DVCon 2012
Design & Verification Conference & Exhibition

February 28 – March 1, 2012

Low Power SoC Verification: IP Reuse and Hierarchical Composition using UPF

Sorin Dobre
QUALCOMM CDMA Technologies

Amit Srivastava
Rudra Mukherjee
Erich Marschner
Chuck Seeley
Mentor Graphics

Agenda

- Introduction
- Power Aware Design and Verification
- IEEE 1801 Unified Power Format
- Need for UPF Methodology
- UPF 2.0 Methodology
  - Overview
  - Concepts
  - Features
  - Steps
  - Highlights
- Summary

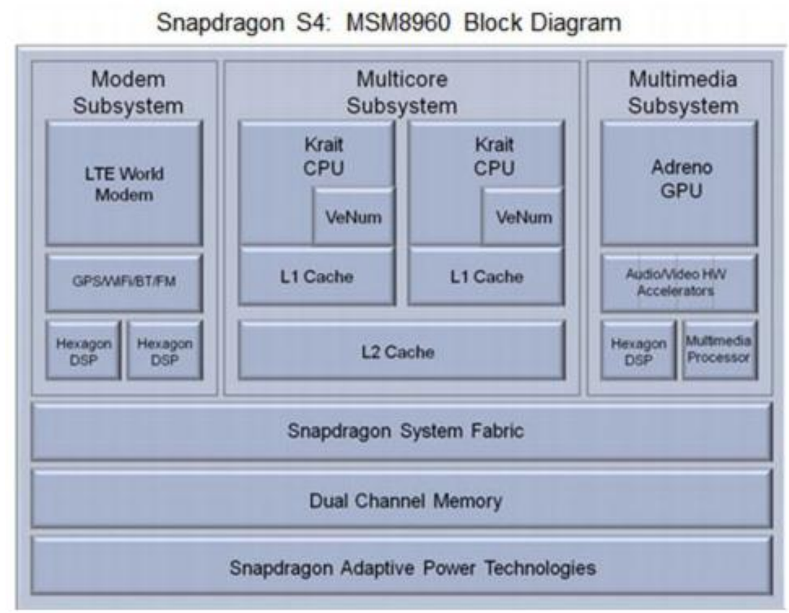
Introduction

Today's SoCs are

- Incredibly Complex
- Multiple Sub-Systems
- Providing Many Functions
- Increasingly Integrated
- Battery-Powered

They Must

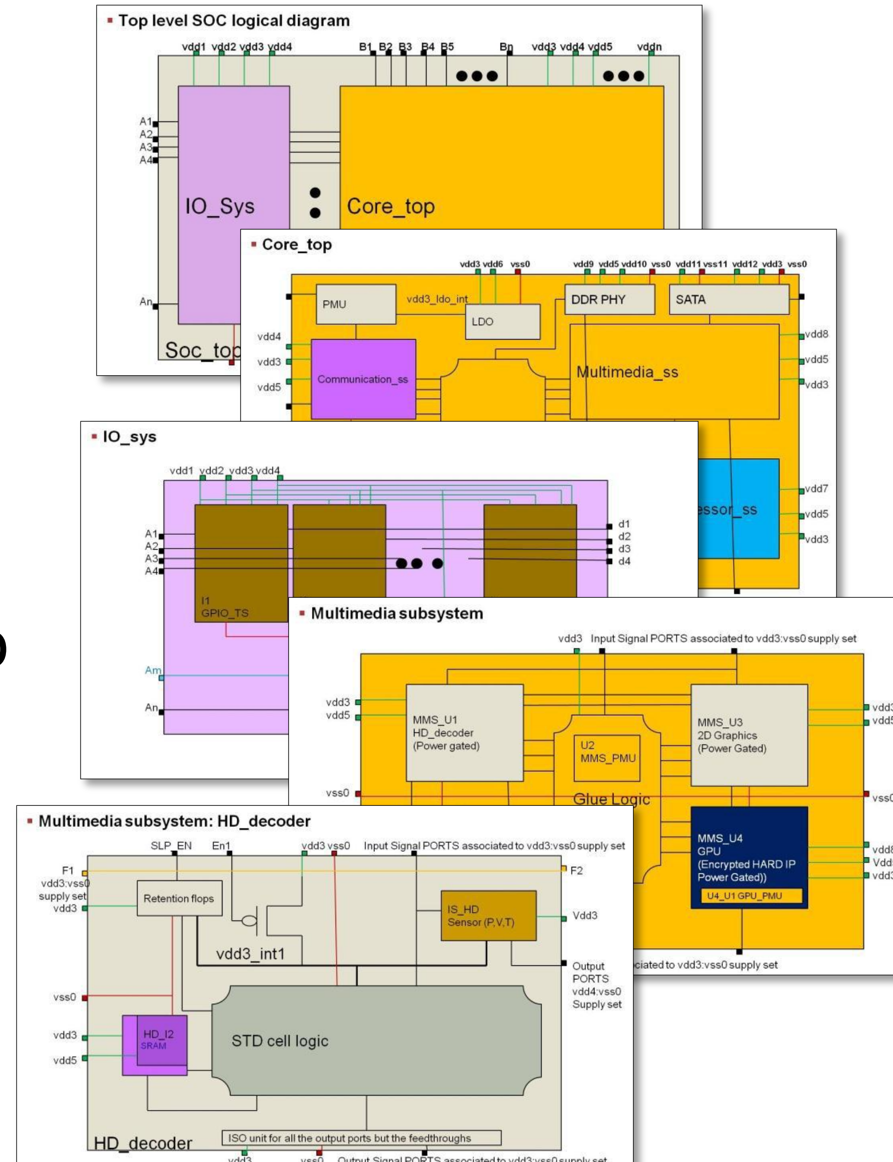
- Minimize Power Use
  - Power Management
- But Still Function Correctly
  - Power Aware Verification



