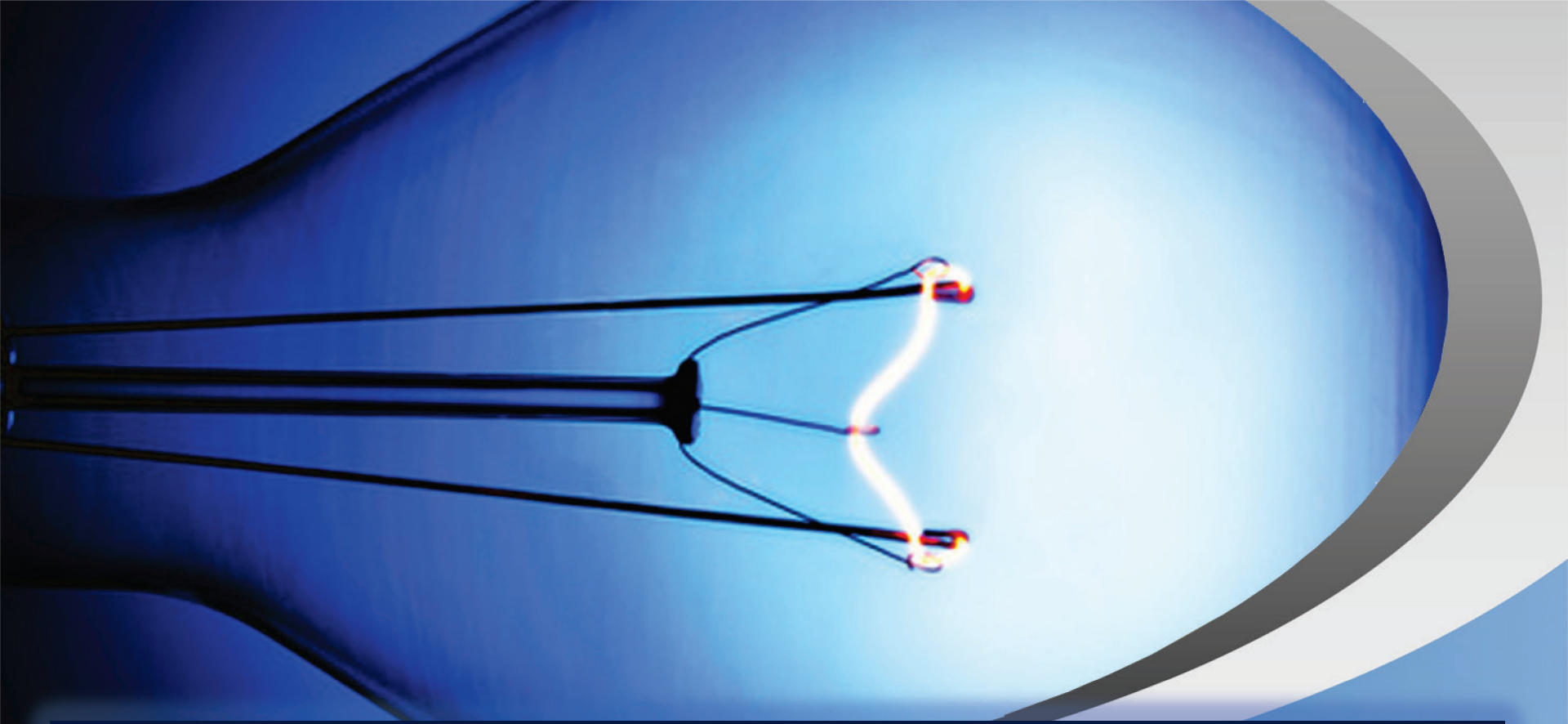


Low Power Design, Verification, and Implementation with IEEE 1801™ UPF

Tutorial presented by members of the IEEE P1801 WG

DVCon 2013
Design & Verification Conference & Exhibition

UPF UNIFIED POWER FORMAT

IEEE

accellera
SYSTEMS INITIATIVE

Introduction

Erich Marschner

Verification Architect

Mentor Graphics

Introduction

- Welcome and Logistics
- Low Power Design Challenges
- The Unified Power Format (UPF)
- IEEE P1801 Working Group
- Tutorial Agenda

Welcome and Logistics

Welcome

- **This is Tutorial 3:**

  Low Power Design, Verification, and Implementation with IEEE 1801™ UPF

Logistics

- **Please ask questions during each talk.**
- **Please fill out the survey questionnaire at the end of the tutorial.**

Low Power Design Challenges

- **Consumer Demand**
  - More Features and Performance
  - Longer Battery Life
  - Less Heat
- **Technology Trends**
  - Increasing leakage at lower device nodes
  - Dominates below 65nm
- **Modeling Power Management**
  - HDL semantics do not consider power
  - Traditional flows assume that power is always on
- **Power Management Complexity**
  - Can lead to functional errors
  - Requires more verification
  - Requires earlier verification

The Unified Power Format (UPF)

- **Standard format for defining power intent**
  - Developed by Accellera and IEEE
- **Extension of Tcl tool command language**
  - Familiar syntax, used with Tcl
- **Enables early verification of power intent**
  - To find and fix power management issues as soon as possible
- **Drives verification and implementation**
  - from RTL to Layout

IEEE P1801 UPF Working Group

- **Officers:**
  - Chair: John Biggs
  - Vice-Chair: Erich Marschner
  - Secretary: Jeffrey Lee
- **Working Group Entity Members:**

Accellera	CSR	PMC-Sierra	
AMD	Intel	Qualcomm	
ARM	LSI	Si2	
Atrenta	Marvell	ST	
Broadcom	MediaTek	Synopsys	
Cadence	Mentor	TI	(tutorial presenters)

Tutorial Agenda

- **Introduction**
  - Erich Marschner, Mentor
- **Low Power Design and Verification**
  - Jeffrey Lee, Synopsys
- **Unified Power Format (UPF)**
  - Erich Marschner, Mentor
- **BREAK**
- **Power Intent Specification Methodology**
  - Qi Wang, Cadence
- **Using UPF for IP Design**
  - John Biggs, ARM
- **Using UPF for System Design**
  - Sushma Honnavara-Prasad, Broadcom
- **What's New in UPF 2.1**
  - Qi Wang, Cadence

Low Power Design and Verification

Jeffrey Lee

Staff Corporate Application Engineer

Synopsys

Functional Intent vs. Power Intent

What is the difference?

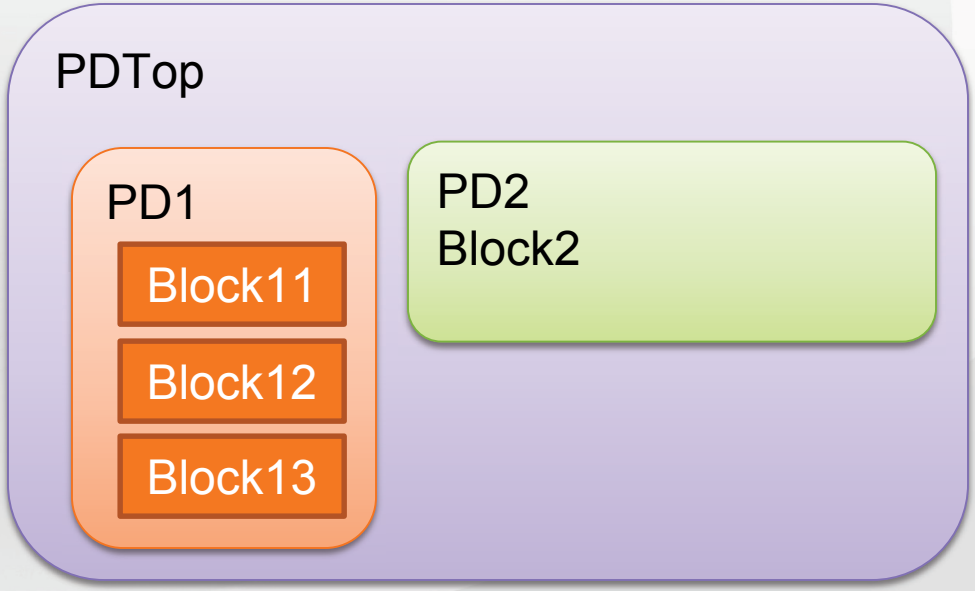
Functional intent specifies	Power intent specifies

• **Architecture**
  • Design hierarchy
  • Data path
  • Custom blocks
• **Application**
  • State machines
  • Combinatorial logic
  • I/Os
  • EX: DSP, Cache
• **Usage of IP**
  • Industry-standard interfaces
  • Memories
  • etc
	• **Power distribution architecture**
	  • Power domains
	  • Supply rails
	  • Shutdown control
	• **Power strategy**
	  • Power state tables
	  • Operating voltages
	• **Usage of special cells**
	  • Isolation cells, Level shifters
	  • Power switches
	  • Retention registers

Captured in RTL	Captured in UPF

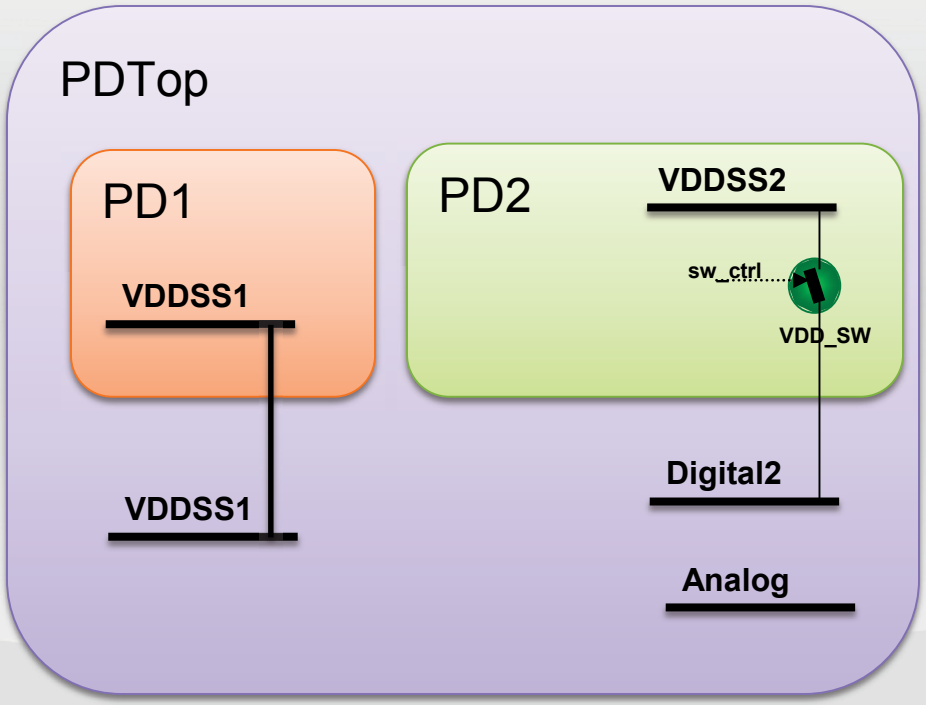
Power Domains

- Defined as:
  - A collection of instances that are treated as a group for power management purposes. The instances of a power domain typically, but do not always, share a primary supply set. ...



Power Domain PD1 contains 3 instances
Power Domain PD2 contains only Block2

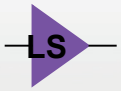
Power Supply Network



- **Describes logical connectivity of the power supplies, or power rails in your design**
  - May includes connectivity through power switches

Power Management Cells

- **Level Shifter**



  - Translates signal values from an input voltage swing to a different output voltage swing

- **Isolation Cell**

  - Passes logic values during normal mode operation and clamps its output to some specified logic value when a control signal is asserted

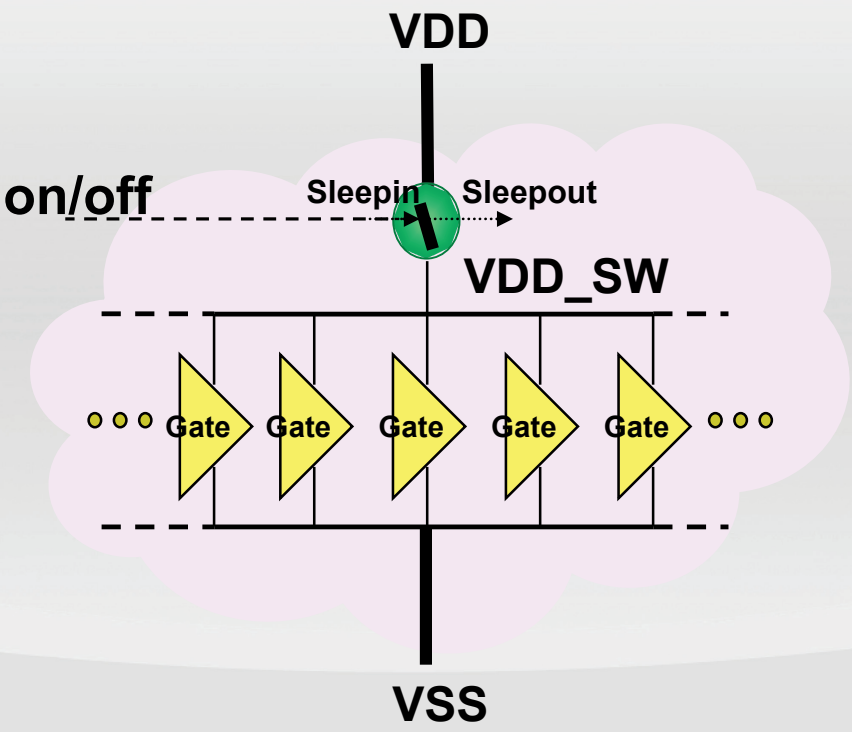
- **Retention register**

  - Extends the functionality of a sequential element with the ability to retain its memory value during the power-down state

Power Management Techniques

- Power Gating
- Multi-voltage
- Dynamic Voltage and Frequency Scaling

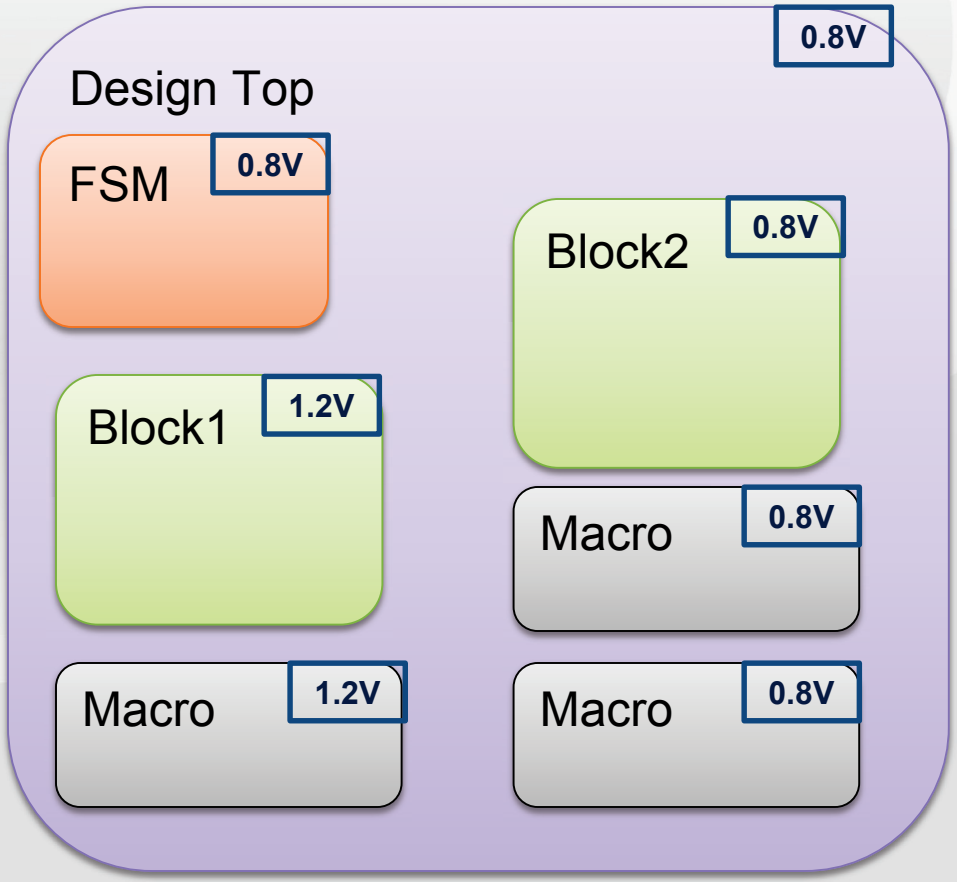
Power Gating



- Power reduction technique to save leakage power by shutting off, or powering down, unnecessary logic
- Enabled by power switch, or MTCMOS, cells

