connect to many ports with the same pg_type.

2025

DESIGN AND VERIFICATION™

DVCON

CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA

FEBRUARY 24-27, 2025

Refinable Macros and Terminal Boundaries in UPF 4.0: Empowering Soft IPs of the Future

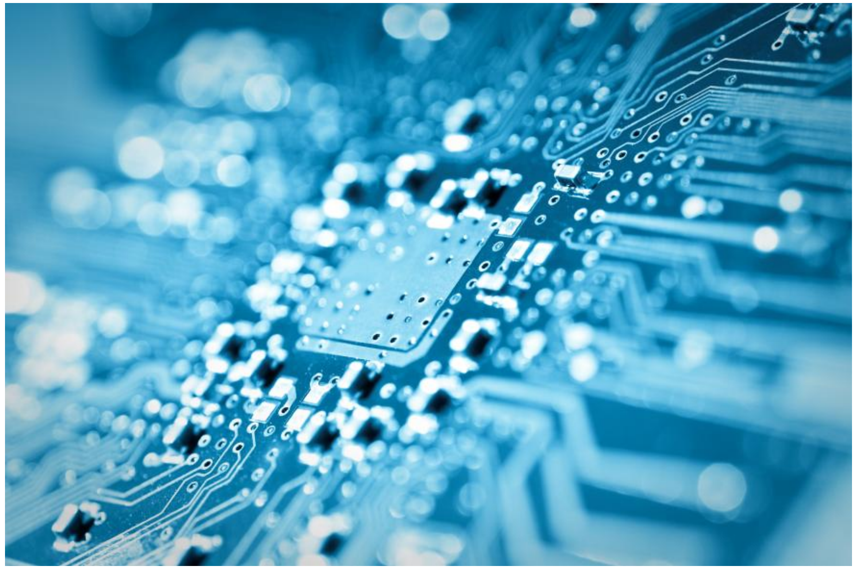
accellera

SYSTEMS INITIATIVE

Refinable Macros in UPF 4.0

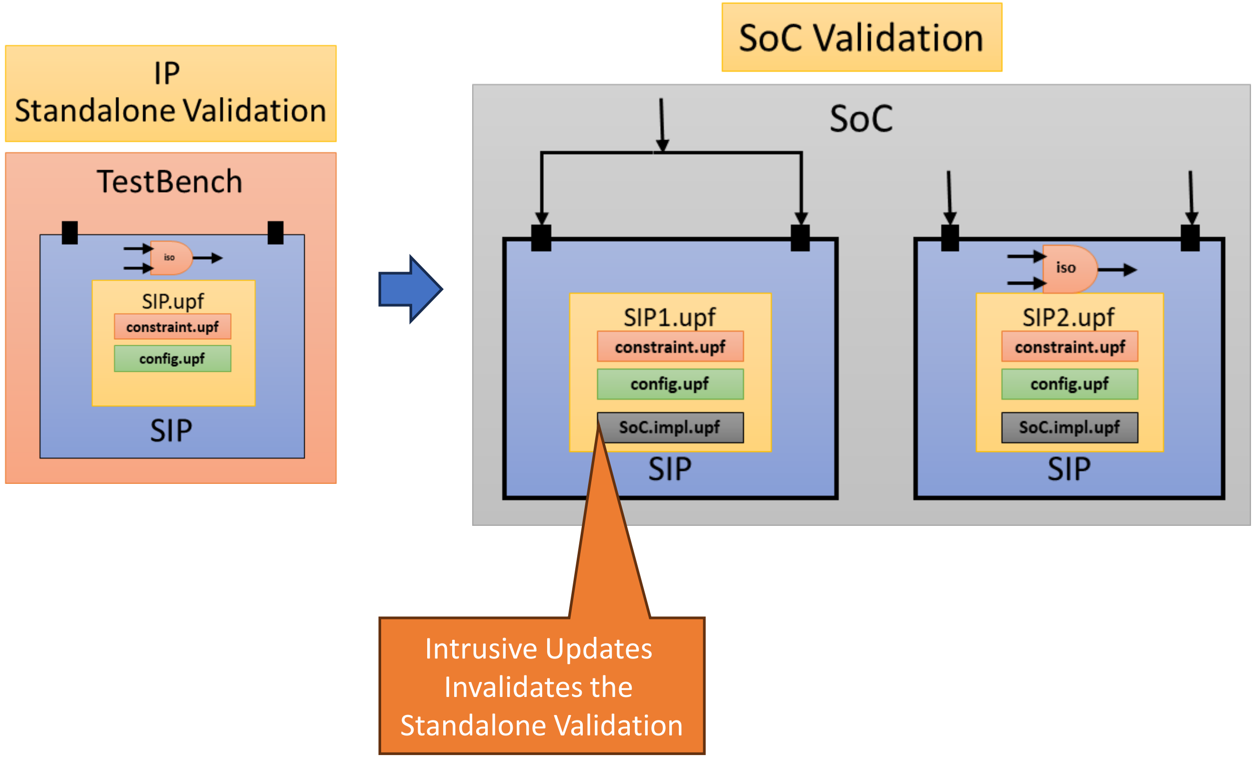
Empowering Soft IPs(SIP) of the Future

- ***Enabling Non-Intrusive Refinements in Bottom-up Verification Flows***
- **Presenter**
  - Amit Srivastava
- **Agenda**
  - Why we need Refinable Macros
  - How they differ from Soft Macros
  - Marking IPs as Refinable Macro
  - Refinable Macro in Action