Snapdragon™ Mobile Processor
http://www.qualcomm.com/chipsets/snapdragon

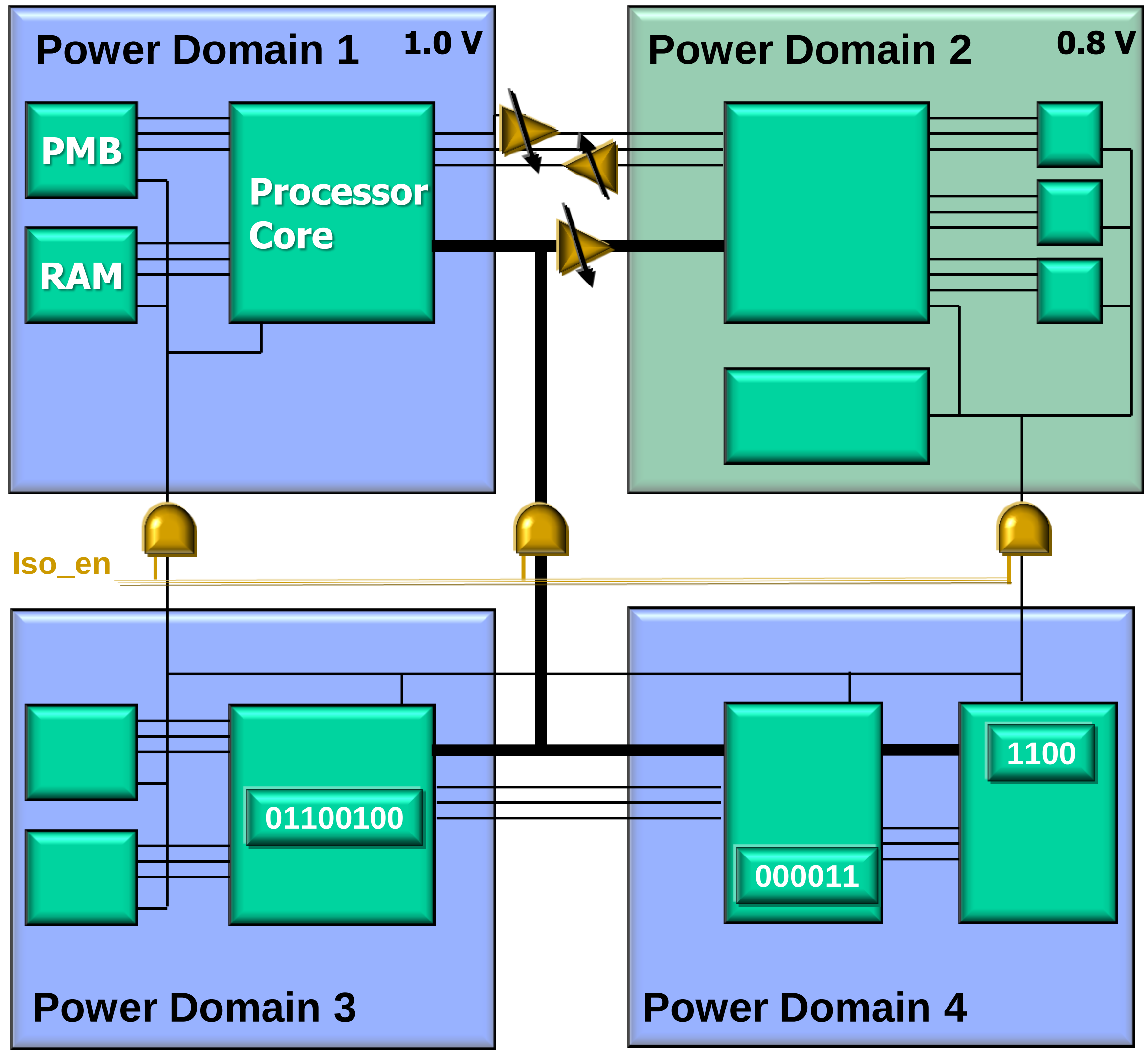
SoC Design

- Module-Based Methodology
  - Many IP blocks
  - System of Subsystems
- For Base Functionality
  - Divide and Conquer
  - Hierarchical Composition
- For Power Management Too
  - Power Intent for IP blocks
  - Power Intent for Subsystems
  - Power Intent for System



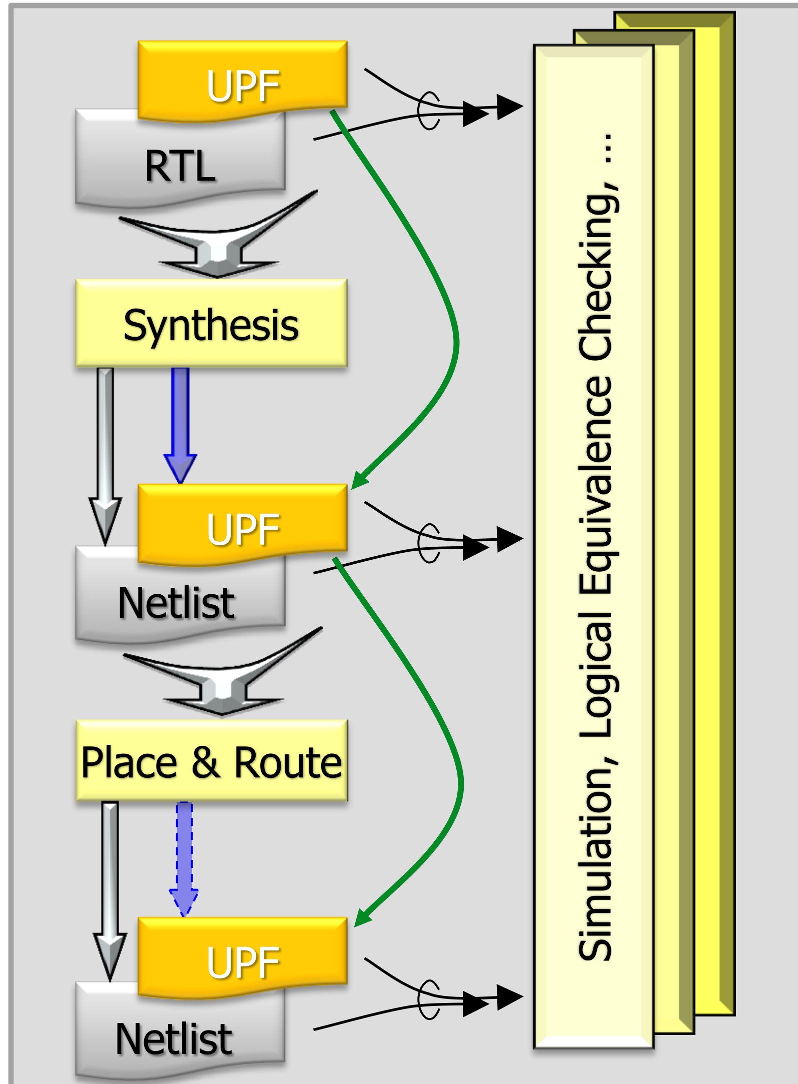
Power Aware Design and Verification

- Different Systems have different power management
- Power Gating
  - Isolation
  - Retention
- Multi-Voltage
  - Level Shifting
- Body Bias
  - Forward Bias
  - Reverse Bias
- Dynamic Voltage & Frequency Scaling
- UPF provides commands which can express the power management