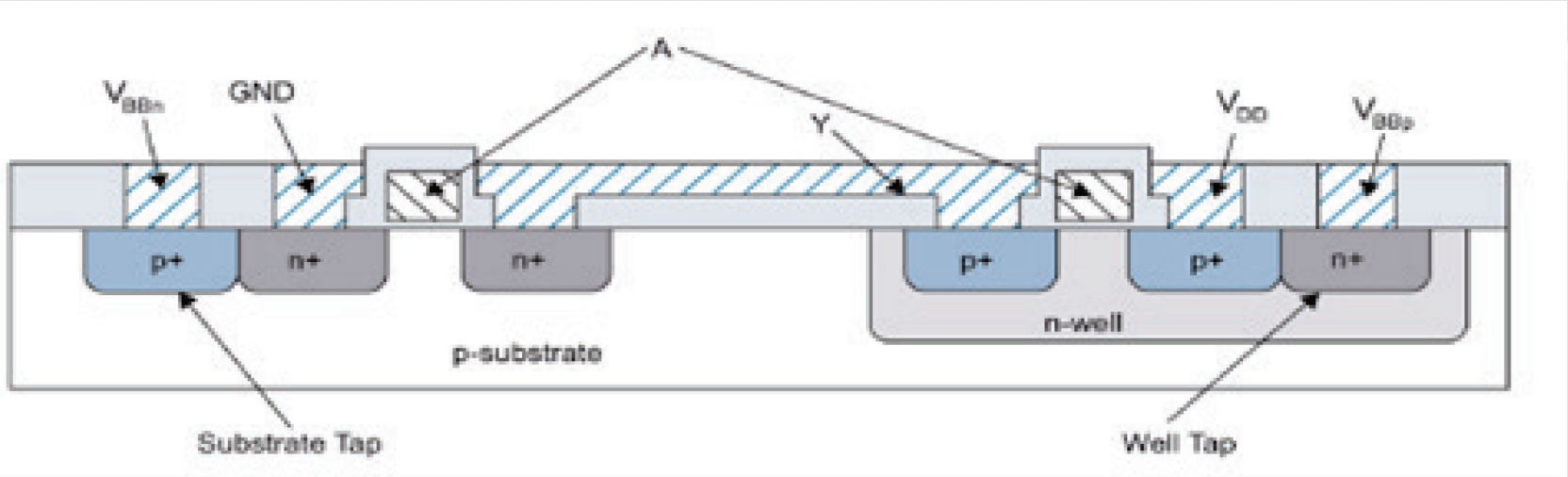
Multi-Voltage

- **Power savings technique to operate different blocks of logic at different voltages**
  - Less critical blocks can be operated at a lower voltage for power savings



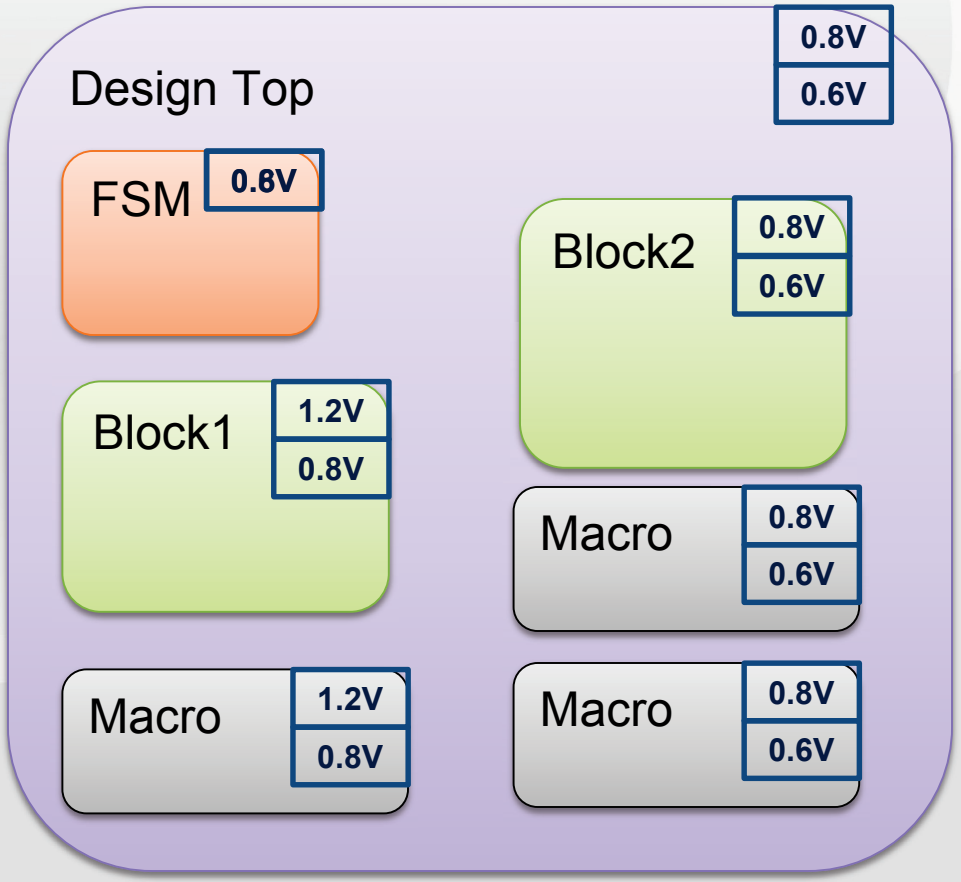
Bias Voltage

- Used to change the threshold voltage of a cell to improve the leakage characteristics of the cell



Dynamic Voltage and Frequency Scaling

- **Power Saving Technique to change the voltage and/or clock frequency while the chip is running to save power**

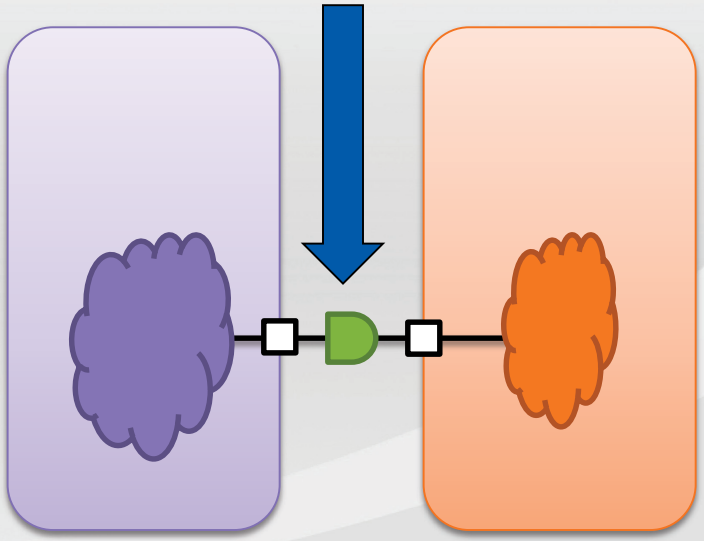


Power Management Architecture

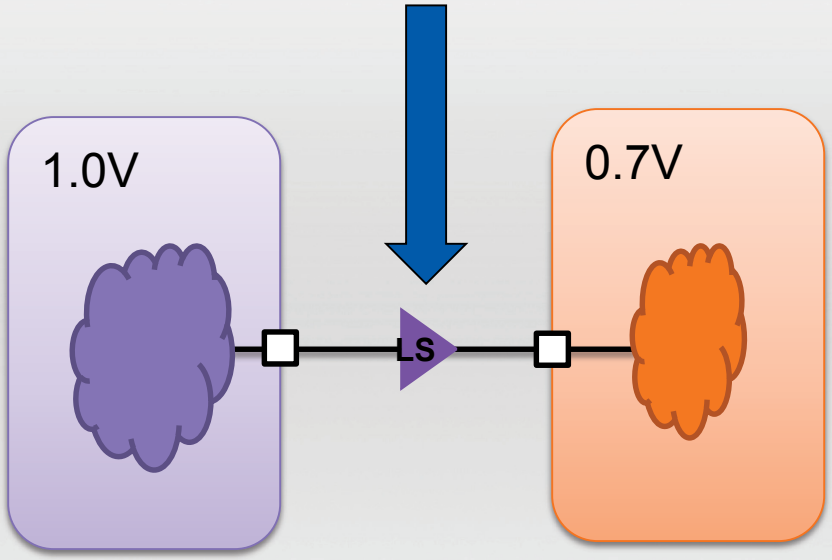
- Power States and Transitions
- Isolation and Level Shifting
- State Retention

Isolation Cells

- **Isolation cells are typically used to protect logic that is powered on from logic that is powered off**
  - Used to prevent unknown values in unpowered logic from propagating into live logic
  - Can also be used to prevent leakage current from live logic from improperly powering unpowered logic



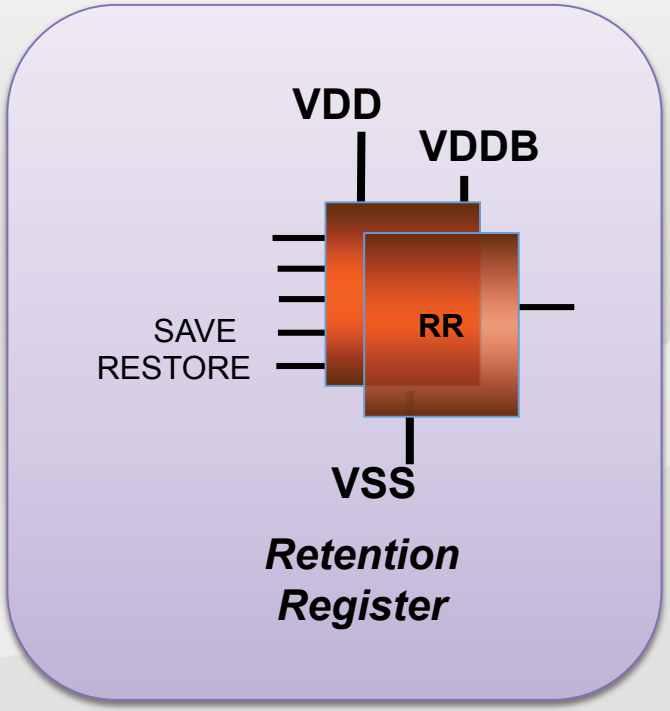
Level Shifters



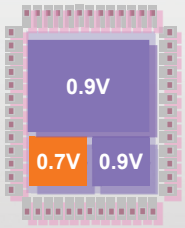
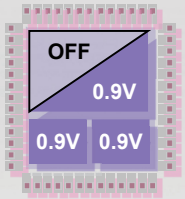
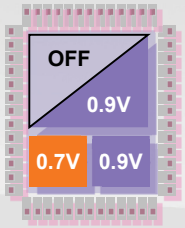
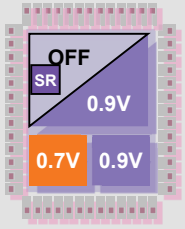
- **Changes the voltage from one discrete value to another discrete value**
  - A 1'b1 driven by 1.0V logic may be too much for 0.7V logic and likewise a 1'b1 from 0.7V logic may not translate into a 1'b1 for 1.0V logic
  - A level shifter changes a 0.7V 1'b1 to a 1.0V 1'b1 so you are propagating valid digital values through the circuit

State Retention

- **A sequential element that can retain its value despite being powered off**
  - Useful to recover the last known state of the design when power was removed
  - Reduces the amount of time needed reset a design to a specific state to continue operation



Multi-Voltage Special Cell Requirement

	Level Shifters	Isolation Cells	Power Switches (MTCMOS)	Retention Registers	Always-on Logic
 Multiple Voltage (MV) Domains	✓				
 Multi-Supply with Shutdown No State Retention		✓	✓		
 Multi-Voltage with Shutdown	✓	✓	✓		
 Multi-Voltage with Shutdown & State Retention	✓	✓	✓	✓	✓

Unified Power Format (UPF)

Erich Marschner

Verification Architect

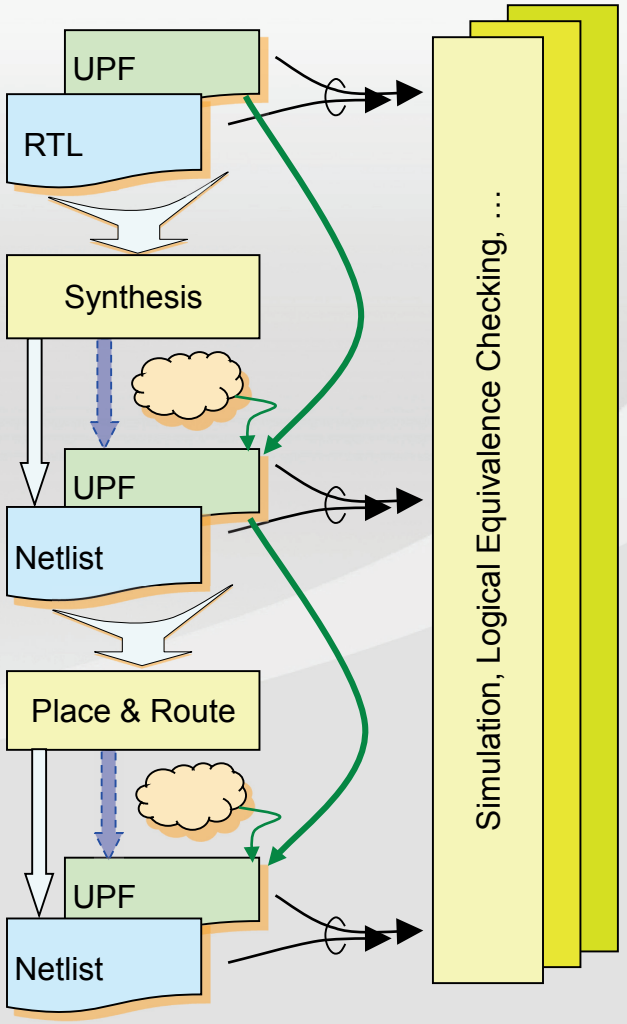
Mentor Graphics

What is UPF?

- **An Evolving Standard**
  - Accellera UPF in 2007 (1.0)
  - IEEE 1801-2009 UPF (2.0)
  - IEEE 1801-2013 UPF (2.1)
    - (Not Yet Approved)
- **For Power Intent**
  - To define power management
  - To minimize power consumption
  - Through control of leakage
- **Based upon Tcl**
  - Tcl syntax and semantics
  - Can be mixed with non-UPF Tcl
- **And HDLs**
  - SystemVerilog, Verilog, VHDL
- **For Verification**
  - Simulation or Emulation
  - Static/Formal Verification
- **And for Implementation**
  - Synthesis, DFT, P&R, etc.

Low Power Design & Verification Flow

- **RTL is augmented with UPF**
  - To define the power architecture for a given implementation
- **RTL + UPF verification**
  - To ensure that the power architecture is complete and consistent with expected power states of the design
  - To ensure that the design will work correctly under power management with this power architecture
- **RTL + UPF implementation**
  - Synthesis, test insertion, place & route, etc.
  - UPF may be updated by the user or the tool
- **NL + UPF verification**
  - Power aware equivalence checking, static analysis, simulation, emulation, etc.