Motivation: Challenges with SIPs in Bottom-Up Verification

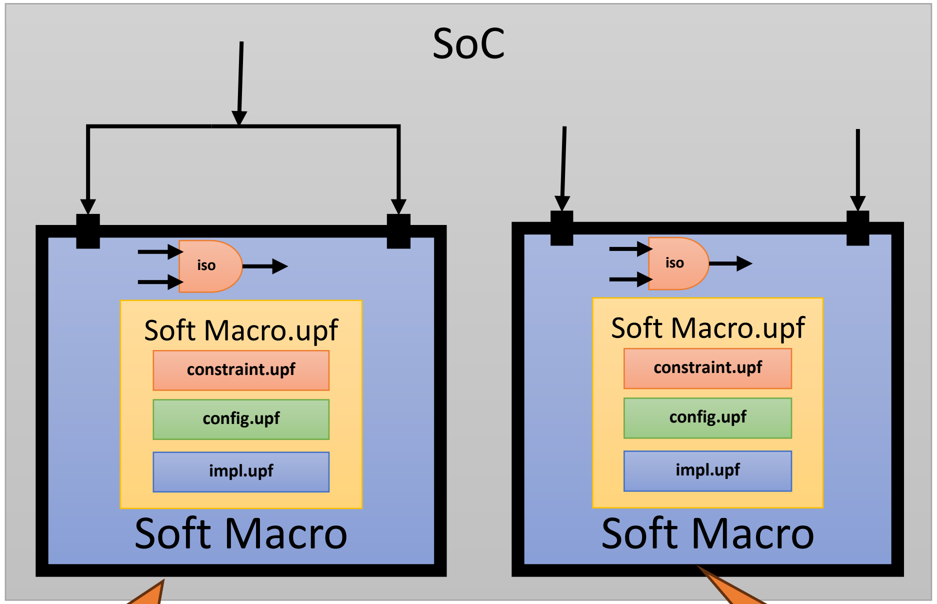
- Standalone UPF **Verification**
- Higher-Level Implementation Requires Power Intent **Updates**
- **Intrusive** Methods Risk Invalidating Power Intent
- Need a methodology
  - **Safe Changes**
  - No Re-verification



Intrusive Updates Invalidates the Standalone Validation

Why Soft Macros Fall Short for Bottom-Up Verification

- **Rigid Implementation Boundaries**
  - Suitable for Bottom-Up Implementation Flows
- No Room for **Non-Intrusive Refinements**
- Forces **Intrusive Edits** and Re-Validation
- Lacks **System-Level Optimization**

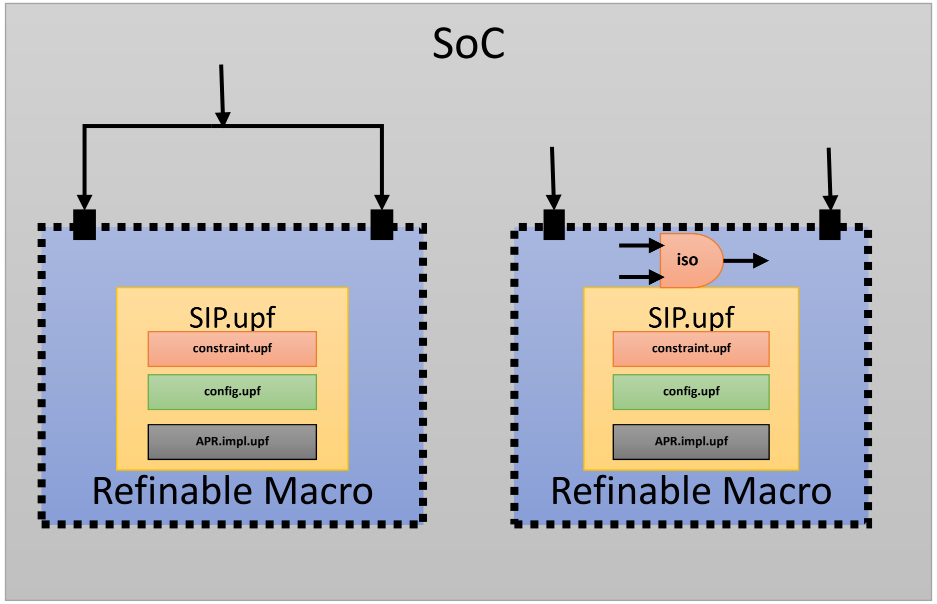


STRICT
TERMINAL BOUNDARIES

Pre-Hardened outside SoC
No logic optimizations

Refinable Macros in UPF 4.0

- Ideal for Bottom-Up Verification
- Refinable Terminal Boundaries
- Non-Intrusive Power Intent Updates
- Verification Integrity Preserved
- Enables System-Level Optimization



Marking IPs as Refinable Macros

- Simple UPF Attribute
  - IP or External Marking
- Override to Soft Macro if Needed
- Non-Intrusive Terminal Boundary
- Retains IP Verification

Mark IP as Refinable Macro

Safe Updates

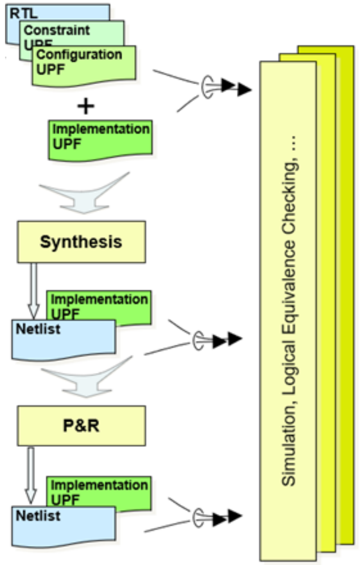
Preserves Verification

```
# Mark directly in IP UPF:
set_design_attributes -models . -is_refinable_macro true

# Or mark externally:
set_design_attributes -models IP_Design -attribute {UPF_is_refinable_macro TRUE}
```

Implementation UPF
& `load_upf -implementation`

- **Safe Refinements** via Allowed Commands and options
    - UPF 4.0 defines allowed commands
- -implementation Enforces **Correct-by-Construct** UPF
    - EDA Tools Enforce Compliance
- **No Alteration** of Original IP UPF
- **Preserves Verification** Integrity

```
# SoC UPF
load_upf ip_impl.upf \
   -scope myIP \
   -implementation

# ip_impl.upf
set_isolation PGD_to_AON \
  -domain PGD \
  -location parent
  -update
```