IEEE 1801 Unified Power Format (UPF 2.0)

- RTL is augmented with a UPF specification
  - To define the power architecture for a given implementation
- RTL + UPF drives implementation tools
  - Synthesis, place & route, etc.
- RTL + UPF also drives power-aware verification
  - Ensures that verification matches implementation



IEEE Std 1801™-2009

The Need for UPF Methodology

Current Practice

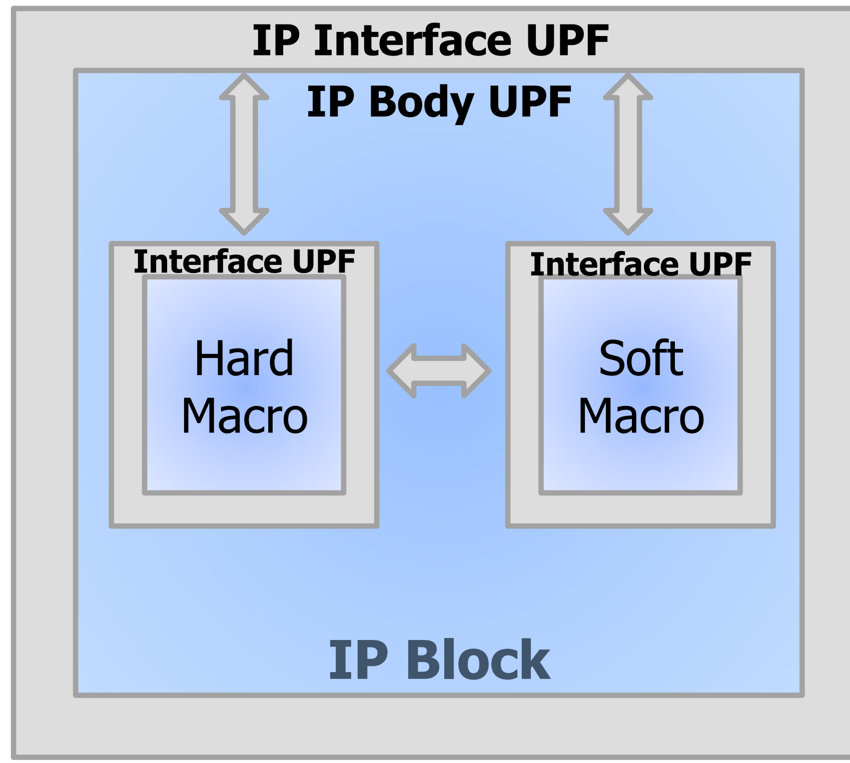
- UPF is written at one go (instance-based view of design)
  - Tedious and error prone
  - Doesn't scale well in an SoC environment
  - Relies on full system verification for correctness

UPF Methodology Should

- Align UPF creation with design process (block/module view of design)
  - Enable parallel development
    - Divide and conquer approach
    - Hierarchical partitioning and composition
  - Promote interoperability
  - Ensure success

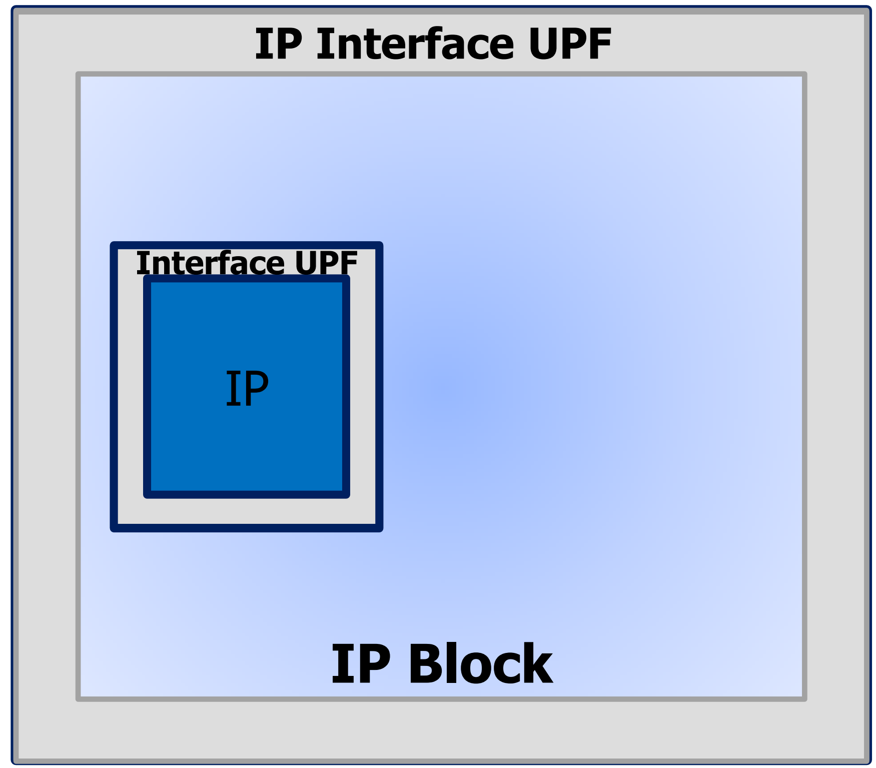
UPF 2.0 Methodology Overview

- Two components of Power Intent
  - Power Intent Interface
  - Power Intent Body
- Power Intent Interface
  - Provides abstract interface for any block (hard or soft)
  - Enables transparent replacement of soft IP with hard(ened) IP
  - Can remain stable throughout the flow
- Power Intent Body
  - Defines the actual behavior/implementation of the Power Intent
  - Only depends on Interface UPFs for external communication
  - Body for soft macro integrates lower level component interfaces



Methodology Concepts

- **Recursive** definition
  - Same concepts apply from at all levels of hierarchy
- **Module** (or model) based development
- Support different IP flavors
  - **Hard IPs**
  - **Soft IPs**
- IP **reuse** and **hierarchical composition**



Methodology Features

- **Top-level Power Domain** defines power interface
  - Supply Set handles on Power Domain represent supply interface
  - Supply constraints provide constraint checking during integration
  - Enables definition of IP block power states for use in larger context
- **Formal parameters** enable configuration/integration
  - Supply set handles for supply connections and port attributes
  - Logic ports for control connections
  - Hides implementation specific details
- **Information flows** down and up the hierarchy
  - Supply network structure, including embedded switches/LDOs
  - Supply set constraints and power states
    - Voltage constraints on input supplies
    - Power states on domains powered by input supplies
    - Power states on the block based on power domain states
- **Consistent modeling principles** for all blocks
  - Hard IPs or Soft IPs
  - Internal switches and LDOs