Accellera UPF (2007)

Navigation:
- set_scope
- set_design_top

Supply Nets:
- create_supply_port
- create_supply_net
- connect_supply_net
- create_power_switch

Power Domains:
- create_power_domain
- set_domain_supply_net

Power States:
- add_port_state
- create_pst
- add_pst_state

Strategies:
- set_retention
- set_retention_control
- set_isolation
- set_isolation_control
- set_level_shifter

Implementation:
- map_retention_cell
- map_isolation_cell
- map_level_shifter_cell
- map_power_switch_cell

HDL Interface:
- bind_checker
- create_hdl2upf_vct
- create_upf2hdl_vct

Code Management:
- upf_version
- load_upf
- save_upf

IEEE 1801-2009 UPF

Navigation:
- set_scope
- set_design_top

Supply Nets:
- create_supply_port
- create_supply_net
- connect_supply_net
- create_power_switch

Power Domains:
- create_power_domain
- set_domain_supply_net
- create_composite_domain

Power States:
- add_port_state
- create_pst
- add_pst_state
- add_power_state
- describe_state_transition

Simstates:
- add_power_state
- set_simstate_behavior

Attributes:
- set_port_attributes
- set_design_attributes
- HDL and Liberty attributes

Supply Sets:
- create_supply_set
- supply set handles
- associate_supply_set
- connect_supply_set

Strategies:
- set_retention_elements
- set_retention
- set_retention_control
- set_isolation
- set_isolation_control
- set_level_shifter

Implementation:
- map_retention_cell
- map_isolation_cell
- map_level_shifter_cell
- map_power_switch_cell
- use_interface_cell

Control Logic:
- create_logic_port
- create_logic_net
- connect_logic_net

HDL Interface:
- bind_checker
- create_hdl2upf_vct
- create_upf2hdl_vct

Code Management:
- upf_version
- load_upf
- save_upf
- load_upf_protected
- load_simstate_behavior
- find_objects

IEEE 1801-2013 UPF* *Not Yet Approved

Navigation:
- set_scope
- set_design_top

Supply Nets:
- create_supply_port
- create_supply_net
- connect_supply_net
- create_power_switch

Power Domains:
- create_power_domain
- set_domain_supply_net
- create_composite_domain

Power States:
- add_port_state
- create_pst
- add_pst_state
- add_power_state
- describe_state_transition

Simstates:
- add_power_state
- set_simstate_behavior

Attributes:
- set_port_attributes
- set_design_attributes
- HDL and Liberty attributes

Supply Sets:
- create_supply_set
- supply set handles
- associate_supply_set
- connect_supply_set
- set_equivalent

Strategies:
- set_repeater
- set_retention_elements
- set_retention
- set_retention_control
- set_isolation
- set_isolation_control
- set_level_shifter

Implementation:
- map_retention_cell
- map_isolation_cell
- map_level_shifter_cell
- map_power_switch_cell
- use_interface_cell

Control Logic:
- create_logic_port
- create_logic_net
- connect_logic_net

HDL Interface:
- bind_checker
- create_hdl2upf_vct
- create_upf2hdl_vct

Code Management:
- upf_version
- load_upf
- save_upf
- load_upf_protected
- load_simstate_behavior
- find_objects
- begin_power_model
- end_power_model
- apply_power_model

Power Management Cells:
- define_always_on_cell
- define_diode_clamp
- define_isolation_cell
- define_level_shifter_cell
- define_power_switch_cell
- define_retention_cell

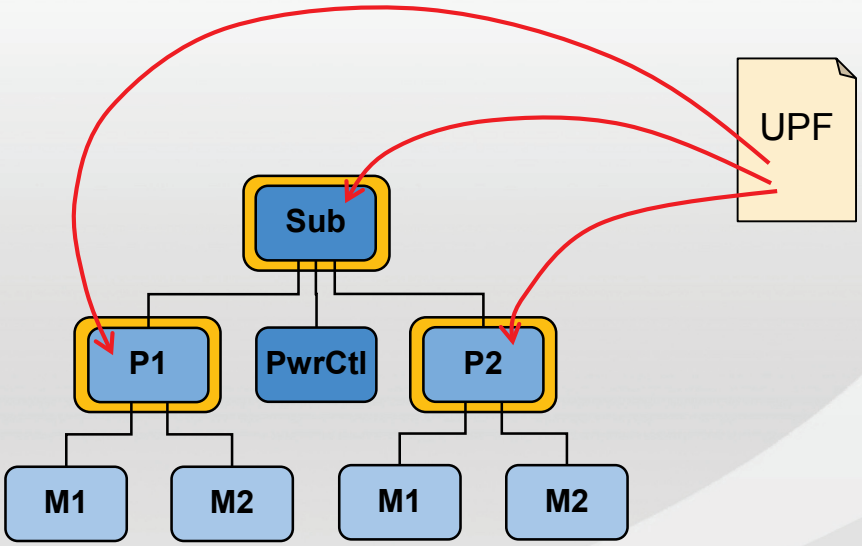
Basic Concepts

Features of the Accellera subset of IEEE 1801 UPF (1.0)

- Logic Hierarchy
- Navigation
- Supply Network
- Power Domains
- Power State Tables
- Power Management Strategies
- Verifying with UPF
- Implementing with UPF

Logic Hierarchy

- Design Hierarchy
  - Instances, generate stmts, block stmts, etc.
- Logic Hierarchy
  - Instances only
  - UPF objects
    - are created in instance scopes
    - can be referenced with hierarchical names
- Mapping to Floorplan
  - May or may not reflect implementation
    - Depends upon the user and tools



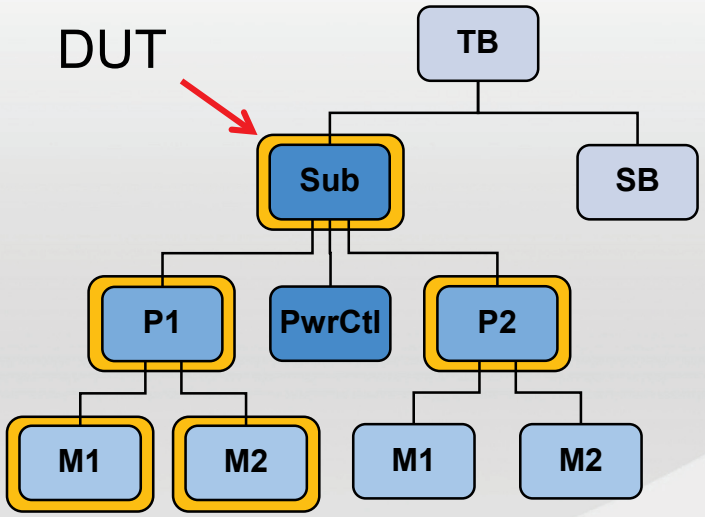
Navigation

- **set_design_top**

```
set_design_top TB/Sub
```

- **set_scope**

```
set_scope .
set_scope P1/M1
set_scope ..
set_scope M2
set_scope /P2
```

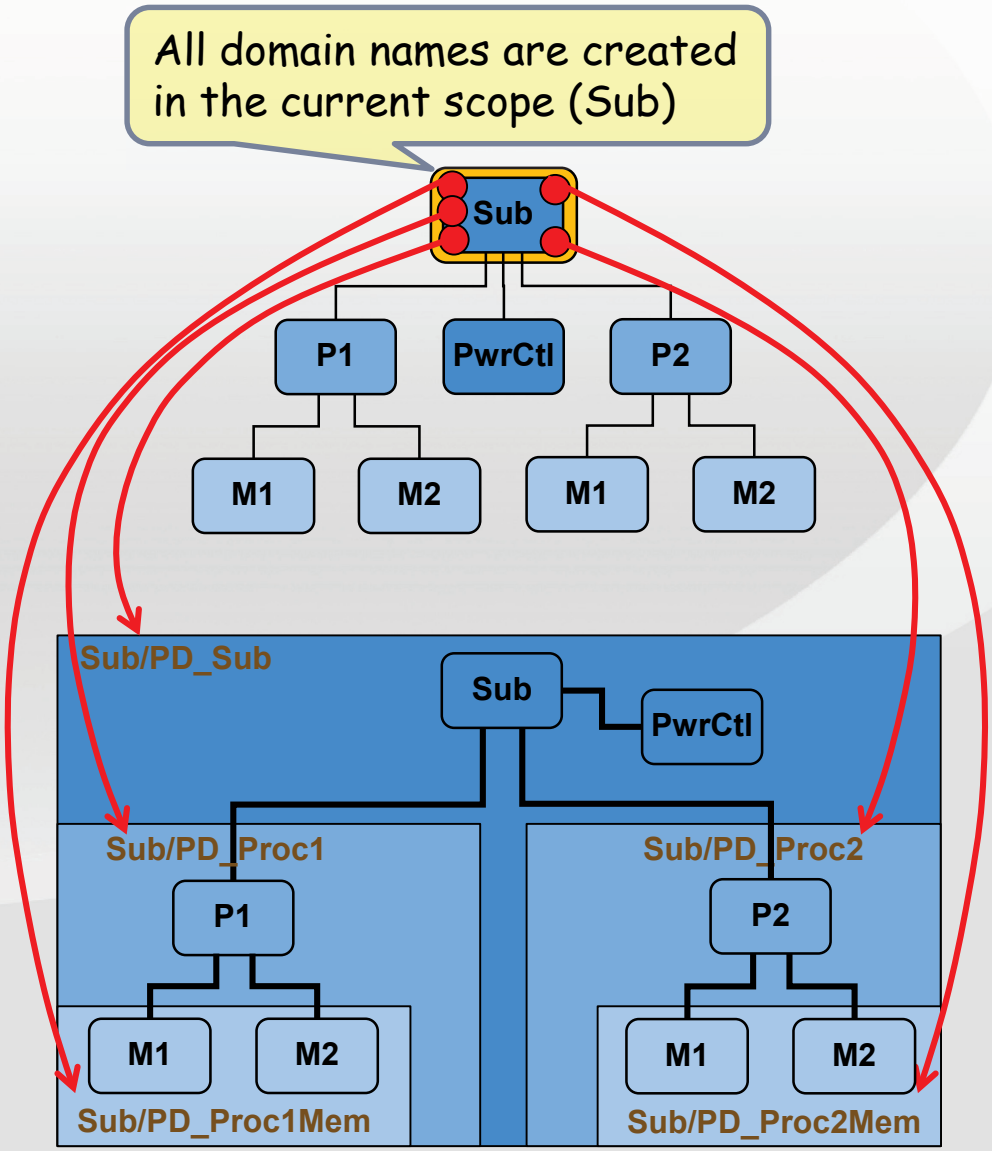


Note:
These are all instance names

Power Domains

- **create_power_domain**

```
set_scope Sub
create_power_domain PD_Sub \
   -include_scope

create_power_domain PD_Proc1 \
   -elements {P1}
create_power_domain PD_Proc1Mem \
   -elements {P1/M1 P1/M2}

create_power_domain PD_Proc2 \
   -elements {P2}
create_power_domain PD_Proc2Mem \
   -elements {P2/M1 P2/M2}
```

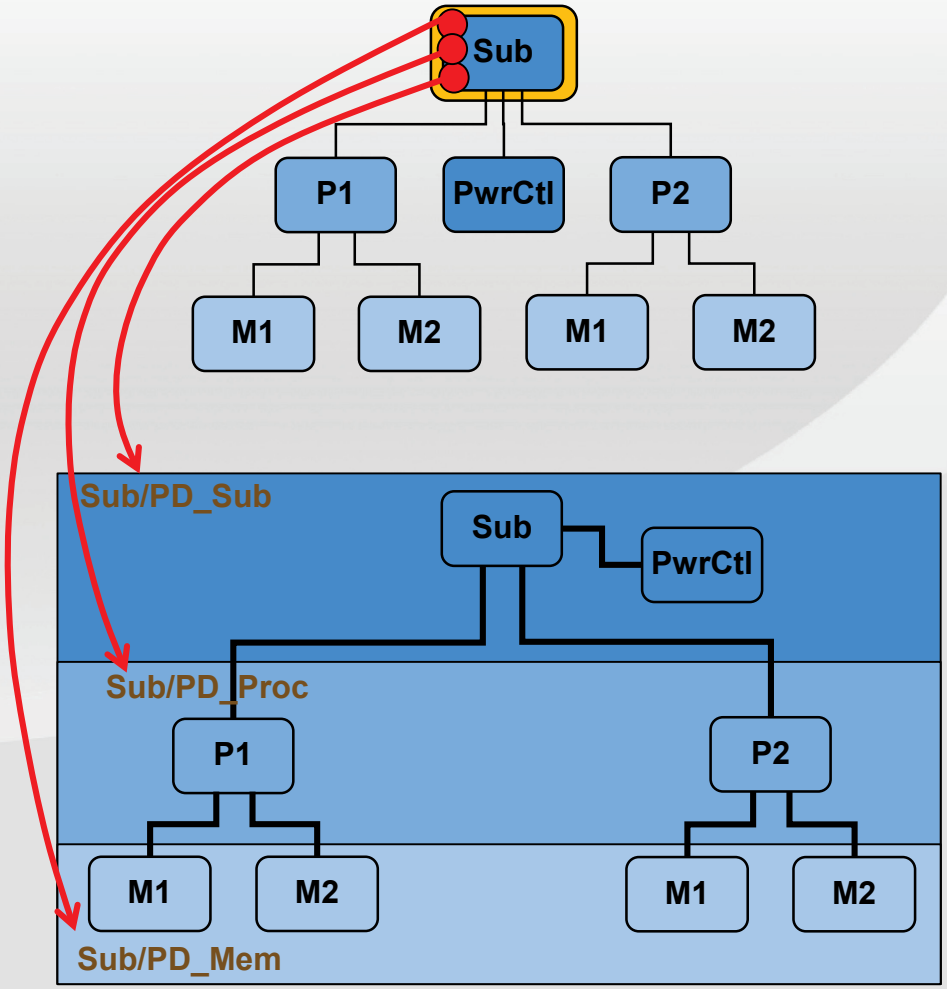


Power Domains 2

- create_power_domain

```
set_scope Sub
create_power_domain PD_Sub \
   -include_scope
create_power_domain PD_Proc \
   -elements {P1 P2}
create_power_domain PD_Mem \
   -elements {P1/M1 \
              P1/M2 \
              P2/M1 \
              P2/M2 }
```

Another way to partition



Supply Network

- **create_supply_port**

```
create_supply_port VDD -direction in
create_supply_port VSS -direction in
```

- **create_supply_net**

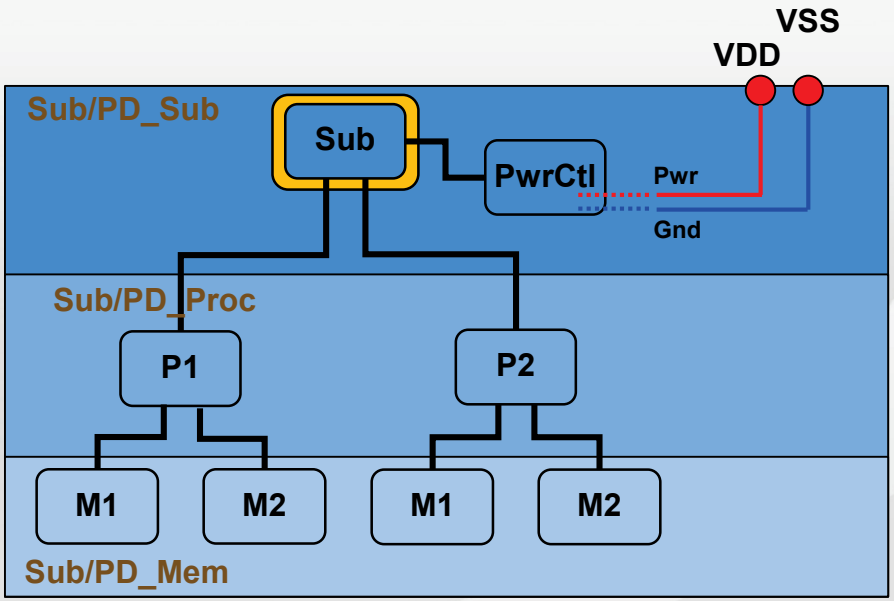
```
create_supply_net Pwr -domain PD_Sub
create_supply_net Gnd -domain PD_Sub
```

- **connect_supply_net**

```
connect_supply_net Pwr -ports {VDD}
connect_supply_net Gnd -ports {VSS}
```

- **set_domain_supply_net**

```
set_domain_supply_net PD_Sub \
   -primary_power_net Pwr \
   -primary_ground_net Gnd
```



Domain supply nets are implicitly connected to all instances contained in the power domain.