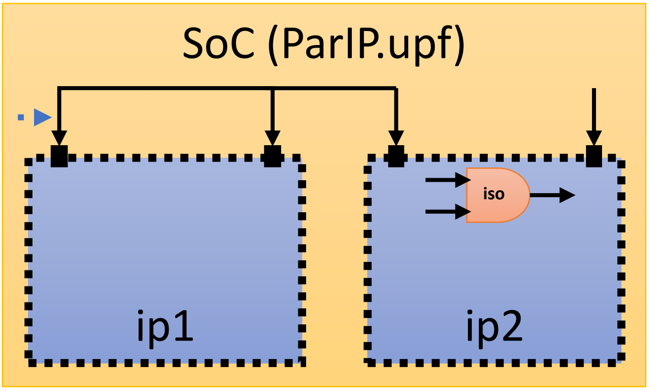


Practical Example: Refinable Macros in Action

ip.upf

```
set_design_attributes -models . \
-is_refinable_macro TRUE

create_supply_set ss_IP_AON
create_supply_set ss_IP_PGD

create_power_domain AON -elements {.}
create_power_domain PGD -elements {ip1_pgd_wrapper}

## Isolates all outputs where different
## supplies power source and sink
set_isolation PGD_to_AON -domain PGD \
 -isolation_supply_set ss_IP_AON \
 -applies_to outputs -source ss_IP_PGD \
 -diff_supply_only TRUE \
 -isolation_signal pwr_manager/iso_en_b \
 -isolation_sense low
```

AON and PGD supplies are shorted for one instance of the IP



ParIP.upf

```
create_power_domain par_AON -elements {.}
create_supply_set ss_SOC_AON
create_supply_set ss_SOC_PGD

load_upf ip.upf -scope ip1
load_upf ip_impl.upf -scope ip1 -implementation
associate_supply_set {ss_SOC_AON ip1/ss_IP_AON}
associate_supply_set {ss_SOC_AON ip1/ss_IP_PGD}

load_upf ip.upf -scope ip2
load_upf ip_impl.upf -scope ip2 -implementation
associate_supply_set {ss_SOC_AON ip2/ss_IP_AON}
associate_supply_set {ss_SOC_PGD ip2/ss_IP_PGD}
```

ip_impl.upf

```
set_isolation PGD_to_AON \
 -domain PGD \
 -location parent
 -update
```

Implementation updates only

Conclusion & Key Takeaways

- UPF 4.0 Bridges SIP Verification Gaps
- Refinable Macros Enable Non-Intrusive Updates
- Implementation UPF Ensures Correct-by-Construct
- Preserves Verification Integrity, Saves Time
- Join Our Paper Session

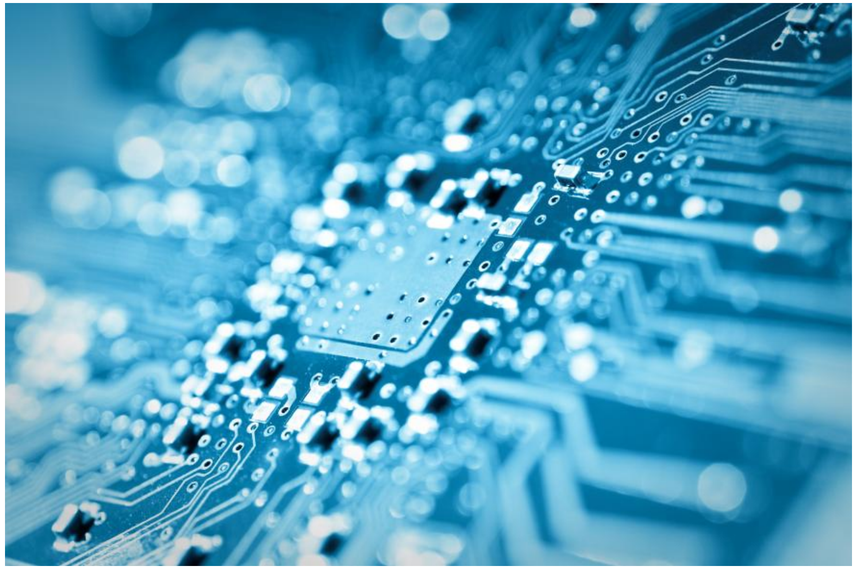
2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

Retention Modeling in UPF 4.0

accellera
SYSTEMS INITIATIVE

Future Proofing Retention in UPF 4.0

- Future Proofing Power Intent Specification through UPF 4.0 for Evolving Advanced State Retention Strategies
- **Primary Author**
  - Lakshmanan Balasubramanian
- **Agenda**
  - Why retention semantics were updated
  - What is new in UPF 4.0
  - Examples enabled by new changes