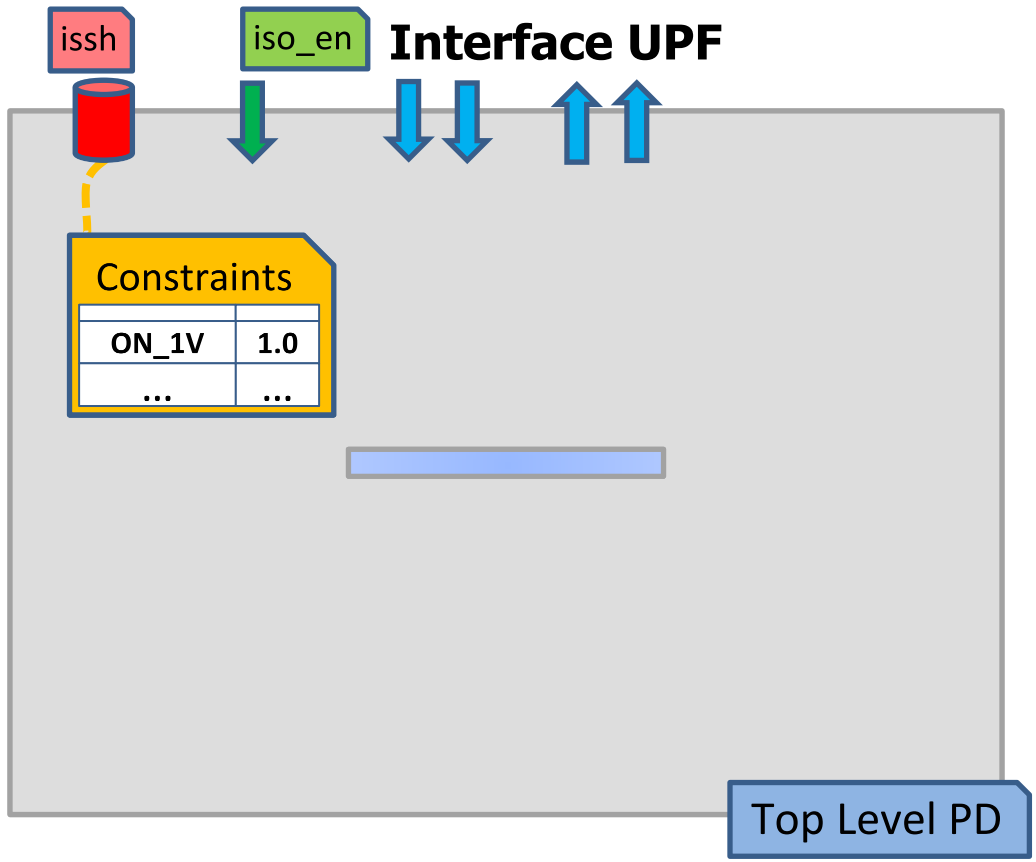
Interface Structure

Top-level power domain
create_power_domain PD \
-include_scope

Input supply sets
create_power_domain PD -update \
-supply { **issh** … }

Input supply constraints
add_power_state **PD.issh** \
-state **ON_1V** {
-supply_expr { **power == 1.0** … }
}

Input power control ports
create_logic_port **iso_en**



Interface Structure

Input power control ports

create_logic_port **sw_en**

Internal supply constraints

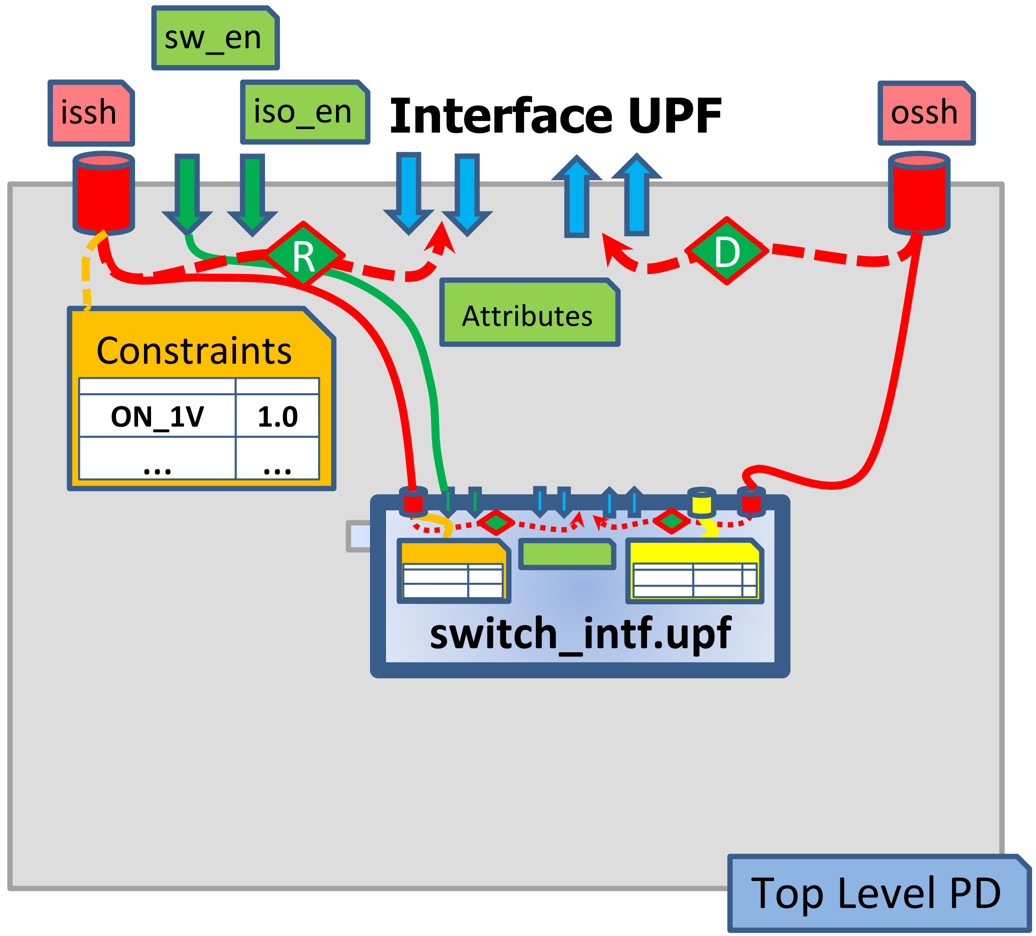
load_upf_protected **switch_intf.upf** \
–params { … } –hide_globals