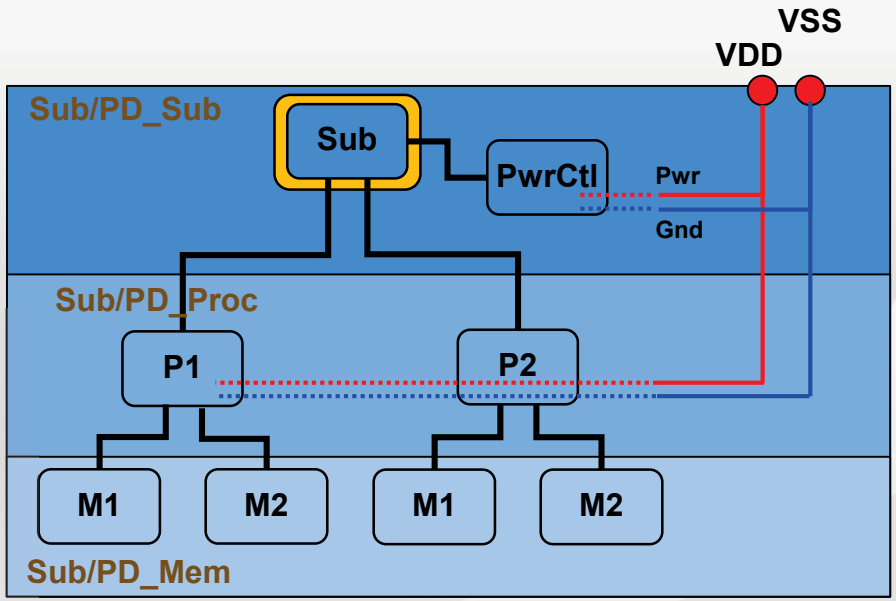
Supply Network 2 - Reuse

- **create_supply_net -reuse**

```
create_supply_net Pwr -reuse \
   -domain PD_Proc
create_supply_net Gnd -reuse \
   -domain PD_Proc
```

- **set_domain_supply_net**

```
set_domain_supply_net PD_Proc \
   -primary_power_net Pwr \
   -primary_ground_net Gnd
```



The -reuse option allows an existing supply net to be extended into and used within another power domain

Supply Network 3 – Internal Switching

- **create_supply_net**

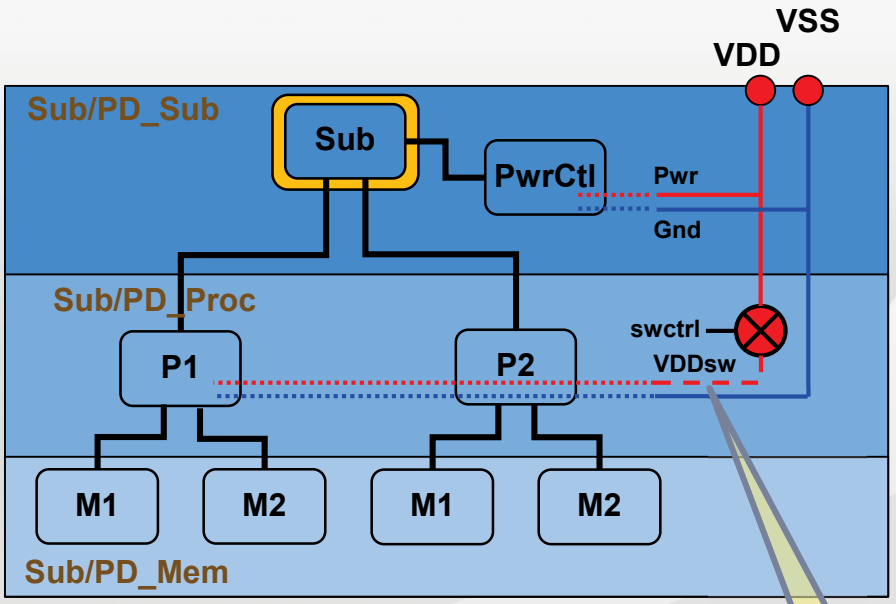
```
create_supply_net VDDsw\
    -domain PD_Proc
```

- **create_power_switch**

```
create_power_switch SW \
    -domain PD_Proc \
    -output_supply_port {swout VDDsw}\
    -input_supply_port {swin Pwr} \
    -control_port {swctrl} \
    -on_state {SWon swin swctrl} \
    -off_state {SWoff !swctrl}
```

- **set_domain_supply_net**

```
set_domain_supply_net PD_Proc \
    -primary_power_net VDDsw \
    -primary_ground_net Gnd
```



A power switch creates a switched supply. Either power or ground can be switched.

The control signal must be already present in the HDL description

Supply Network 4 – External Switching

- **create_supply_port**

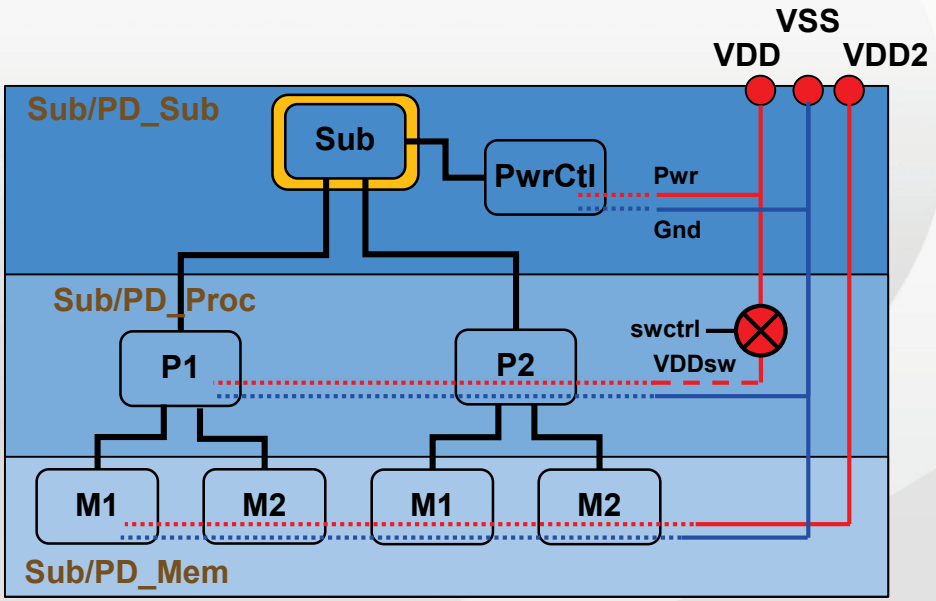
```
create_supply_port VDD2 \
    -domain PD_Mem
```

- **create_supply_net -reuse**

```
create_supply_net Gnd -reuse \
    -domain PD_Mem
```

- **set_domain_supply_net**

```
set_domain_supply_net PD_Mem \
    -primary_power_net VDD2 \
    -primary_ground_net Gnd
```

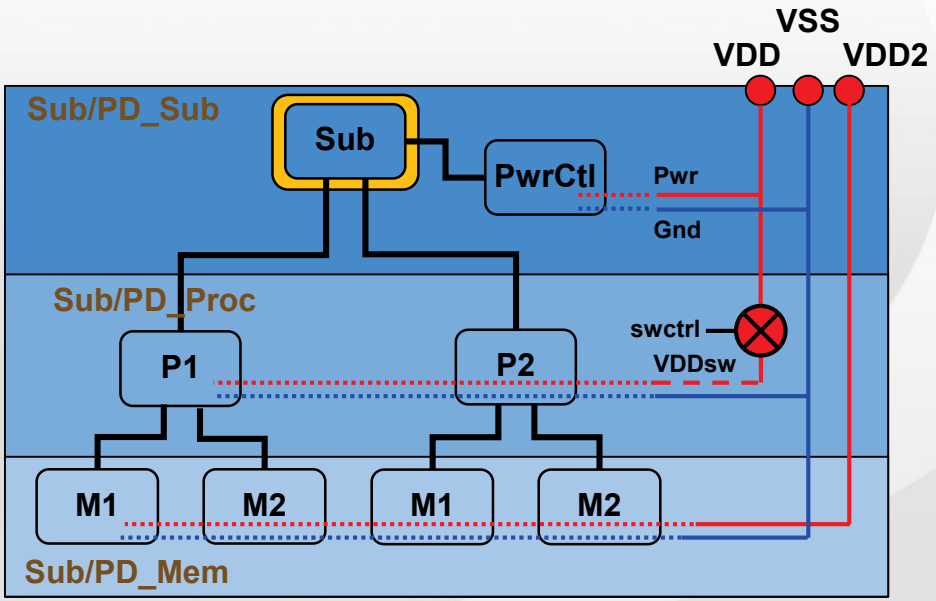


A port name can be referenced directly wherever a net name is allowed.

Supply Port States

- **add_port_state**

```
# always on
add_port_state VDD \
   -state {ON_10 1.0}

# internally switched
add_port_state SW/swout\
   -state {ON_10 1.0} \
   -state {OFF off}

# externally switched
add_port_state VDD2 \
   -state {ON_08 0.8} \
   -state {OFF off}

# common ground
add_port_state VSS \
   -state {ON_00 0.0}
```



PD_Sub	PD_Proc	PD_Mem	
VDD (port)	VDDsw (net)	VDD2 (port)	VSS (port)
ON_10	ON_10 or OFF	ON_08 or OFF	ON_00

Power State Table

- **create_pst**

```
create_pst PST1                    -supplies { VDD   VDDsw   VDD2   VSS }
```

- **add_pst_state**

```
add_pst_state Normal    -pst PST1 -state {ON_10   ON_10  ON_08  ON_00}
add_pst_state Sleep     -pst PST1 -state {ON_10   OFF    ON_08  ON_00}
add_pst_state Hibernate -pst PST1 -state {ON_10   OFF    OFF    ON_00}
```

	PD_Sub	PD_Proc	PD_Mem	
State \ Supply	VDD (port)	VDDsw (net)	VDD2 (port)	VSS (port)
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

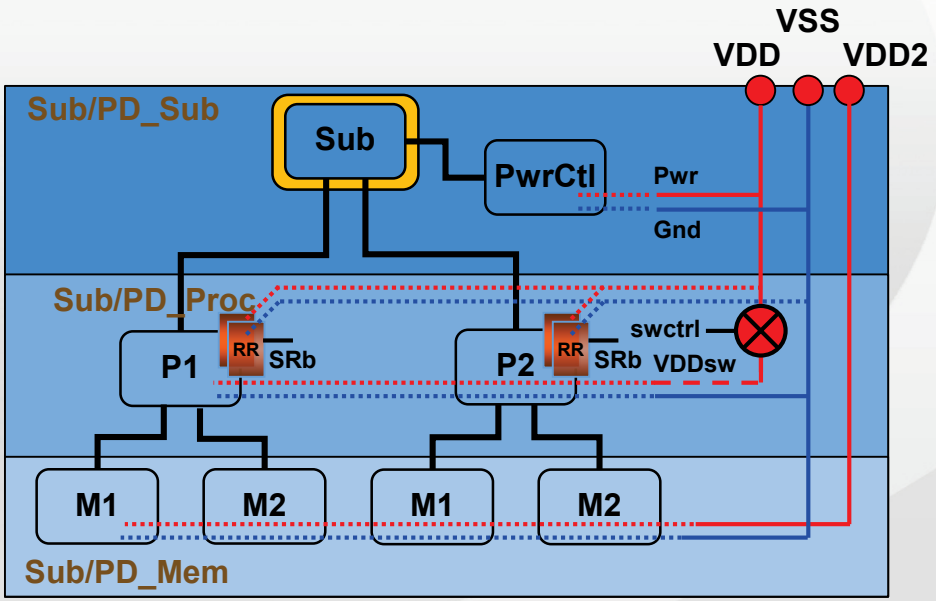
Retention Strategies

- **set_retention**

```
set_retention RET1 \
    -domain PD_Proc \
    -retention_power_net Pwr \
    -retention_ground_net Gnd
```

- **set_retention_control**

```
set_retention_control RET1 \
    -domain PD_Proc \
    -save_signal {SRb posedge} \
    -restore_signal {SRb negedge}
```



The control signal must be already present in the HDL description

	PD_Sub	PD_Proc	PD_Mem	
State \ Supply	VDD (port)	VDDsw (net)	VDD2 (port)	VSS (port)
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Isolation Strategies

- **set_isolation**

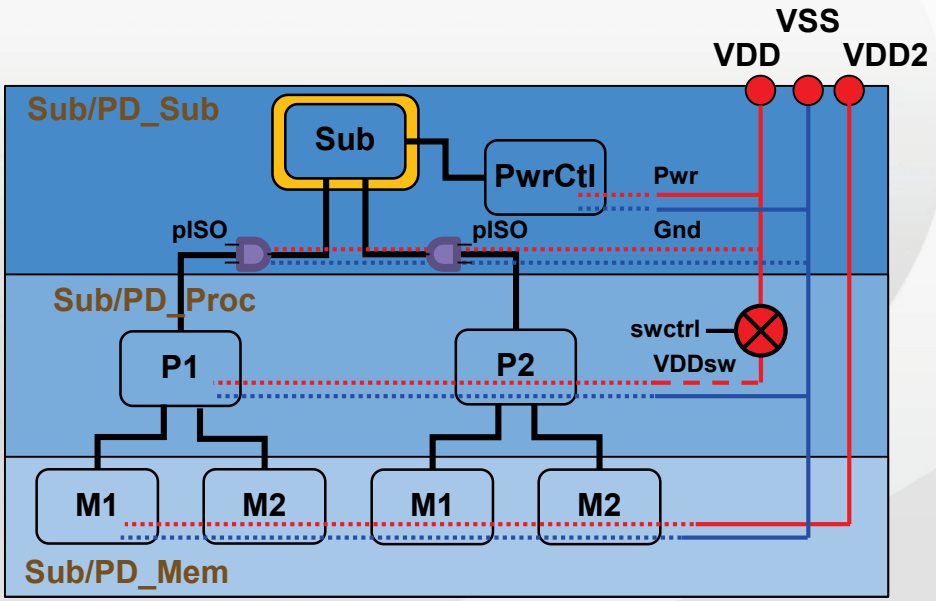
```
set_isolation ISOproc \
    -domain PD_Proc \
    -applies_to outputs \
    -clamp_value 0 \
    -isolation_power_net Pwr \
    -isolation_ground_net Gnd
```

- **set_isolation_control**

```
set_isolation_control ISOproc \
    -domain PD_Proc \
    -isolation_signal pISO \
    -isolation_sense high \
    -location parent
```

(=> Sub)

The control signal must be already present in the HDL description



	PD_Sub	PD_Proc	PD_Mem	
State \ Supply	VDD (port)	VDDsw (net)	VDD2 (port)	VSS (port)
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Isolation Strategies 2