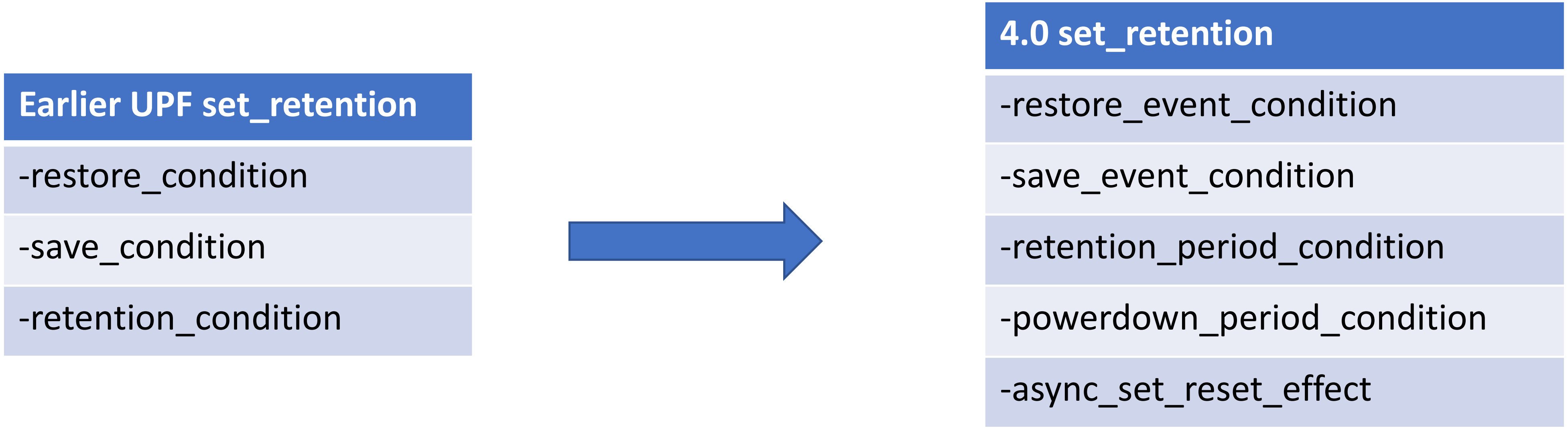


Motivation

- Advances in state retention cell design have exposed limitations in earlier versions of the UPF LRM
- Enhancements needed to model the more complex clock, setup, retention relationships provided by these new technologies
- Improved modeling will catch issues early in the design cycle
- 4.0 improvements were designed to be forward looking and provide a flexible platform that can adjust to future requirements

Retention Overview of changes

- Existing set_retention conditions were expanded and redefined to be more accurate
- Ability to specify how set/reset will affect the behaviors



Earlier UPF set_retention
-restore_condition
-save_condition
-retention_condition

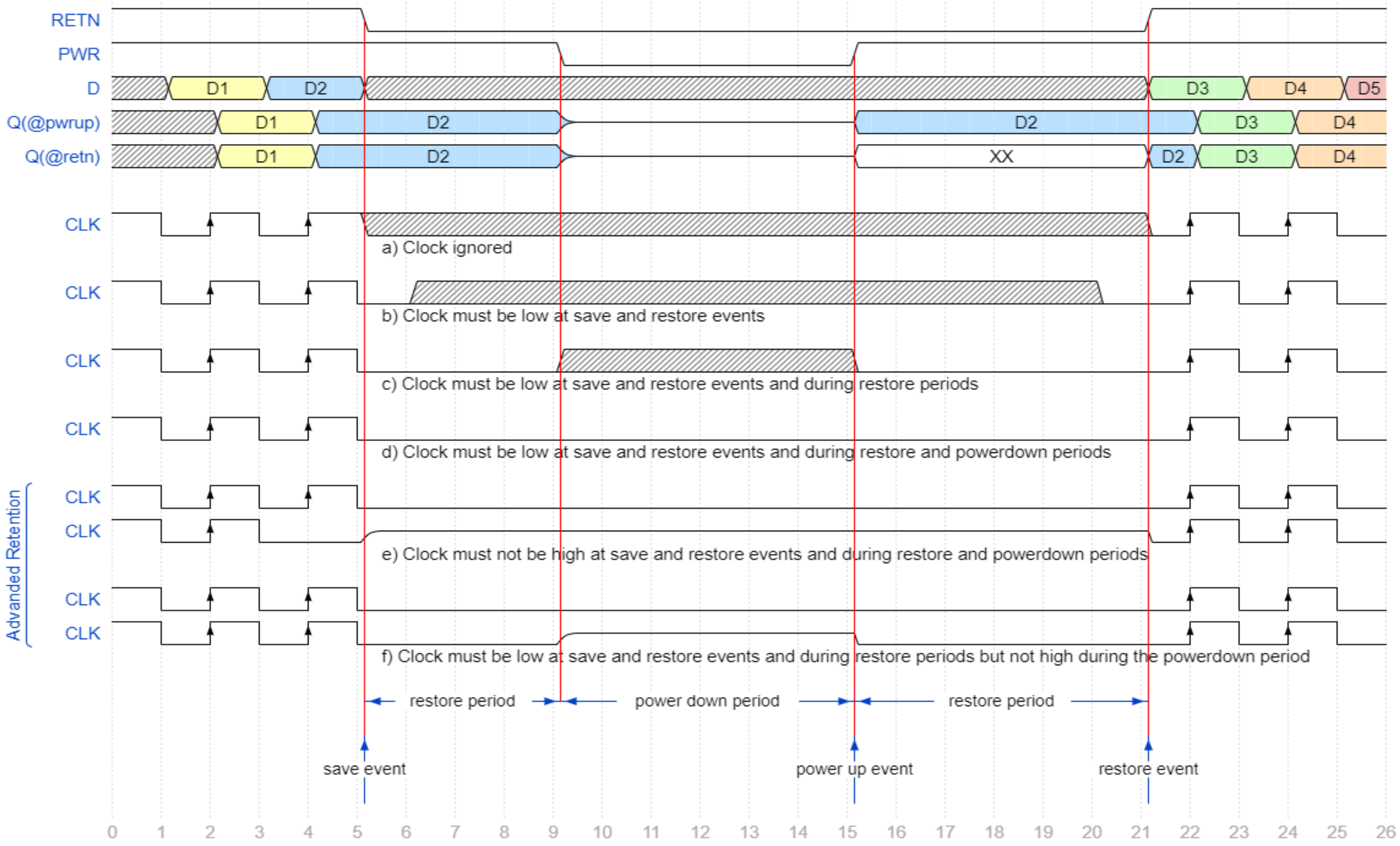
4.0 set_retention
-restore_event_condition
-save_event_condition
-retention_period_condition
-powerdown_period_condition
-async_set_reset_effect

4.0 new retention options

4.0 set_retention	
-restore_event_condition	gates the restore event from triggering the restore operation of the register. The register is restored when the restore event occurs and the **-restore_event_condition** is *True*.
-save_event_condition	gates the save event, defining the save behavior of the register. The register contents are saved when the save event occurs and the -**save_event_condition** is True.
-retention_period_condition	**-restore_period_condition** also gates the restore operation. The restore operation occurs continuously during the entire restore period. The **restore_period_condition** shall be TRUE throughout the entire restore period;
-powerdown_period_condition	defines the conditions under which the retained value is maintained when the domain power is OFF. If the **-powerdown_period_condition** is specified, it shall evaluate to TRUE the entire time that the domain's primary power is OFF for the value of the state element to be retained.
-async_set_reset_effect	specifies how the set/reset signals affect the value of the retained element and the output during the restore period (ignored, reset retained value and output, reset only the output value)