Output supplies

create_power_domain PD –update \
-supply { **ossh** … }

Interface logic port constraints

set_port_attributes Pi –driver_supply…
set_port_attributes Pj –receiver_supply…



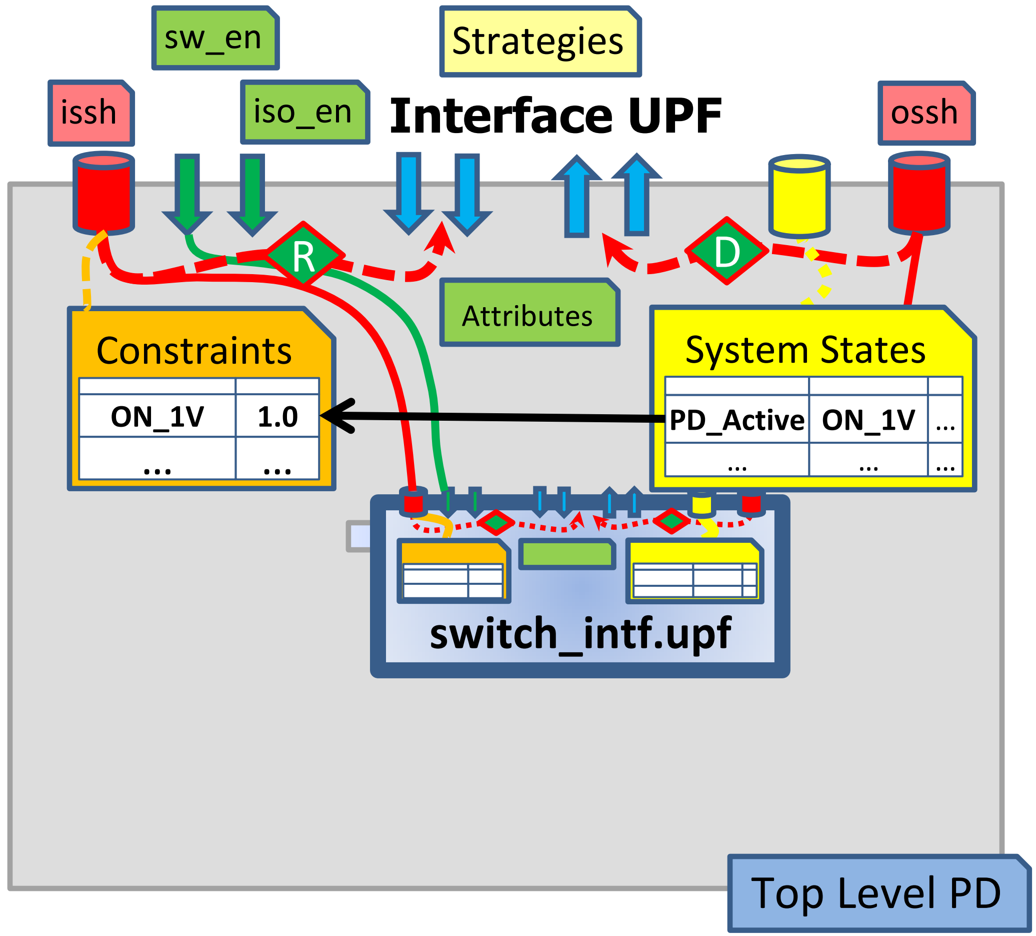
Interface Structure

Output system states

```
add_power_state PD \
  -state PD_Active {
   -logic_expr { PD.issh1 == ON_1V … }
  }
```

Interface strategies

```
set_isolation …
set_level_shifter …
set_retention …
```



Body Structure

Internal power domains

create_power_domain PDI \
 –elements { … }

Integrate subcomponent interfaces

Load interface definition

load_upf sub_ip.upf -scope sub_ip

Connect supplies

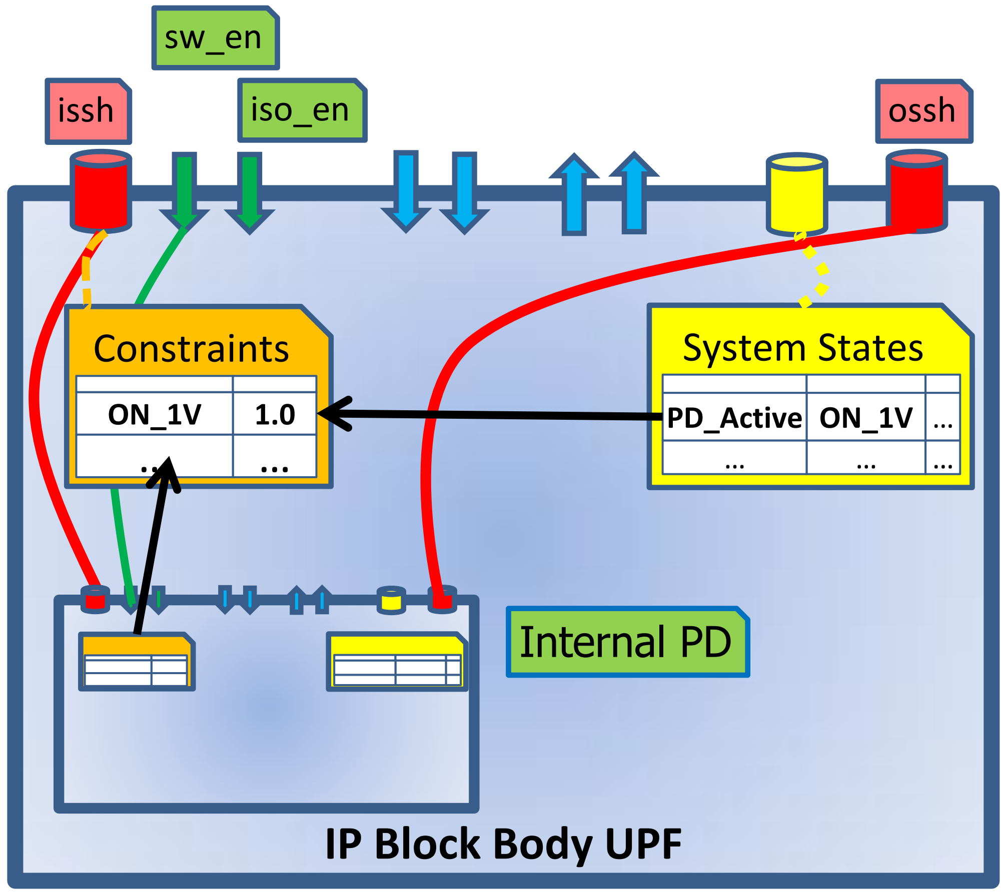
associate_supply_set PD.issh \
 -handle sub_ip/PD_ip.issh