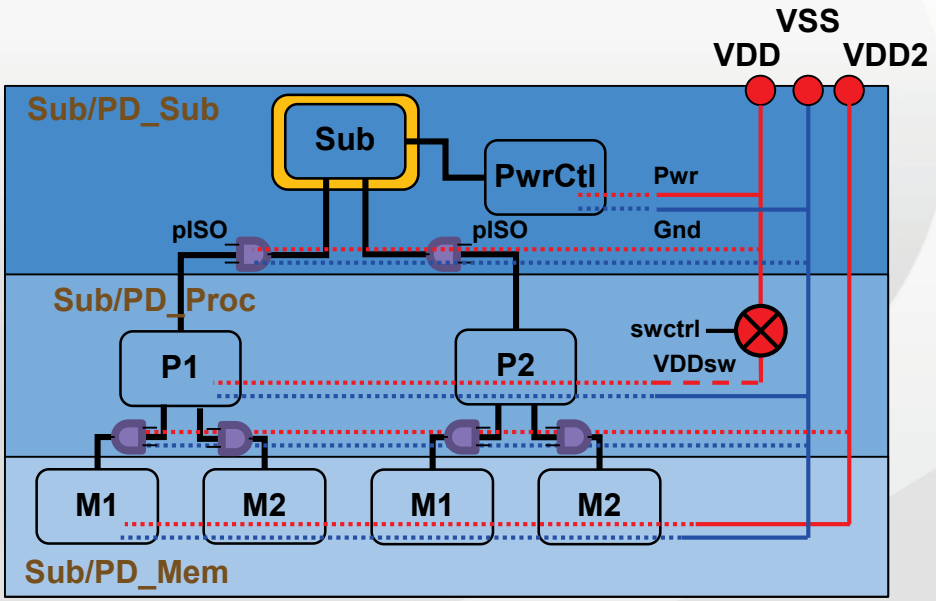
- **set_isolation**

```
set_isolation ISOmem \
    -domain PD_Mem \
    -applies_to inputs \
    -clamp_value 1 \
    -isolation_power_net VDD2 \
    -isolation_ground_net VSS
```

- **set_isolation_control**

```
set_isolation_control ISOmem \
    -domain PD_Mem \
    -isolation_signal mISO \
    -isolation_sense low \
    -location parent
```

(=> P1, P2)



State \ Supply	PD_Sub: VDD (port)	PD_Proc: VDDsw (net)	PD_Mem: VDD2 (port)	PD_Mem: VSS (port)
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Isolation Strategies 3

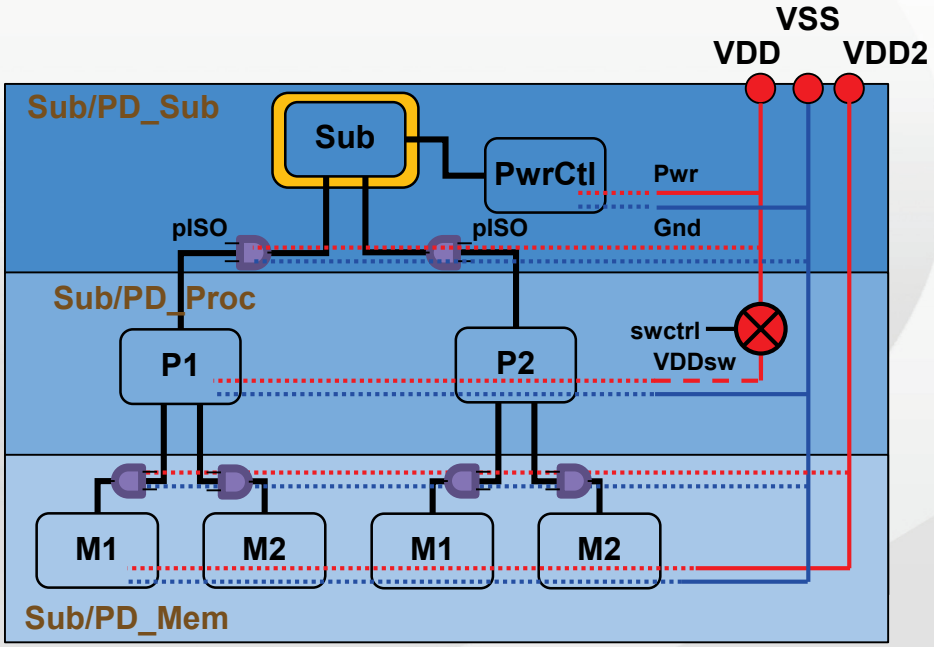
- **set_isolation**

```
set_isolation ISOmem \
    -domain PD_Mem \
    -applies_to inputs \
    -clamp_value 1 \
    -isolation_power_net VDD2 \
    -isolation_ground_net VSS
```

- **set_isolation_control**

```
set_isolation_control ISOmem \
    -domain PD_Mem \
    -isolation_signal mISO \
    -isolation_sense low \
    -location self
```

(=> M1, M2)

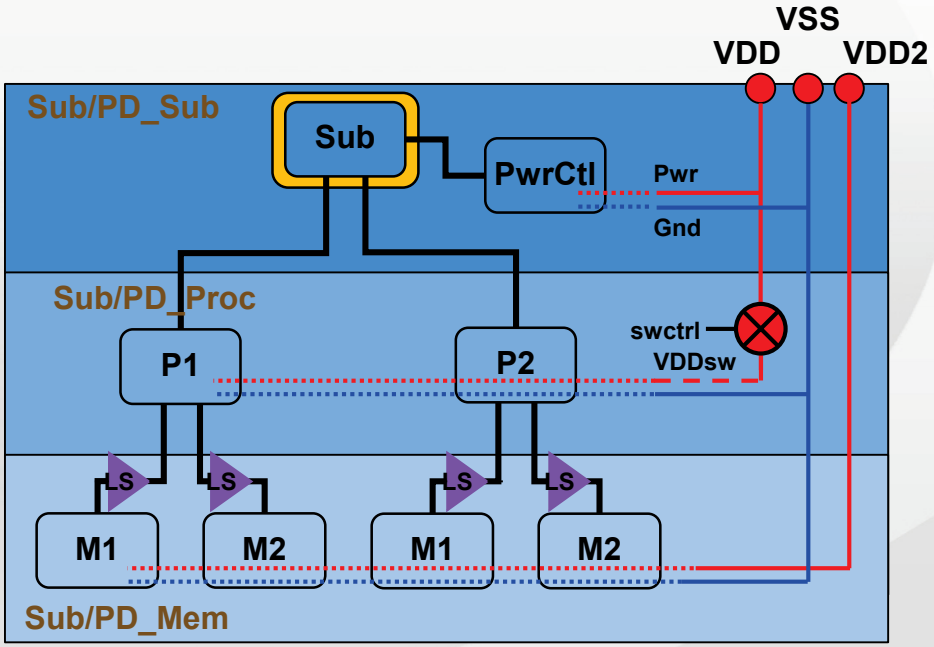


	PD_Sub	PD_Proc	PD_Mem	
State \ Supply	VDD (port)	VDDsw (net)	VDD2 (port)	VSS (port)
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Level Shifting Strategies

- set_level_shifter

```
set_level_shifter LSmem \
    -domain PD_Mem \
    -threshold 0.1 \
    -applies_to both \ (inputs and outputs)
    -rule both \ (high_to_low and low_to_high)
    -location self
                (=>M1, M2)
```



	PD_Sub	PD_Proc	PD_Mem	
State \ Supply	VDD (port)	VDDsw (net)	VDD2 (port)	VSS (port)
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Verifying and Implementing with UPF

- **Verification**
  - Testbench drives UPF-defined supply ports
  - Using supply_on(), supply_off() functions in package UPF

```
result = supply_on("Sub/VDD", 1.0);     /* VDD <= '{ON, 1.0} */
result = supply_on("Sub/VSS", 0.0);     /* VSS <= '{ON, 0.0} */
result = supply_off("Sub/VDD");         /* VDD <= '{OFF}     */
```

- **Implementation**
  - Map commands can be used to direct implementation

```
map_isolation_cell
map_level_shifter_cell
map_retention_cell
map_power_switch
```

Basic Concepts – Command Review

- **Navigation**
  - set_design_top, set_scope
- **Supply Network**
  - create_supply_port
  - create_supply_net
  - connect_supply_net
  - create_power_switch
- **Power Domains**
  - create_power_domain
  - set_domain_supply_net
- **Power States**
  - add_port_state
  - create_pst, add_pst_state
- **Strategies**
  - set_retention, set_retention_control
  - set_isolation, set_isolation_control
  - set_level_shifter
- **Code Management**
  - load_upf, save_upf
- **Miscellaneous**
  - bind_checker
  - vct commands
  - map commands

not covered

Limitations of the UPF 1.0 Subset

- **Power Supply Concept**
  - Only involves power/ground; unable to describe bias designs
  - Supply net based power state definition can only support on/off states
  - Supply network must be defined to enable verification
- **Hierarchical Composition**
  - The load_upf command allows for file partitioning only
  - Power intent at different integration levels is not well supported
- **Related Supplies for Ports**
  - No way to represent supplies if driving/receiving logic is not present
- **Power Domain Boundary**
  - Incomplete definition interferes with specifying interface logic for glue logic
  - Limits application of isolation and level shifter strategies
- **Control Logic**
  - Must exist in the HDL; cannot be inferred from UPF
  - Needs to vary with implementation decisions

More Advanced Concepts

New Capabilities in IEEE 1801-2009 UPF (2.0)

- Supply Sets
- Power Domains
- Power States
- Simstates
- Strategies
- Attributes
- Control Logic

Low Power Design Process Support

Requirements

- **Hierchical Composition**
  - To support IP-based design
- **More Abstract Modeling**
  - To simplify specifications
- **Incremental Specification**
  - To reflect distributed development
- **More Fine-Grained Control**
  - To leverage technology effectively

Solutions

- **Supply Network Abstraction**
  - Supply sets
  - Supply attributes
- **New Power State Definitions**
  - add_power_state with -simstate
  - describe_state_transition
- **Command Refinement**
  - The –update option
- **Other New Features**
  - Strategy extensions
  - Control logic support

Partitioning Power Intent

- **File Sub.upf**

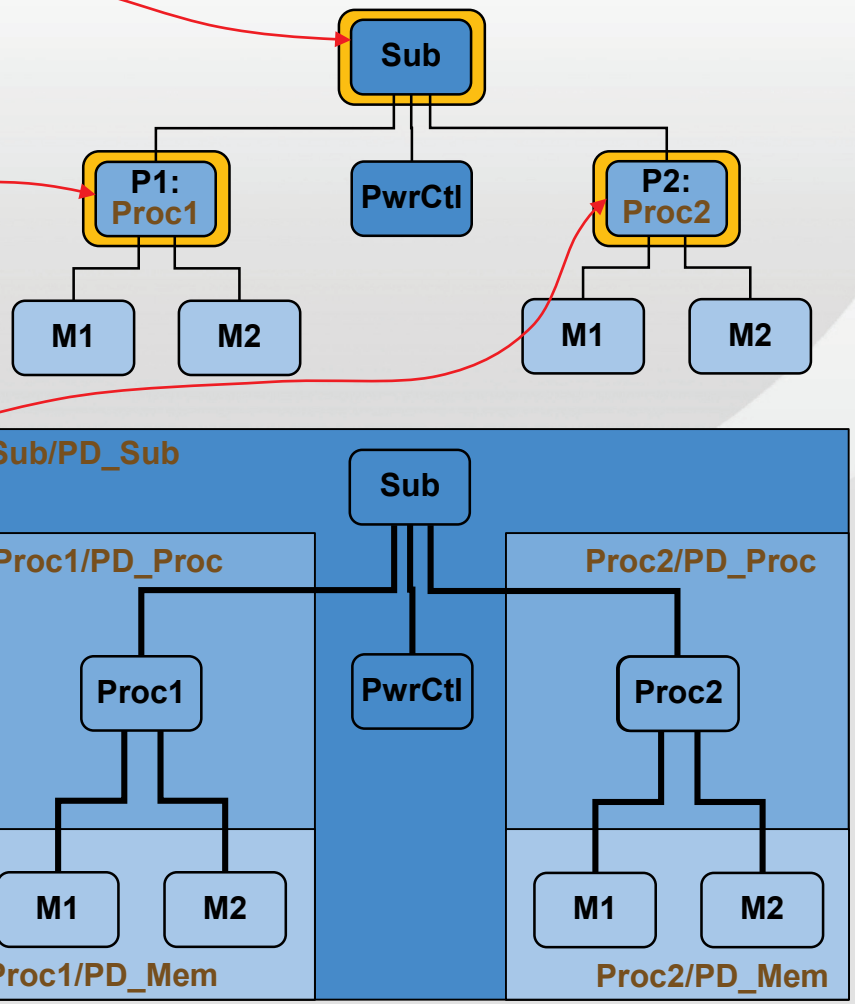
```
create_power_domain PD_Sub
    -include_scope
load_upf Proc1.upf -scope P1
load_upf Proc2.upf -scope P2
```

- **File Proc1.upf**

```
set_design_top Proc1
create_power_domain PD_Proc \
   -include_scope
create_power_domain PD_Mem \
   -elements {M1 M2}
```

- **File Proc2.upf**

```
set_design_top Proc2
create_power_domain PD_Proc \
   -include_scope
create_power_domain PD_Mem \
   -elements {M1 M2}
```

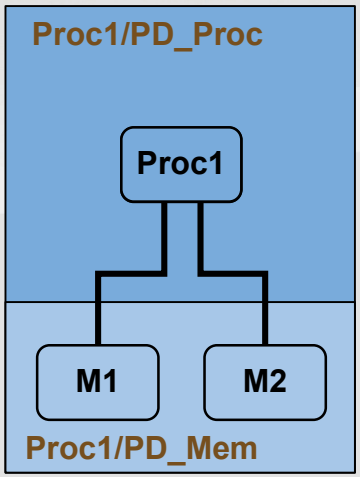


Partitioning Power Intent 2

How can we define the supply network and power states for a standalone block?