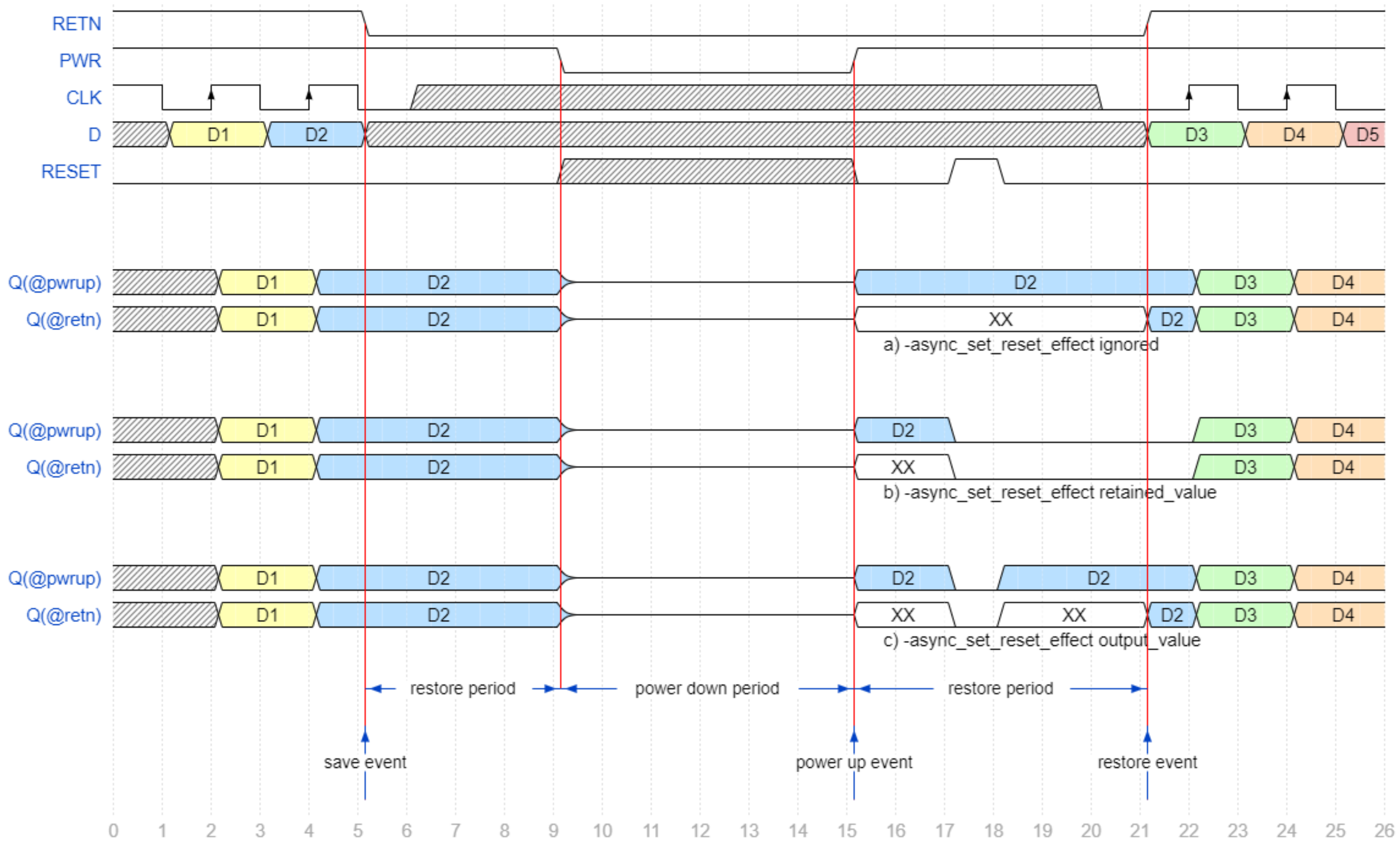
Retention Waveform Example

- 4.0 clearly defines the periods of the full retention cycle
- Each period can have an independent set of requirements relative to clock, reset, and other design signals
- Overcomes limits and ambiguities in previous versions



RETN
PWR
D
D1
D2
D3
D4
D5
Q(@pwrup)
Q(@retn)
XX
CLK
a) Clock ignored
b) Clock must be low at save and restore events
c) Clock must be low at save and restore events and during restore periods
d) Clock must be low at save and restore events and during restore and powerdown periods
Advanded Retention
e) Clock must not be high at save and restore events and during restore and powerdown periods
f) Clock must be low at save and restore events and during restore periods but not high during the powerdown period
restore period
power down period
restore period
save event
power up event
restore event
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

Async_set_reset_effect



`ignored`: does not change retained value or output value

`retained_value`: changes both the retained value and output

`output_value`: only changes the output, the saved value propagates once the reset is deasserted

Additional Changes for retention

- Improved definition of **UPF_GENERIC CLOCK**
- Allow **UPF_GENERIC_CLOCK** and **UPF_GENERIC_ASYNC_SET_RESET** to be used in all conditions
- 9.7 simulation of retention section overhauled
  - Greatly enhanced examples with detailed waveforms for most common retention types

Agenda

- Introduction
- Interconnect between UPF supplies and arbitrary HDL types
- Improvements in successive refinement and refinable macros
- Overview of Retention Changes
- Virtual Supply and Virtual Equivalence
- General Updates
- Beyond 4.0
- Q/A

Virtual Supplies & Equivalence

- Motivation
  - Pre-4.0, no way to model a supply net that did not physically exist in the design
    - No way to create driver/receiver supply to model external supplies
    - Required use of power models to create internal supplies for macros and use them in power states and strategy filters
  - Many tools already have ad-hoc methods to address these
- Solution
  - Define virtual supply nets, ports and supply_sets
    - Supply nets/ports/sets that are virtual have no physical implementation
    - Can be used in add power state, connect_supply_net, source/sink filtering, etc
    - Have the same simulation semantics as non-virtual supplies
    - New concept of virtually equivalent – general concept is the same as electrically equivalent but without interchangeability