Update supply constraints

add_power_state … # correlate parent and child power state defns

Connect control signals

connect_logic_net iso_en \
 -port { sub_ip/iso_en }



Body Structure

Load subcomponent bodies
(incl. switches/LDOs)

load_upf_protected … -params { … } \
–hide_globals

Define simstate behavior of
internal power domains

add_power_state PD.primary –update \
-state ON_1V \
{ -simstate NORMAL }

Define internal power domain
strategies

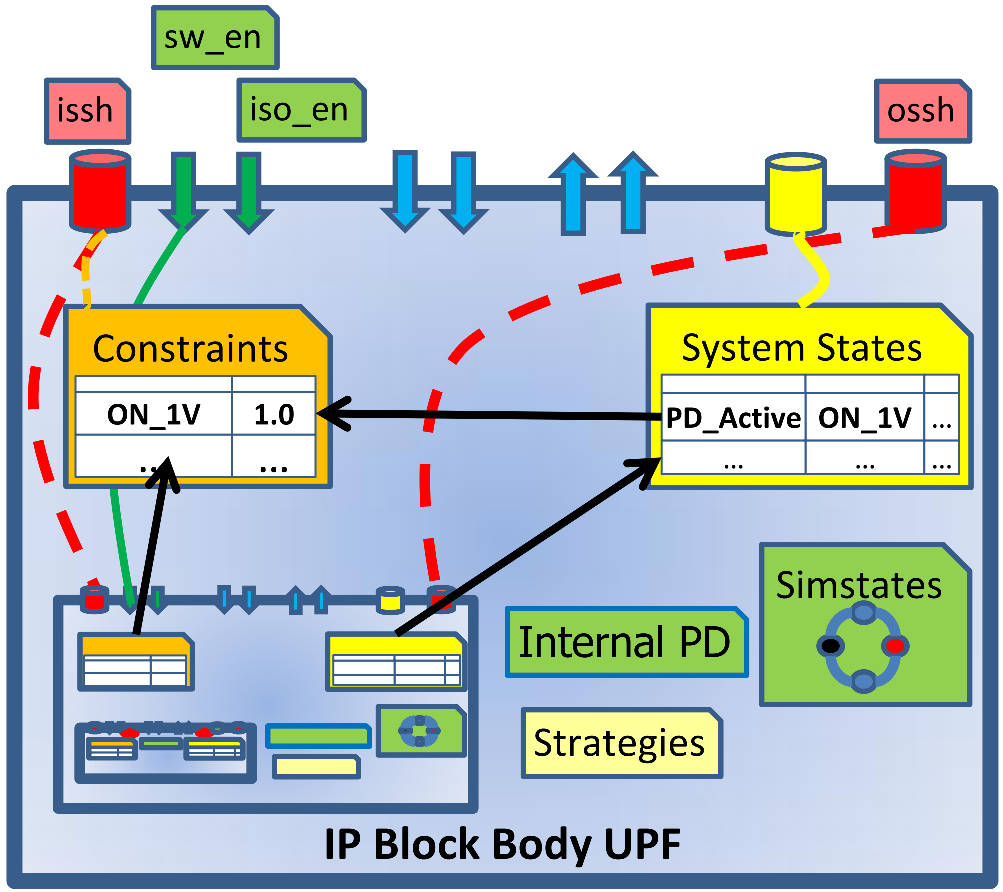
set_isolation …
set_level_shifter …
set_retention …

Specify automatic connections

connect_supply_set … \
-connect { issh pgtype }

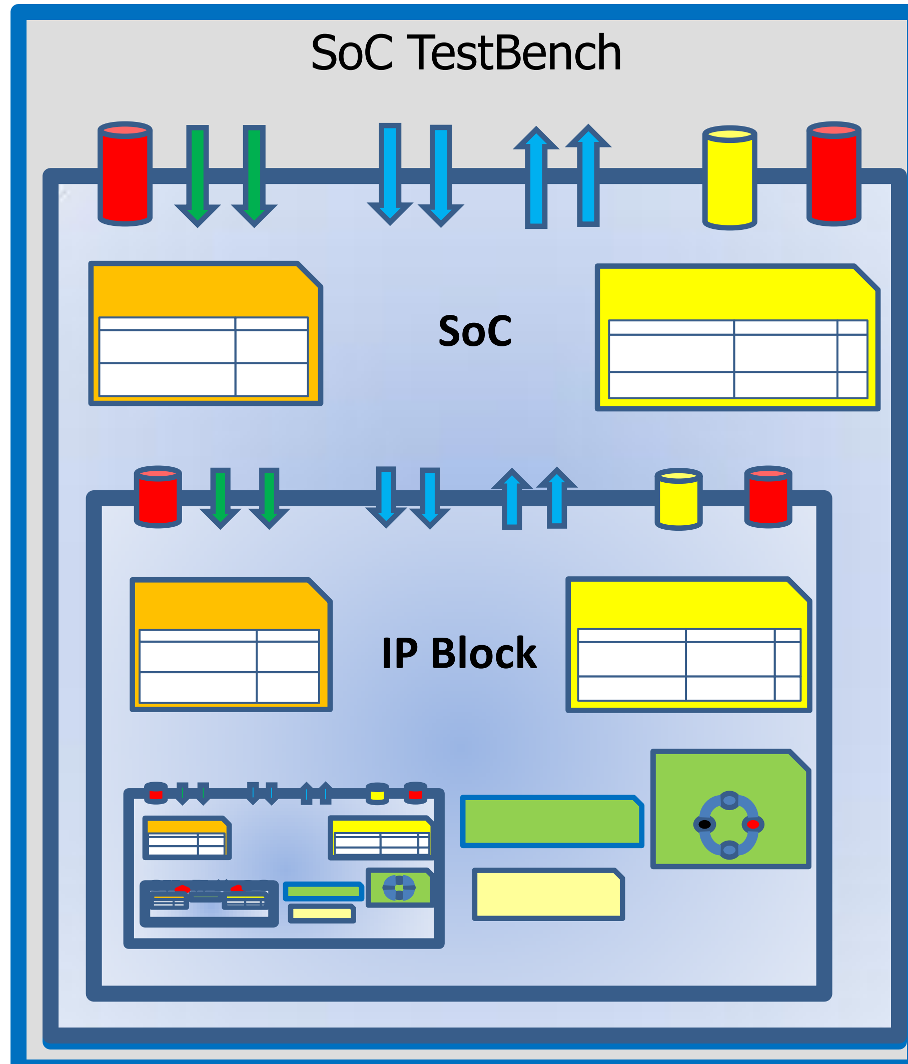
Update output power states
(defined in interface)

add_power_state PD –update …



Verification of IP at different Levels

- IP Level Verification
  - Integrate in TestBench
  - Perform **exhaustive** verification of IP
- System Level Verification
  - Integrate in SoC
  - Perform verification of full SoC similar to IP Level Verification