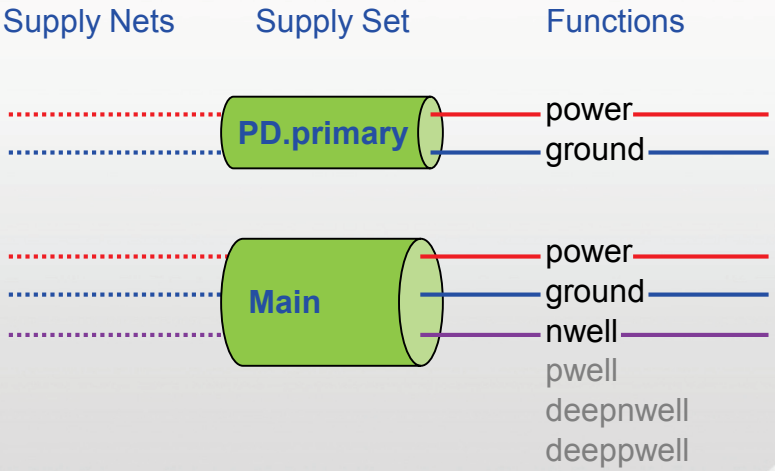
- **File Proc1.upf**

```
set_design_top Proc1
create_power_domain PD_Proc \
   -include_scope
create_power_domain PD_Mem \
   -elements {M1 M2}
```



Supply Sets

- **A group of related supply nets**
- **Functions represent nets**
  - which can be defined later
- **Electrically complete model**
  - power, ground, etc.
- **Predefined supply sets**
  - for domains
- **User-defined supply sets**
  - for domains (local)
  - standalone (global)
- **Supply set parameters**
  - for strategies




```
create_power_domain PD \
   -supply {primary} \
   -supply {backup}
```
(predefined)
(user-defined)

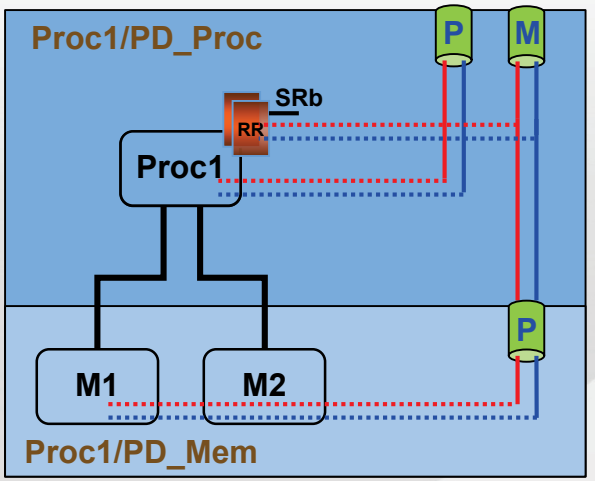
```
create_supply_set Main \
   -function {power} \
   -function {ground} \
   -function {nwell}
```
(user-defined)

```
set_isolation ISO -domain PD \
   -isolation_supply_set PD.backup
```

Block-Level Power Supply Definition

- **Domain supply sets**

```
create_power_domain PD_Proc \
   -include_scope \
   -supply {primary} \
   -supply {memory}

create_power_domain PD_Mem \
   -elements {M1 M2} \
   -supply {primary}
```



- **Supply set association**

```
associate_supply_set PD_Proc.memory \
   -handle PD_Mem.primary
```

- **Supply set connection**

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

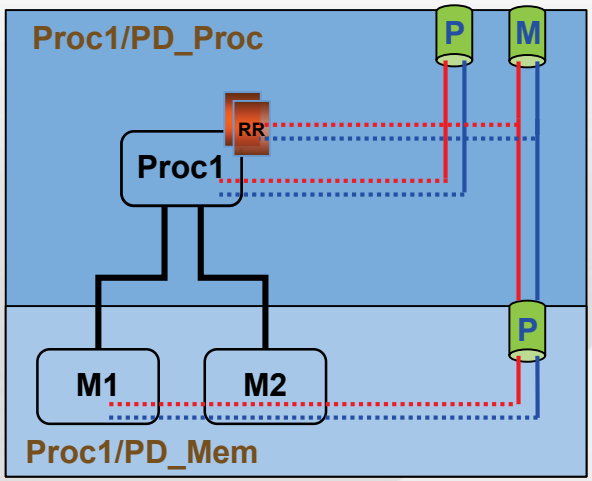
- **Supply set for retention etc.**

```
set_retention RET1 -domain PD_Proc \
   -save_signal {SRb posedge} \
   -restore_signal {SRb negedge} \
   -retention_supply_set PD_Proc.memory
```

Power States of Supply Sets

- add_power_state <supply set>

```
add_power_state PD_Proc.primary \
  -state OFF { \
    -supply_expr {power == OFF} } \
  -state ON_10 { \
    -supply_expr {power == {FULL_ON 1.0} && \
                  ground == {FULL_ON 0.0} }

add_power_state PD_Proc.memory \
  -state OFF { \
    -supply_expr {power == OFF} } \
  -state ON_08 { \
    -supply_expr {power == {FULL_ON 0.8} && \
                  ground == {FULL_ON 0.0} }

add_power_state PD_Mem.primary \
  -state OFF { \
    -supply_expr {power == OFF} } \
  -state ON_08 { \
    -supply_expr {power == {FULL_ON 0.8} && \
                  ground == {FULL_ON 0.0} }
```

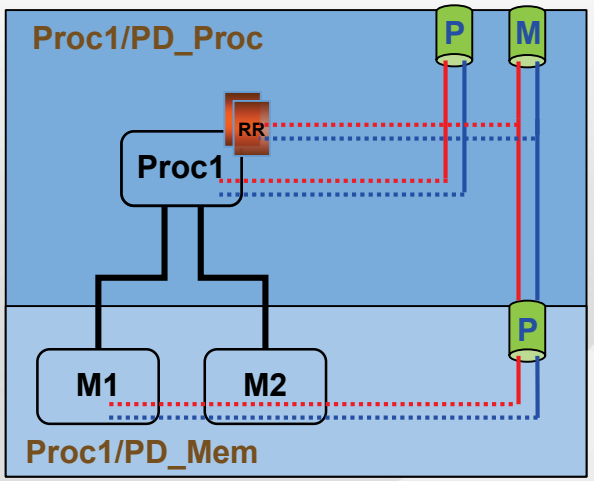


PD_Proc		PD_Mem
primary (supplyset)	memory (supplyset)	primary (supplyset)
ON_10 or OFF	ON_08 or OFF	ON_08 or OFF

Simstates

- **Predefined Simstates**
  - NORMAL
  - CORRUPT_STATE_ON_CHANGE
  - CORRUPT_STATE_ON_ACTIVITY
  - CORRUPT_ON_ACTIVITY
  - CORRUPT
- **add_power_state -update**

```
add_power_state PD_Proc.primary -update \
  -state OFF { -simstate CORRUPT } \
  -state ON_10 { -simstate NORMAL }

add_power_state PD_Proc.memory -update \
  -state OFF { -simstate CORRUPT } \
  -state ON_08 { -simstate NORMAL }

add_power_state PD_Mem.primary -update \
  -state OFF { -simstate CORRUPT } \
  -state ON_08 { -simstate NORMAL }
```

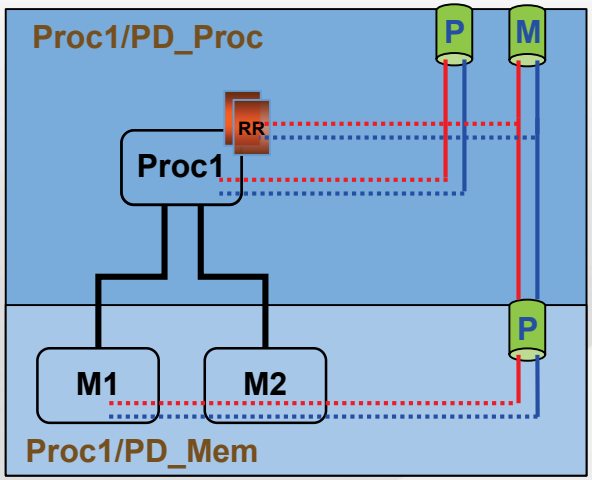


PD_Proc		PD_Mem
primary (supplyset)	memory (supplyset)	primary (supplyset)
ON_10 or OFF	ON_08 or OFF	ON_08 or OFF

Power States of Power Domains

- add_power_state <domain>

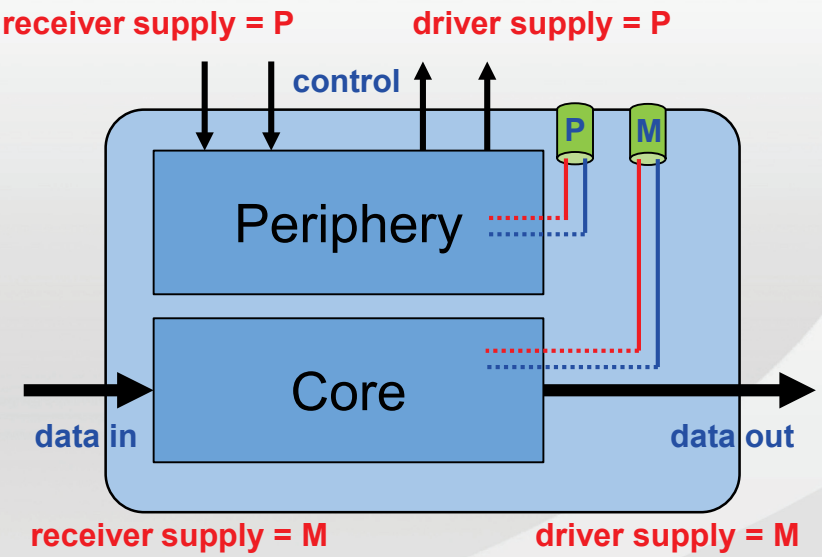
```
add_power_state PD_Mem \
  -state UP {-logic_expr {primary == ON_08} } \
  -state DOWN {-logic_expr {primary == OFF} }


add_power_state PD_Proc \
  -state Normal { \
    -logic_expr {primary == ON_10 && \
                 memory == ON_08 && \
                 PD_Mem == UP} } \
 -state Sleep { \
    -logic_expr {primary == OFF && \
                 memory == ON_08 && \
                 PD_Mem == UP} } \
 -state Hibernate { \
    -logic_expr {primary == OFF && \
                 memory == OFF && \
                 PD_Mem == DOWN} }
```



PD_PROC	primary	memory	PD_MEM
Normal	ON_10	ON_08	UP
Sleep	OFF	ON_08	UP
Hibernate	OFF	OFF	DOWN

Supply Related Attributes

- **Driver supply attribute**
  - Supply set powering driving logic
- **Receiver supply attribute**
  - Supply set powering receiving logic
- **Behavioral models**
  - Supplies may not be evident
  - set_port_attributes -driver_supply …
  - set_port_attributes -receiver_supply …
- **Hard macros (e.g., Memory)**
  - Attributes of cell pins:
    - PG type attributes
    - Related supply attributes
  - Imply anonymous supply sets
  - Can be defined in UPF or Liberty



If the memory cell has separate supplies for peripheral logic and memory core, different ports may have different driver supplies or receiver supplies.

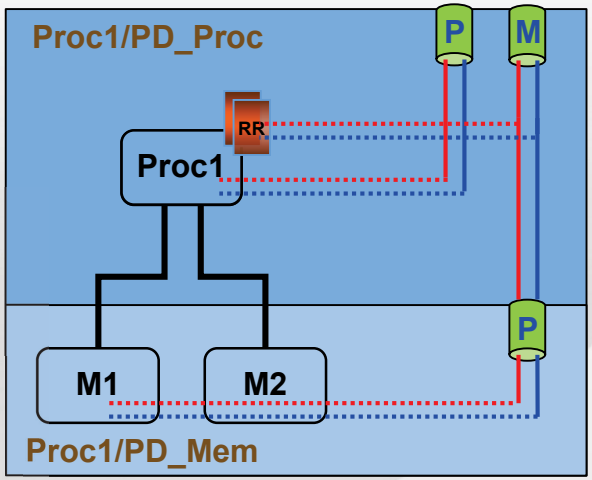
Supply Sets and Strategies

- **set_isolation -source / -sink**

```
set_isolation ISO1 -domain PD_Mem \
   -applies_to inputs \
   -source PD_Proc.primary …
set_isolation ISO1 -domain PD_Mem \
   -applies_to outputs \
   -sink PD_Proc.primary …
```

- **set_isolation -diff_supply_only**

```
set_isolation ISO2 -domain PD_Mem \
   -applies_to both \
   -diff_supply_only …
```



Isolates any path from PD_Proc to PD_Mem

Isolates any path from PD_Mem to PD_Proc

Isolates any port of PD_Mem except those that connect back to PD_Mem

Control Logic

- **create_logic_port**

```
create_logic_port pISO -direction in
```

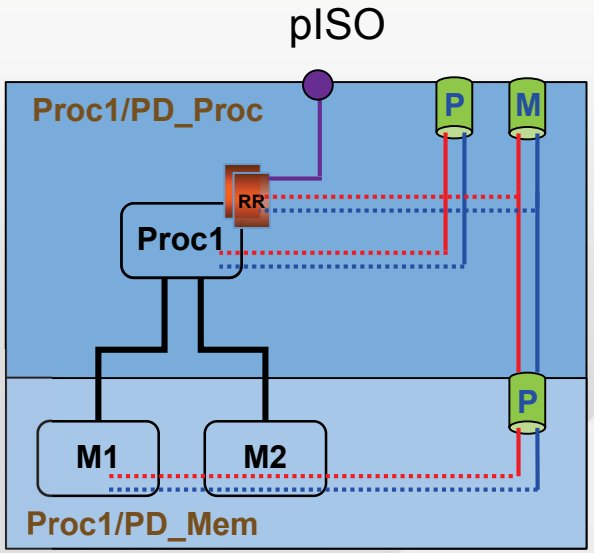
- **create_logic_net**

```
create_logic_net pISOnet
```

- **connect_logic_net**

```
connect_logic_net pISOnet -ports pISO
```

```
set_isolation ISOproc \
   -domain PD_Proc\
   -applies_to outputs \
   -clamp_value 0 \
   -isolation_signal pISO \
   -isolation_sense high \
   -location parent
```

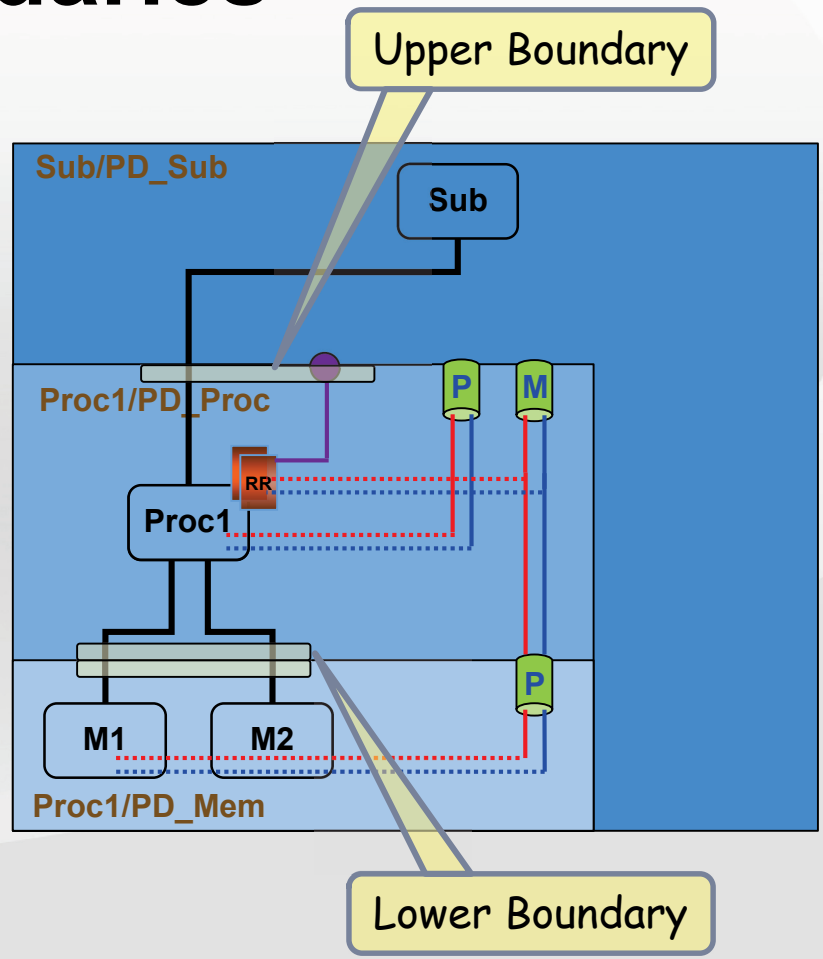


Create the necessary isolation control port and net within the UPF file

Reference the control signal as before, but within set_isolation (no set_isolation_control)

Power Domain Boundaries

- **UPF 1.0**
  - Boundary is defined by port declarations
    - "Upper boundary" only
- **UPF 2.0**
  - Boundary is defined by domain crossings
    - "Upper boundary"
      - Ports connected to nets in a 'higher' domain (lowconn of those ports)
    - "Lower boundary"
      - Nets connected to ports in a 'lower' domain (highconn of those ports)



Incremental Command Update

UPF 1.0 Method

- **set_retention**

```
set_retention RET1 \
   -domain PD_Proc \
   -retention_power_net Pwr \
   -retention_ground_net Gnd
```

- **set_retention_control**

```
set_retention_control RET1 \
   -domain PD_Proc \
   -save_signal {SRb posedge} \
   -restore_signal {SRb negedge}
```

UPF 2.0 Method

- **set_retention**

```
set_retention RET1 \
   -domain PD_Proc \
   -save_signal {SRb posedge} \
   -restore_signal {SRb negedge}
```