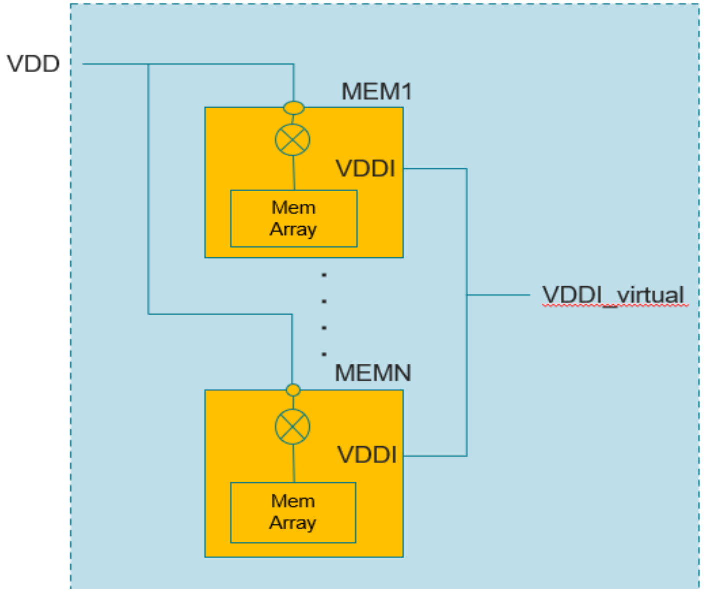
Virtual Supplies, Supply Sets and Ports

- Virtual supplies allow
  - designers to describe supplies that are not physically connected to the block
  - virtual connections between internal supplies of macros to setup equivalence
  - specification of driver and receiver supply of ports and macro pins to enable –source/-sink filtering
  - power states to be easily defined for cases where there is no physical supply net
- Virtual supply restrictions
  - They cannot be used to power any active logic in the design:
  - Cannot be a primary supply of a domain, can not be used as the supply for any strategy
  - They cannot be written out in the physical design outputs
  - Connection of supply subnets does not create interchangeability
- Syntax changes

```
create_supply_net –virtual
create_supply_port –virtual
create_supply_set -virtual
```

Case1: Virtual Supply used to model functionally equivalent supplies

- Motivation: Simplify the power state and LS/ISO strategies for macros
- Methodology
  - The internal supply (VDDI) pins are connected with a virtual supply net making them virtually equivalent
  - Allows creation of virtual supply sets that can be used for power states and –source/-sink iso/LS rules
  - Implementation tools are forbidden from using a virtual net to power logic
  - Implementation tools will not write these virtual connections into the output Verilog



Virtual Supply UPF code

```
# Create the virtual supply net and virtual supply set
create_supply_net VDDI_virtual -virtual -resolve parallel
create_supply_set SS1_virtual -function {power VDDI_virtual} -function {ground VSS} -virtual
```

An internal pin, virtual connection
- no physical connection

```
#connect the virtual supply net to each memory’s internal supply pin VVDDI
connect_supply_net VDDI_virtual -ports { MEM1/VDDI …. MEMN/VDDI}
```

Virtual supplies can have states specified in same way as physical

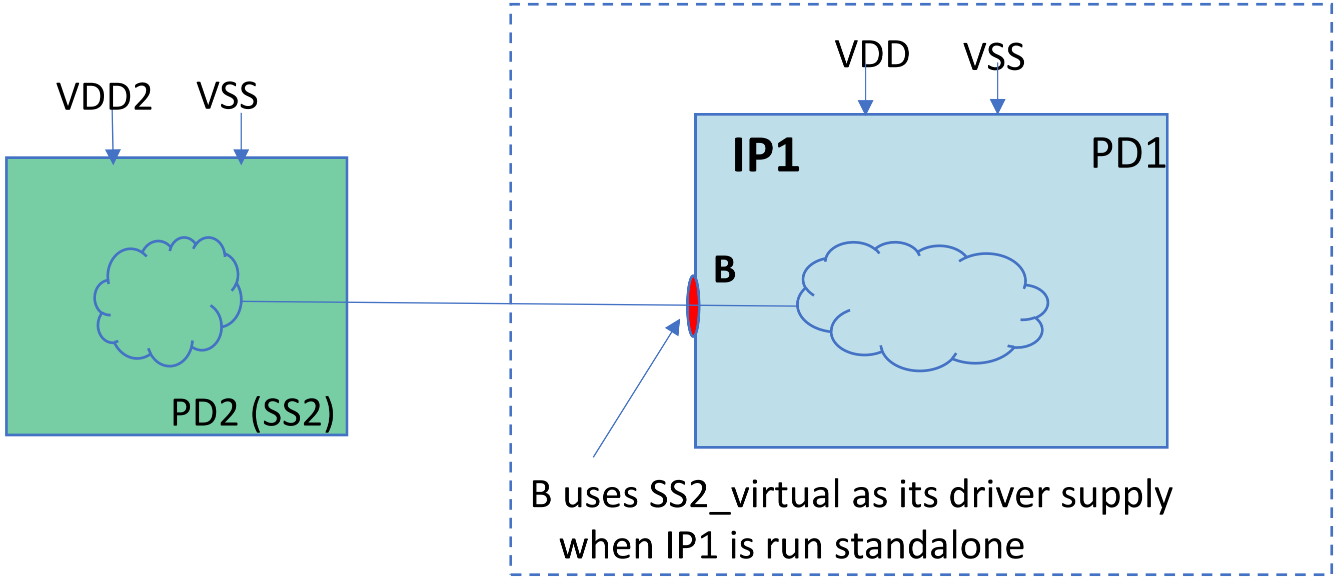
```
# A single supply level power state instead of one per MEM block
add_power_state SS1_virtual -supply  \
-state {OFF -supply_expr {power == OFF}} \
-state {ON -supply_expr { power == {FULL_ON} && ground == FULL_ON}}
```

```
# The logic expression has a single term, instead of one term per MEM block
add_power_state PD1 -domain -state {ON -logic_expr {SS1_virtual == ON}}
```