Methodology Highlights

- Defines constraints via add_power_state power state definitions
  - Supply expressions to define voltage constraints
  - Logic expressions to define control conditions and power state dependencies
- Enables constraint checking by tools
  - Statically during UPF processing
  - Dynamically during simulation
- Leverages UPF's progressive refinement capability
  - Allows efficient reuse and common UPF throughout the flow
  - Correct integration
  - Parallel development
- Combines the benefits of various approaches
  - Top-down
  - Bottom-up
- Leverages Tcl capabilities to increase automation
  - Parameterize UPF files for greater reuse
  - Reduces code by combining redundant steps
- Provides sufficient flexibility to cater to various needs
  - Provides different variations in UPF for each steps
- Supports both Hard IP and Soft IP
  - Goes beyond Liberty files to provide a complete power intent specification
- Enables Reusability and Portability
  - Gives IP providers a standard way to deliver a comprehensive power model for an IP block

Summary

- **Aligned** with IP based SoC design practices
- **Simplifies** verification complexity
- **Parallel development** of system and IP
- **Consistent** modeling for **Hard** and **Soft** IPs
- **Documents** power intent of IP for reuse
- Works within **UPF 2.0** constraints

Summary of Key UPF Commands

Define/Update Interface

- create_power_domain -include_scope -elements -supply -update
- create_logic_port
- set_port_attributes –driver_supply -receiver_supply

Load UPF for Instances/Power Elements

- load_upf –scope
- load_upf_protected -params -hide_globals

Connect Supply Sets/Nets

- associate_supply_set -handle
- connect_supply_set -connect
- connect_logic_net -port

Define/Update Constraints

- add_power_state -state -supply_expr -update

Define/Update/Correlate Power States

- add_power_state -state -logic_expr -update

Define/Update Simstates

- add_power_state -state -simstate -update

Define/Update Strategies

- set_isolation …
- set_level_shifter …
- set_retention …

THANK YOU

QUESTIONS ??

Amit Srivastava (amit_srivastava@mentor.com)
Rudra Mukherjee (rudra_mukherjee@mentor.com)
Erich Marschner (erich_marschner@mentor.com)
Chuck Seeley (chuck_seeley@mentor.com)
Sorin Dobre (sdobre@qualcomm.com)

Memory (Hard IP) Interface UPF - part 1

1. #Top level system PD

```
create_power_domain pd_SRAM -include_scope
```

2. # INPUT Supply Set   Handles

```
create_power_domain pd_SRAM -update \
   -supply { core_ssh } \
   -supply { primary }
```

3. # INPUT logic Ports – None

4. # INPUT Supply Set CONSTRAINTS

```
add_power_state pd_SRAM.primary \
   -state ON_Svs {
      -supply_expr {
        ( pd_SRAM.primary.power  == { FULL_ON, 0.7 } )
      && ( pd_SRAM.primary.ground == FULL_ON )
      }
   } \
   -state ON_Nom {
      -supply_expr {
        ( pd_SRAM.primary.power  == { FULL_ON, 0.8 } )
      && ( pd_SRAM.primary.ground == FULL_ON )
      }
   } \
   -state ON_Turbo {
      -supply_expr {
        ( pd_SRAM.primary.power  == { FULL_ON, 1.0 } )
      && ( pd_SRAM.primary.ground == FULL_ON )
      }
   } \
-state ON_Min {
      -supply_expr {
        ( pd_SRAM.primary.power  == { FULL_ON, 0.5 } )
      && ( pd_SRAM.primary.ground == FULL_ON )
      }
   } \
   -state OFF_STATE {
      -supply_expr {
      ( pd_SRAM.primary.power  == OFF )
      }
   }
```

Memory (Hard IP) Interface UPF - part 2

```
add_power_state pd_SRAM.core_ssh \
   -state ON_Svs_Normal {
      -supply_expr {
        ( pd_SRAM.core_ssh.power  == { FULL_ON, 0.855 } )
      && ( pd_SRAM.core_ssh.ground == FULL_ON )
      }
   } \
   -state ON_Turbo {
      -supply_expr {
        ( pd_SRAM.core_ssh.power  == { FULL_ON, 1.0 } )
      && ( pd_SRAM.core_ssh.ground == FULL_ON )
      }
   } \
   -state ON_Min {
      -supply_expr {
        ( pd_SRAM.core_ssh.power  == { FULL_ON, 0.65 } )
      && ( pd_SRAM.core_ssh.ground == FULL_ON )
      }
   } \
   -state OFF_STATE {
      -supply_expr {
      ( pd_SRAM.core_ssh.power  == OFF )
      }
   }
```

5. # INTEGRATE internal supplies – None

6. # OUTPUT Supply Set Handle – None

7. # OUTPUT System States

```
add_power_state pd_SRAM \
   -state PD_SRAM_Active {
      -logic_expr {
            (  ((pd_SRAM.primary  == ON_Turbo) && (pd_SRAM.core_ssh == ON_Turbo))
            || ((pd_SRAM.primary  == ON_Nom) && (pd_SRAM.core_ssh == ON_Svs_Normal))
            || ((pd_SRAM.primary  == ON_Svs) && (pd_SRAM.core_ssh == ON_Svs_Normal))
            )
         && (slp1          == 1'b0)
         && (slp2          == 1'b0)
         && (clamp_mem       == 1'b0)
      }
   } \
```