- **set_retention -update**

```
set_retention RET1 -update \
   -domain PD_Proc \
   -retention_supply_set PD_Proc.memory \
   -use_retention_as_primary

set_retention RET1 -update \
   -domain PD_Proc \
   -instance {reg1 reg2 reg3}
```

Incremental Development

Specification

- Define power domains and their supply sets
  - including power states and simstates
- Connect PD supplies as needed
  - to PD strategies
  - to PD instances
- Verify

Implementation

- Define supply ports and switches
- Group supply nets into sets
  - with a supply net for each function
- Associate a supply set with each PD supply
- Implement

IEEE 1801-2009 UPF - Summary

- **Supply Sets**
  - create_power_domain –supply
  - create_supply_set
  - associate_supply_set
  - connect_supply_set
- **Power States**
  - add_power_state, describe_state_transition
- **Simstates**
  - add_power_state -supply -state -simstate
- **Strategies**
  - filters and supply sets/domains
- **Attributes**
  - UPF_* attributes, set_port_attributes, set_design_attributes, Liberty attributes
- **Power Domains**
  - power domain lower boundary
  - create_composite_domain
- **Control Logic**
  - create_logic_port, create_logic_net
  - connect_logic_net
- **Code Management**
  - load_upf_protected
  - find_objects
- **Other**
  - use_interface_cell
  - load_simstate_behavior
  - set_retention_elements

not covered

Break

Power Intent Specification Methodology with IEEE-1801

Qi Wang

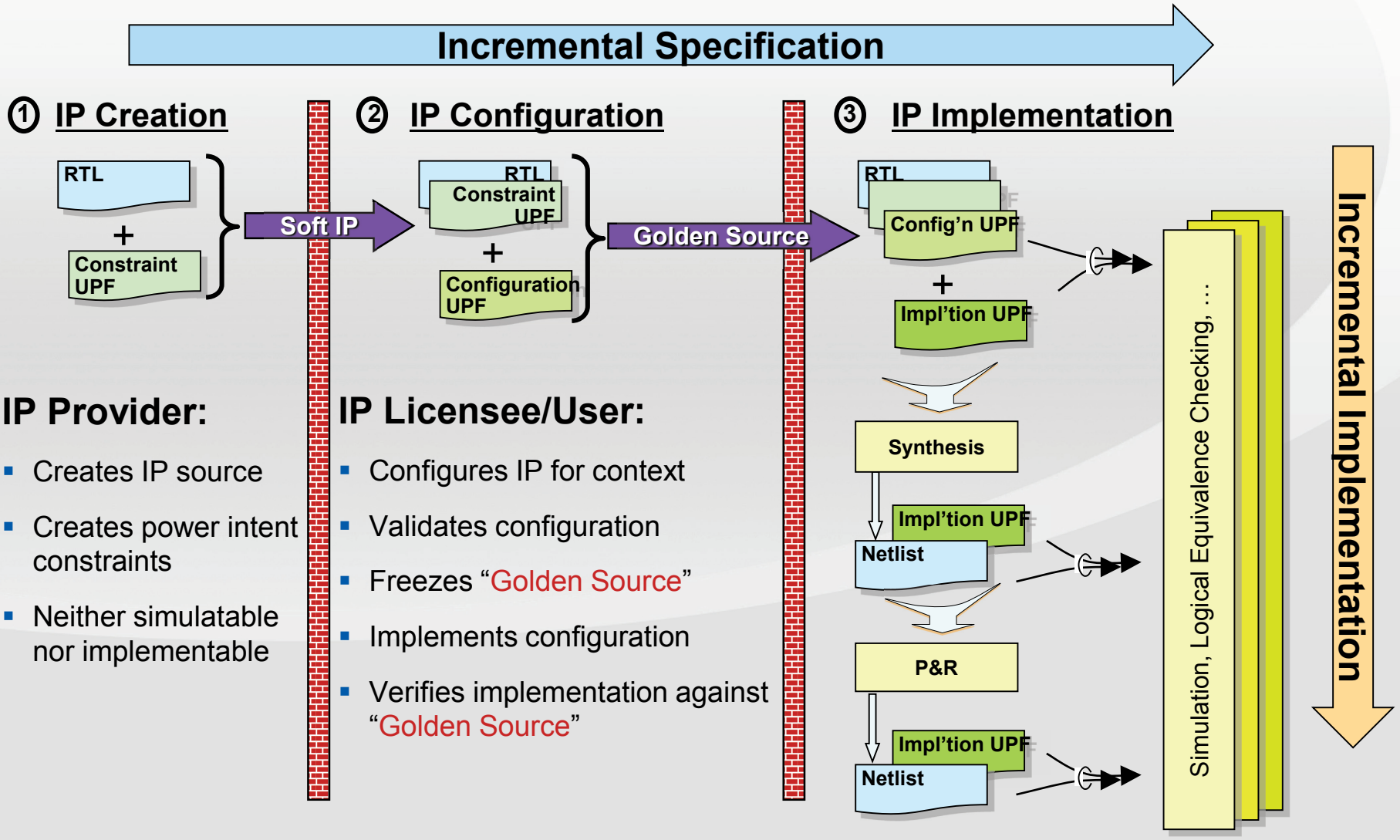
Technical Marketing Group Director

Cadence Design Systems, Inc.

Agenda

- Successive refinement
- Hierarchical power intent specification

Successive Refinement



IP Provider:

- Creates IP source
- Creates power intent constraints
- Neither simulatable nor implementable

IP Licensee/User:

- Configures IP for context
- Validates configuration
- Freezes "Golden Source"
- Implements configuration
- Verifies implementation against "Golden Source"

Example of Incremental Development
Using Multiple Commands

1. **UPF Constraints**
   - IP provider needs to identify what clamp value to use when isolation is required:

```
set_port_attributes -clamp_value 0 -elements I1 -applies_to outputs
```

2. **UPF Configuration**
   - Power intent to enable simulation and implementation:

```
set_isolation -update my_iso -domain my_pd -clamp_value 0\
        -isolation_signal CLAMP -isolation_sense high \
        -location parent
```

3. **UPF Implementation**
   - Finally the UPF is updated with implementation details:

```
use_interface_cell my_iso -domain my_pd \
    -lib_cells {iso1}
```

Example of Incremental Development
Using -update

1. **RTL abstracts**
   - Shared global supply sets

```
create_supply_set SS1
add_power_state SS1 -state {idle -simstate CORRUPT_ON_ACTIVITY \
        -logic_expr { pcm/biased && !pcm/on} }
```

2. **Implementation details**
   - Update power network when required during physical implementation

```
create_supply_set SS1 -update \
    -function {power VDD} -function {ground VSS}
add_power_state SS1 -update -state {idle \
    -supply_expr {{power == '{FULL_ON,1.0}}&&{ground == '{FULL_ON,0.2}}}}
```

Hierarchical Power Intent Specification

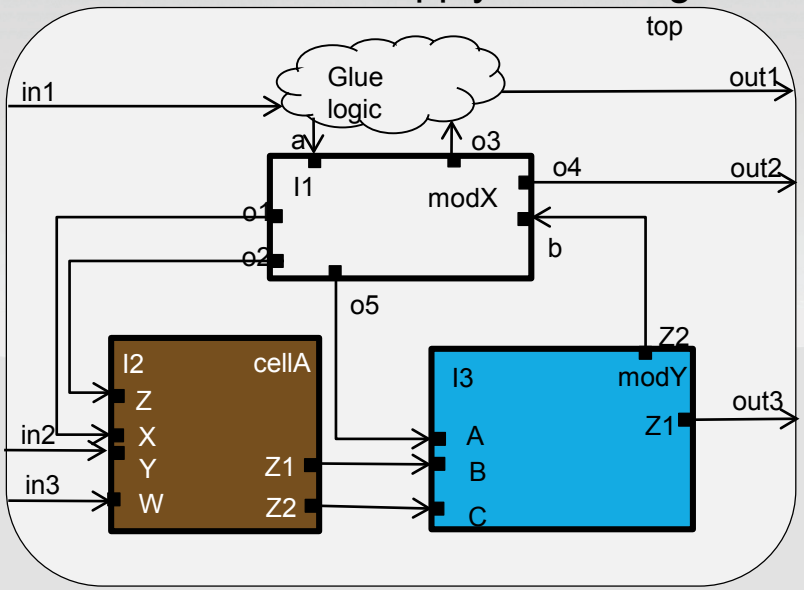
- **Soft IP**
  - An HDL module to be implemented logically or physically.

- **Hard IP**
  - A block that has been completely implemented and can be used as it is in other blocks. This may be modeled by an HDL module for verification or as a library cell for implementation.

Hierarchical Power Intent Specification

Soft IP Modeling

- **Coding guidelines**
  - One file per soft IP
    - Include all power intents of the IP in one file
    - Define interface supply sets & logic



Soft IP UPF: top.upf

```
# define interface logic ports
create_logic_port iso_enable -direction in
…

#define power domains and interface supply sets
create_power_domain PDTop \
  -include_elements \
  -supply { SSH1 } -supply { SSH2 }
…

#define power states for supply sets
add_power_state PDTop.SSH1 …

#define system power states
add_power_state PDTop …

#define strategies
set_isolation iso1 -domain PDTop \
  -isolation_signal iso_enable \
  -isolation_supply_set PDTop.SSH1
```

Hierarchical Power Intent Specification

Soft IP Integration

- **Coding guidelines**
  - Use "**load_upf** <file> **-scope** <inst_name>" to load in block UPF
  - Use **associate_supply_set** to connect the SoC level supply sets with the interface supply sets of the soft IP
  - Use **connect_logic_net** to connect any logic ports at the interface

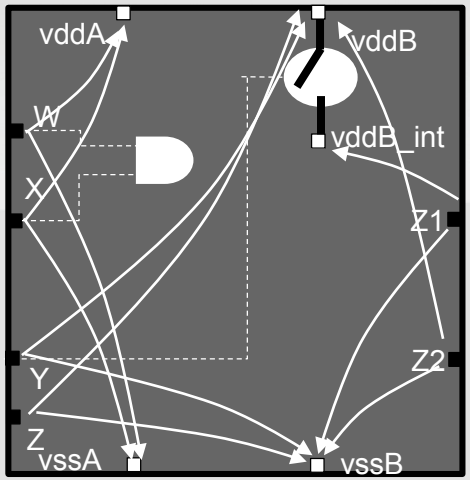
Soft IP UPF: soc.upf

```
# load and instantiate the soft IP UPF
load_upf top.upf -scope I1/I2

#define the connection for IP level interface supply
associate_supply_set SS1 \
  -handle {I1/I2/PDTop.SSH1}
associate_supply_set SS2 \
  -handle {I1/I2/PDTop.SSH2}

# define the connection for IP level interface logic port
connect_logic_net pcm/iso \
  -port I1/I2/iso_enable

…
```

Hierarchical Power Intent Specification

Hard IP Modeling

- **Use Liberty**
  - Good for implementation
  - Limitations
    - Cannot define supply sets and power states
- **Use 1801 coding guidelines**
  - One file per soft IP
    - Include all power intents of the IP in one file
    - Define interface supply sets



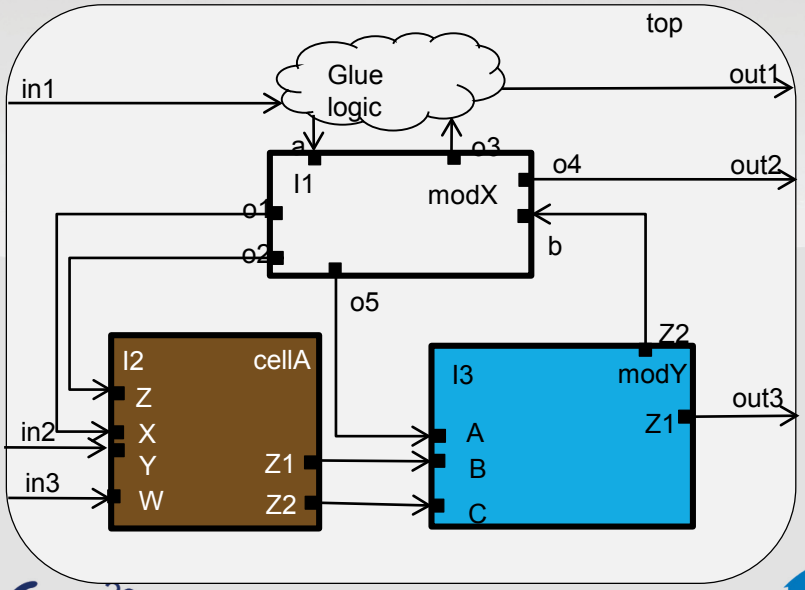
Hard IP UPF: cell.upf

```
#define power domains and interface supply sets
create_power_domain PD -elements {.}\
  -supply { SSAH } -supply { SSBH }
…
#associate the interface supplies to boundary supply
#ports or internally generated supplies
create_supply_set PD.SHAH \
  -function { power vddA } \
  -function { ground vssA } -update
…
#define data port and interface supply set handle
#associations
set_port_attributes -ports {W X} \
  -receiver_supply PD.SSAH
set_port_attributes -ports {Y Z} \
  -receiver_supply PD.SSBH

#define power states for supply sets and system
add_power_state PD.SSAH …
add_power_state PD …\
```

Hierarchical Power Intent Specification

Hard IP Integration

- **Coding guidelines**
  - Use "**`load_upf`** `<file>` **`-scope`** `<inst_name>`" to load in block UPF
  - Use **`associate_supply_set`** to connect the SoC level supply sets with the interface supply sets of the soft IP



Soft IP UPF: top.upf

```
# load and instantiate the hard IP UPF
load_upf cell.upf -scope I2

#define the connection for IP level interface supply
associate_supply_set PDTop.SSH1 \
  -handle {I2/PD.SSAH}
associate_supply_set PDTop.SSH2 \
  -handle {I2/PD.SSBH}
...
```

Using UPF for System Design

Sushma Honnavara-Prasad

Sr Staff Engineer

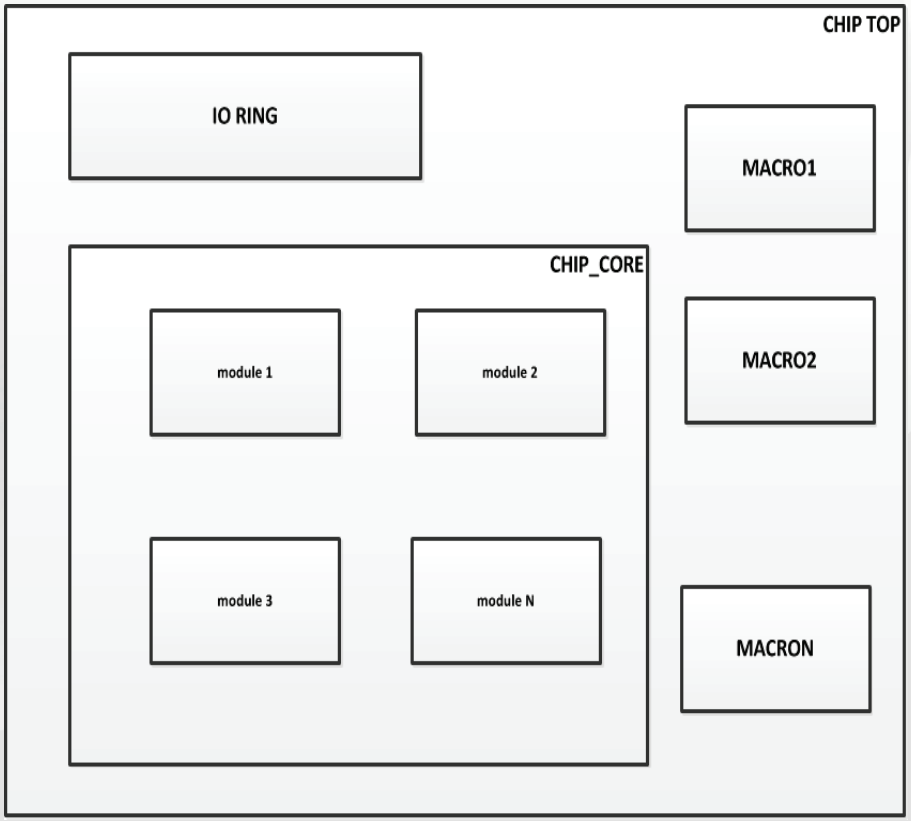
Broadcom

Agenda

- **Introduction**
- **Block implementation examples**
  - Hard IP with 2 domains
  - IO pads
- **SoC Integration**
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