Virtual supplies can be used as source/sink for ISO/LS strategies

```
# A single set_isolation covers any output driven by any of the connected MEM blocks
set_isolation ISO1 -domain PD1 -source SS1_virtual -applies_to outputs
```

Case2 : Virtual supply to model external supplies



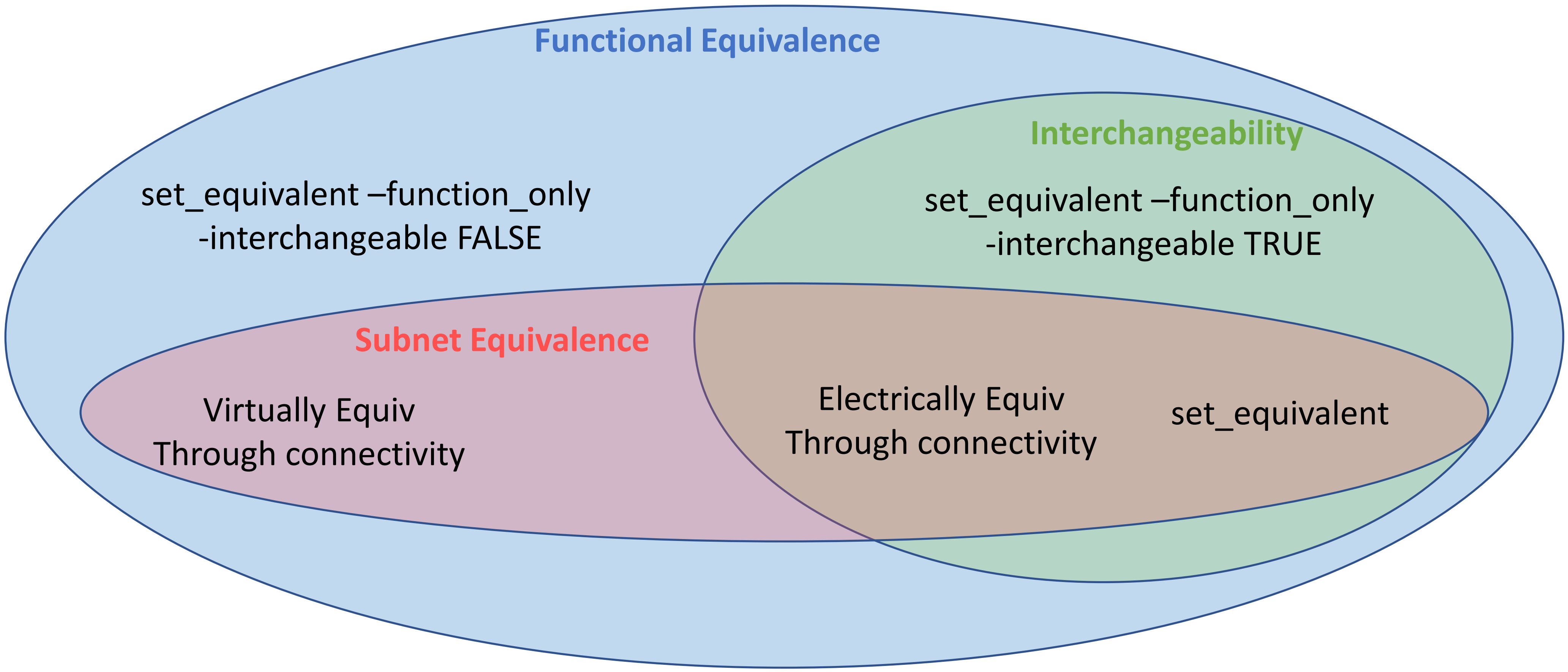
B uses SS2_virtual as its driver supply when IP1 is run standalone

```
create_supply_net VDD2_virtual -virtual
create_supply_set SS2_virtual -virtual
      -function {power VDD2_virtual}
     -function {ground VSS}


set_port_attribute -ports B
       -driver_supply SS2_virtual


set_isolation iso1 -domain PD1
        -source SS2_virtual
        -applies_to inputs
```

Virtual Equivalence

- Prior to 4.0, any supply nets connected to each other were electrically equivalent
  - Subnet equivalent – any drivers on any of the connected supply nets had to be treated as one net and had to be resolved in simulation
  - The nets were interchangeable: The nets could be used interchangeably including in physical design.
- Virtual Nets are not real connections, they are not interchangeable
- Virtual Equivalence
  - Keeps the subnet equivalence for simulation, but does not include interchangeability
  - Affects transitive properties
    - A(real) connects to B(real), and B(real) connects to C(real); Then A and C are electrically equiv
    - A(real) connects to B(virtual) and B(virtual) connects to C(real); then A and C are only virtually equiv and are non-interchangeable

Understanding Equivalence



Agenda

- Introduction
- Interconnect between UPF supplies and arbitrary HDL types
- Improvements in successive refinement and refinable macros
- Overview of Retention Changes
- Virtual Supply and Virtual Equivalence
- General Updates
- Beyond 4.0
- Q/A

Details on select topics

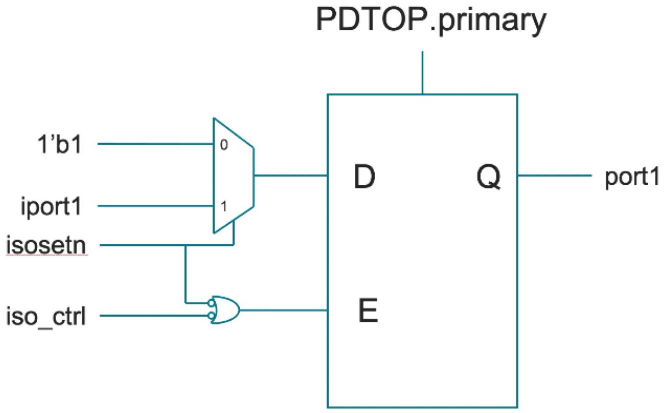
- Support for set/reset on Latch Isolation
- `map_retention_clamp_cell`
- `find_objects -expand_to_bits`
- Precedence Updates
- `set_port_attributes -feedthrough`
- Naming Updates