Memory (Hard IP) Interface UPF - part 3

```
-state PD_SRAM_Dormant {
    -logic_expr {
        (((pd_SRAM.primary  == ON_Turbo) && (pd_SRAM.core_ssh
== ON_Turbo))
        || ((pd_SRAM.primary  == ON_Nom) &&
(pd_SRAM.core_ssh == ON_Svs_Normal))
        || ((pd_SRAM.primary  == ON_Svs) && (pd_SRAM.core_ssh
== ON_Svs_Normal))
        || ((pd_SRAM.primary  == ON_Min) && (pd_SRAM.core_ssh
== ON_Min))
        )
      && (slp1          == 1'b1)
      && (slp2          == 1'b0)
      && (clamp_mem       == 1'b0)
    }
  } \
  -state PD_SRAM_Retention {
    -logic_expr {
        (pd_SRAM.primary  == OFF_STATE)
      && (pd_SRAM.core_ssh == ON_Min)
      && (clamp_mem       == 1'b1)
    }
  } \
```

```
-state PD_SRAM_OFF {
    -logic_expr {
        ( (pd_SRAM.primary == OFF_STATE) &&
          (pd_SRAM.core_ssh == OFF_STATE)
        ) || ( (slp1 == 1'b1) && (slp2 == 1'b1) &&
          (clamp_mem == 1'b0) )
    }
  }
```

8. # Port Constraints

```
set_port_attributes -ports { clamp_mem } \
  -receiver_supply pd_SRAM.core_ssh
set_port_attributes -domains { pd_SRAM -applies_to inputs } \
  -exclude_ports { clamp_mem } \
  -receiver_supply pd_SRAM.primary
set_port_attributes -domains { pd_SRAM -applies_to outputs } \
  -driver_supply pd_SRAM.primary
```

9. # Interface Rules

```
set_isolation iso_data -domain pd_SRAM \
  -elements [ find_objects . -pattern {din*,dout*, ad* }  -object_type
port ] \
  -isolation_signal { slp1 clamp_mem } -isolation_sense high \
  -clamp_value latch
set_isolation iso -domain pd_SRAM \
  -elements {slp1 slp2 } \
  -isolation_signal clamp_mem -isolation_sense high \
  -clamp_value latch
```

Memory (Hard IP) Body UPF

1. **# Internal PDs – None as its Hard Macro**
2. **# INTEGRATE sub-component – None**
3. **# LOAD body UPF of sub-component – None**
4. **# Define simstates – Not required as it's a hard macro**
5. **# Specify Internal Rules – Not required**
6. **# Automatic Connection Semantics**

```
# This information optional if the attributes are from liberty
set_port_attributes -ports { vddmx } -pg_type pg_sram_array
set_port_attributes -ports { vddx }  -pg_type primary_power
set_port_attributes -ports { vss0 }  -pg_type primary_ground

connect_supply_set pd_SRAM.primary \
   -connect { power  {primary_power } } \
   -connect { ground {primary_ground} }

#since ground is common, hence not connecting it again
connect_supply_set pd_SRAM.core_ssh \
   -connect { power {pg_sram_array} }
```

7. **# UPDATE System States – Not needed**

Decoder (Soft IP) Interface UPF - 1

1. # Top level PD

```
create_power_domain pd_DEC -include_scope
```

2. # INPUT Supply Set Handles

```
create_power_domain pd_DEC -update \
   -supply { aon_ssh } \
   -supply { sram_ssh }
```

3. # INPUT Control Ports

```
create_logic_port En1
create_logic_port slp_en
```

4. # INPUT Supply Constraints

```
add_power_state pd_DEC.aon_ssh \
   -state ON_1d2V {
      -supply_expr {
        ( pd_DEC.aon_ssh.power  == { FULL_ON, 1.2 } )
      && ( pd_DEC.aon_ssh.ground == FULL_ON )
      }
   }
add_power_state pd_DEC.sram_ssh \
   -state ON_Svs_Normal {
      -supply_expr {
        ( pd_DEC.sram_ssh.power  == { FULL_ON, 0.855 } )
      && ( pd_DEC.sram_ssh.ground == FULL_ON )
      }
   }
…
```

5. # INTEGRATE Internal Supplies

```
load_upf_protected upf/switch_interface.upf -params {
      {sw_op_sset ss_DEC_switchable}
      {sw_ip_sset pd_DEC.aon_ssh}
      {sw_en En1}
      {inp_ss_state_list {ON_1d2V}}
   } -hide_globals
```

6. # OUTPUT Supply Set Handles

```
create_power_domain pd_DEC -update \
   -supply { primary ss_DEC_switchable }
```

7. # OUTPUT System States

```
add_power_state pd_DEC \
   -state MOD_S1 { -logic_expr { <based on supply set handle states> } } \
   -state MOD_S2 { -logic_expr { <based on supply set handle states> } } \
   …
```

8. # ** INTERFACE: PORT Constraints

```
set_port_attributes …
```

9. # ** INTERFACE: Isolation/Level Shifter/Retention of INTERFACE PD

```
set_retention …
set_isolation …
set_level_shifter …
```

Decoder (Soft IP) Body UPF - 1

```
upf_version 2.0              ;# optional
set_design_top  MOD   ;# body will usually apply to a single module type

# ** BODY: DEFINE Internal Power Domains
# create_power_domain intPD1 …

# ** BODY: INTEGRATE Sub-component IP and Internal Supplies

# ** INTEGRATE: LOAD Interface UPF
load_upf_protected <upf for an instance> -scope <scope of the instance>

# ** INTEGRATE: CONNECT Supplies
associate_supply_set pd_DEC.ssh1  -handle <ssh of the instance PD>
…

# ** INTEGRATE: UPDATE Supplies Constraints
# add_power_state pd_DEC.sram_ssh –update …

# ** INTEGRATE: CONNECT Logic Controls
# NA

# ** BODY: LOAD BODY of sub-component IP and Internal Supplies
load_upf_protected switch_body.upf -params {
        {sw_op_sset ss_DEC_switchable}
        {sw_ip_sset pd_DEC.aon_ssh}
        {sw_en En1}
        {inp_ss_state_list {ON_1d2V}}
        {domain pd_DEC}
    } -hide_globals

# ** BODY: DEFINE SIMSTATE BEHAVIOR for Internal and INTERFACE PDs
# ** BODY: Specify ISOLATION,  LEVEL SHIFTER and Retention with controls connected
# ** BODY: Automatic connections for hard macros
set_port_attributes -ports { is_hd_sensor_i/vdd } -pg_type primary_power
set_port_attributes -ports { is_hd_sensor_i/vss } -pg_type primary_ground
connect_supply_set pd_DEC.aon_ssh \
    -elements is_hd_sensor_i \
    -connect { power  {primary_power} } \
    -connect { ground {primary_ground} }

# ** BODY: Update SYSTEM STATES based on sub-component IPs
```