Support for set/reset on Latch Isolation

- Support for set/reset on Latch based isolation

```
set_isolation
    -async_set_reset {net_name <high|low>} #specify cntrl signal
    -async_clamp_value  <0|1>              #specify set or reset
```

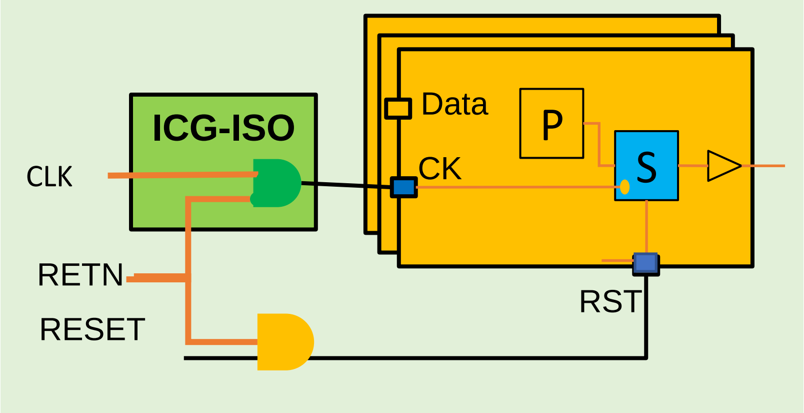
- Create a new port attribute to specify async_clamp_value
  - **UPF_async_clamp_value**
  - **set_port_attributes -async_clamp_value** <0|1>



```
set_isolation ISO -domain
-isolation_signal iso_ctrl
-isolation_sense high
-clamp_value latch
-async_set_reset { isosetn low}
-async_clamp_value 1
. . .
```

map_retention_clamp_cell

- For "zero-pin" state retention a clamp cell is automatically inserted on the clock/reset pins
- The **map_retention_clamp_cell** allows the specification of what cell to use to implement that clamp.



- Example:

```
set_retention RET1 -domain PD1 ...
    -save_signal {RETN high}  -restore_signal {RETN low}
map_retention_cell RET1 -domain PD1 -lib_cells {SCL9T_ZPR_X2}
map_retention_clamp_cell {RET1} -domain PD1
    -clock_clamp_lib_cells {SCL9T_ISOT1_X1}   # green isolation in diagram
    -async_clamp_lib_cells {SCL9T_ISOT2_X1}   # orange isolation in diagram
```

find_objects –expand_to_bits

- **find_objects** can return a single object for a bus or a list of individual bits
  - This was possible before by using patterns like “xyz\[*\]” to return a list of individual bits
- In 4.0, this process has been made easier by adding an “**-expand_to_bits**” option
  - When set true, the individual bits will always be returned
  - When false (or not set), the pre-4.0 behavior will apply
- Whether the individual bits are returned or the full bus is returned can affect the precedence of this list in other commands
  - Example : set_isolation ISO1 –elements [find_objects ….]
  - In the elements list, bits will have higher precedence than the full bus

Find_objects . –object_type port	Return Value
`-pattern {pmda\[*\]}`	`{pmda[1] pmda[0]}`
`-pattern {pmda\[*\]} -expand_to_bits`	`{pmda[1][0] pmda[1][1] pmda[0][1] pmda[0][0]}`

Precedence Updates

- **`set_retention`**
  - **`set_retention`** with **`–no_retention`** now has precedence over **`set_retention`** without **`–no_retention`**
- **`create_power_domain`**
  - Command that applies to all design elements of instances included transitively. If two commands specify different ancestors of a design element, the command with the lower ancestor applies.
- **`set_port_attributes –driver_supply/-receiver_supply`**
  - If after applying the precedence rules above, the predefined port attributes **UPF_receiver_supply** or **UPF_driver_supply** are defined on a given port using both hierarchical and non-hierarchical names then the hierarchical name shall take precedence.
- Composite types
  - When determining precedence, composite data types are treated as a multibit signal. A record field or array index of a composite data type referred to explicitly by name is also treated as a part of the multibit signal.
- **`set_design_attributes -is_hard_macro|-is_soft_macro|-is_refinable_macro`**
  - If the macro has multiple -is_*_macro attributes set, then **`-is_hard_macro`** has highest precedence, followed by **`-is_soft_macro`**

set_port_attributes -feedthrough

- In 3.1, the semantics around multiple feedthrough groups was unclear.
- In 4.0:
  - **set_port_attributes -ports** {*port_list*} **-feedthrough** *feedthrough_name*
  - All ports connected with the UPF_feedthrough attribute set to the same feedthrough name, are defined as connected
- Example:
  - The following code defines two separate feedthrough groups: X that includes a, b and c, and Y that only contains e and f.

```
set_port_attributes -ports {a b} -feedthrough X
set_port_attributes -ports {c} -feedthrough X
set_port_attributes -ports {e f} -feedthrough Y
```

Naming Updates

- Clarify what character should be used in UPF to specify the generate block delimiter
  - In the design flow generate blocks are unique, for simulation they create a hierarchy but for implementation they don’t.   The naming style also can differ based on tools settings.
  - In 4.0, the LRM was updated to define a single style that should be used in the UPF
  - **generate block delimiter character**: A special character used in composing names containing generate block labels. The generate block delimiter character is a dot (.).
- New Library naming
  - When referencing a model in a command argument, its name may be prefixed by its library name followed by a dot ("."). This limits the effect of a command to the particular version of that model compiled into the specified library. A model name specified with the "<library>.<model>" syntax is considered a simple name (5.3.3.2)

Beyond 1801-2024

- 1801-2024 major features were the response to new technologies and design methodologies
- Beyond 1801-2024
  - Continued innovation on mixed signal design and interfacing
    - Enable supply networks to carry information about power generation and consumption
    - Bi-directional supply ports
    - Features to improve static checking of designs with mixed analog/digital components
    - Power modeling in the context of mixed signal integrations
  - Information model improvements beyond SV and VHDL
  - Improvements to support new technology cells
  - Ease of use/Ease of specification improvements

Questions