Introduction

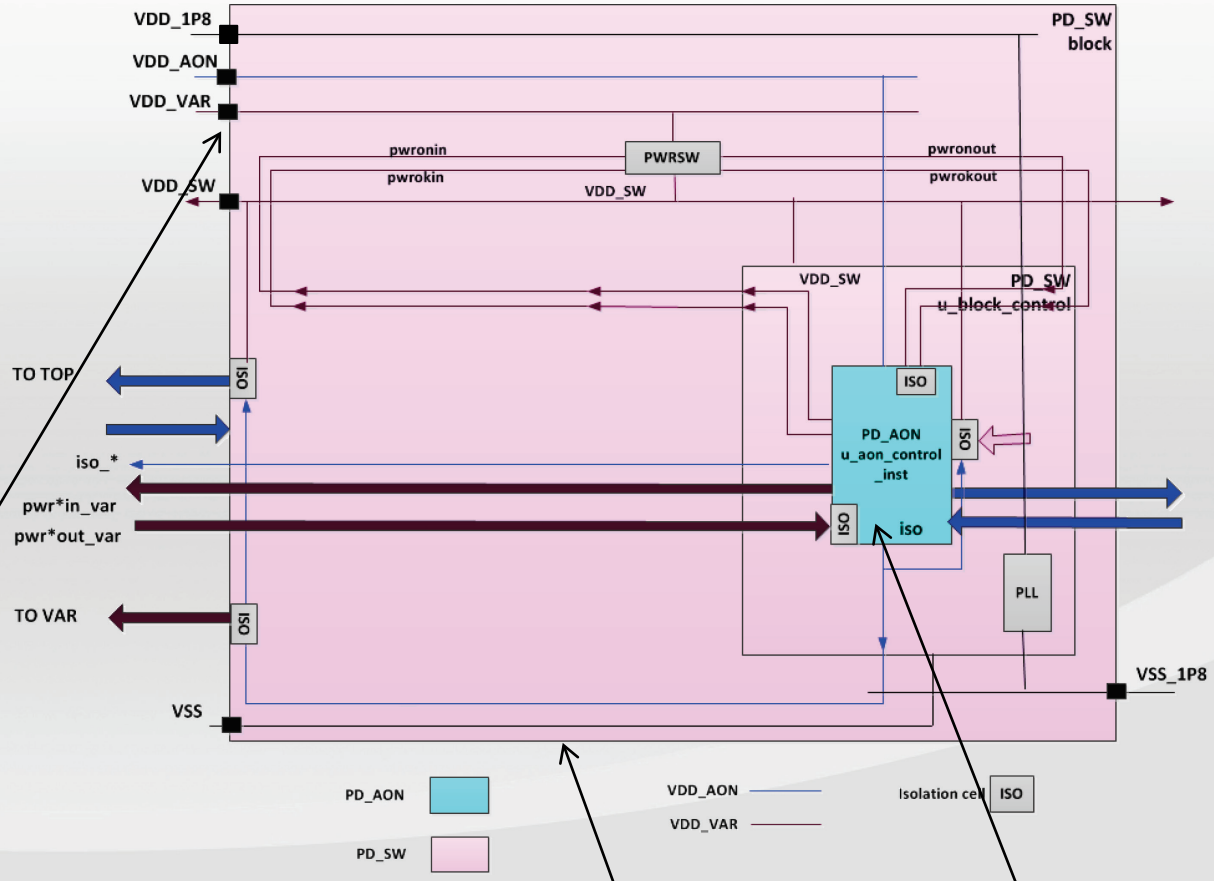
- **A typical SoC contains:**
  - Hard IP (fully implemented macros)
  - Soft IP (HDL integrated into top level)
  - Analog macros
  - IO pads
- **Challenges involved:**
  - Number of power supplies and their connections
  - Number of system power states
  - Modularizing the top level UPF
  - Specification of top level iso/ls requirements due to multiple domains



Agenda

- Introduction
- **Block implementation examples**
  - Hard IP with 2 domains
  - IO pads
- SoC Integration
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

Domain Definition and Supply Ports



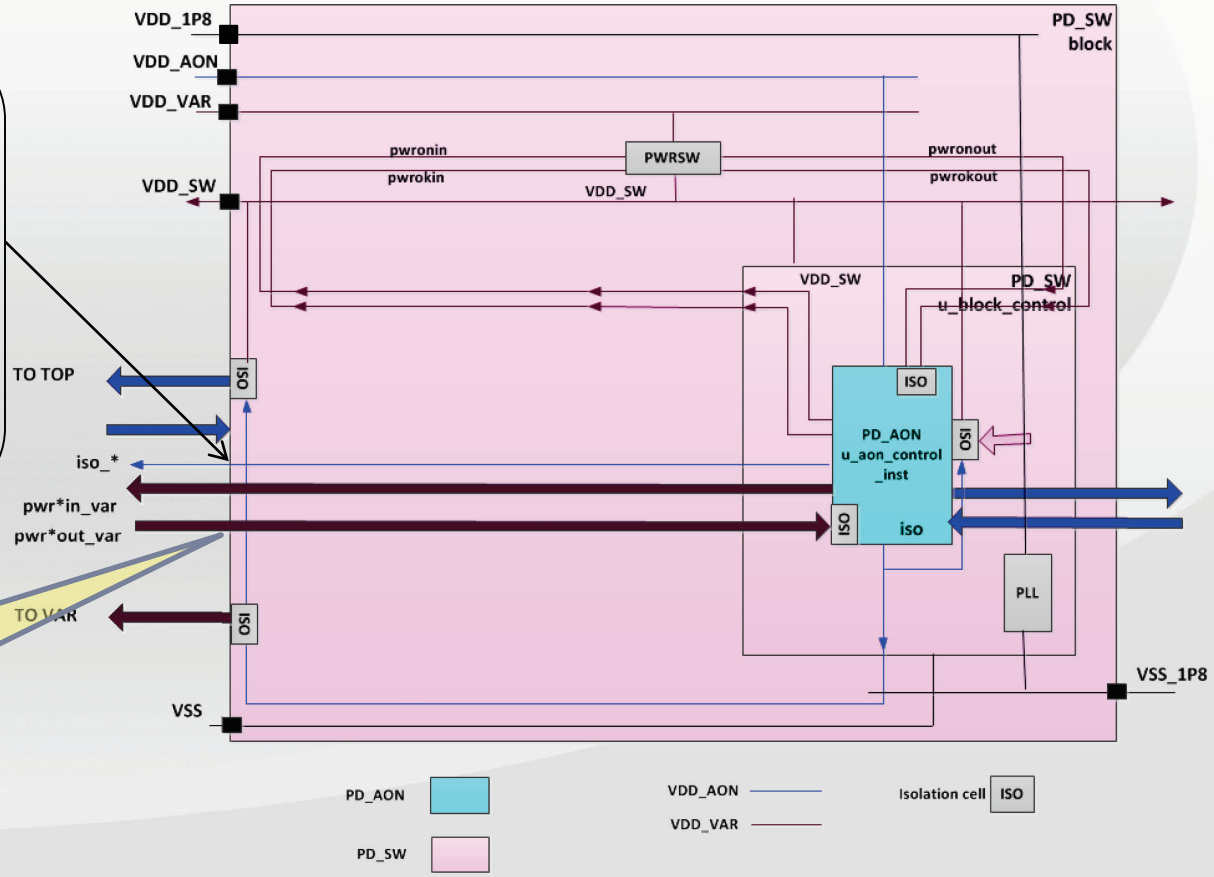

```
create_supply_port VDD_1P8
create_supply_port VDD_AON
...
create_supply_port VSS
..
create_supply_port VDD_SW -direction out
```

```
create_power_domain  PD_SW -include_scope
```

```
create_power_domain  PD_AON -elements { \
u_aon_control_inst \
}
```

Supply Sets and Port Attributes

```
set_port_attributes \
  -applies_to outputs \
  -receiver_supply varss
set_port_attributes \
  -applies_to inputs \
  -driver_supply   varss
set_port_attributes \
  -driver_supply   aonss \
  -ports [ list iso \
    [find_objects . \
     -pattern *_aon* \
     -object_type port] ]
```



Port attributes are required to be able to infer isolation/ level shifters and aon buffering

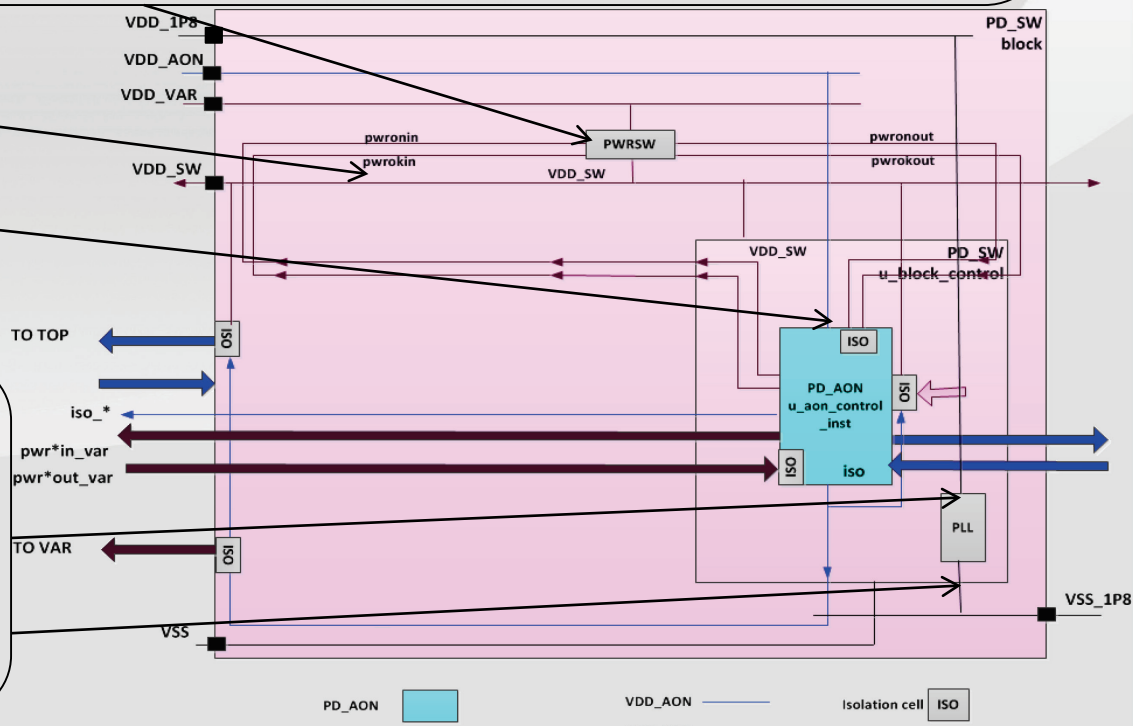
```
create_supply_set aonss -function {power VDD_AON} -function {ground VSS} -function {nwell VDD_AON}
create_supply_set varss -function {power VDD_VAR} -function {ground VSS} -function {nwell VDD_VAR}
create_supply_set swss  -function {power VDD_SW}  -function {ground VSS} -function {nwell VDD_SW}
create_supply_set 1p8ss -function {power VDD_1P8} -function {ground VSS_1P8 }
```

Supply Connections

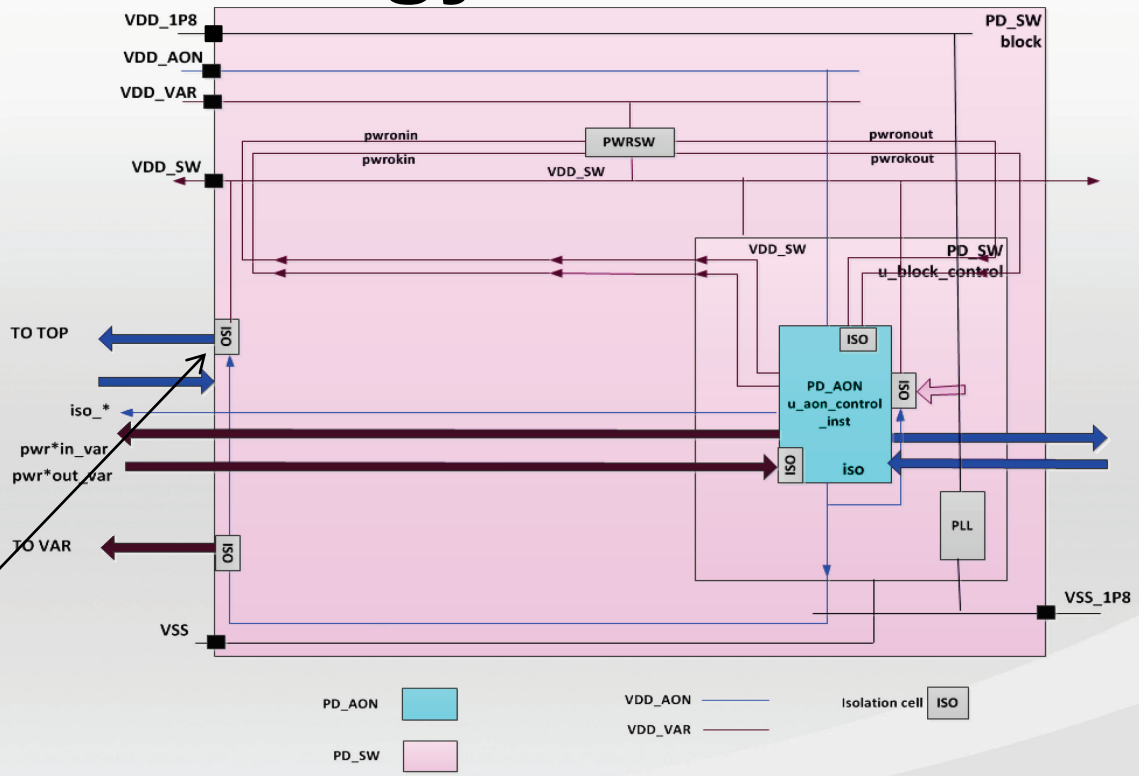
```
create_power_switch PSW_SW -domain PD_SW \
  -output_supply_port { VDD   VDD_SW } \
  -input_supply_port  { VDDB VDD_VAR } \
  -control_port       { pwronin  u_block_control/u_aon_control_inst/pwronin[0] } \
  -control_port       { pwrokin  u_block_control/u_aon_control_inst/pwrokin[0] } \
  -ack_port           { pwronout u_block_control/u_aon_control_inst/pwronout[0] {pwronin}} \
  -ack_port           { pwrokout u_block_control/u_aon_control_inst/pwrokout[0] {pwrokin}} \
  -on_state           { full_on  VDDB {pwronin & pwrokin}} \
  -off_state          { full_off {!pwronin & !pwrokin}}
```

```
associate_supply_set swss \
  -handle PD_SW.primary

associate_supply_set aonss \
  -handle PD_AON.primary
```

```
set pll_inst [find_objects .\
  -pattern *u_pll* -object_type inst \
  -leaf_only -transitive]

connect_supply_net 1p8ss.power  \
  -ports "$pll_inst[0]/AVDD1P8"

connect_supply_net 1p8ss.power  \
  -ports "$pll_inst[0]/AVSS"
```

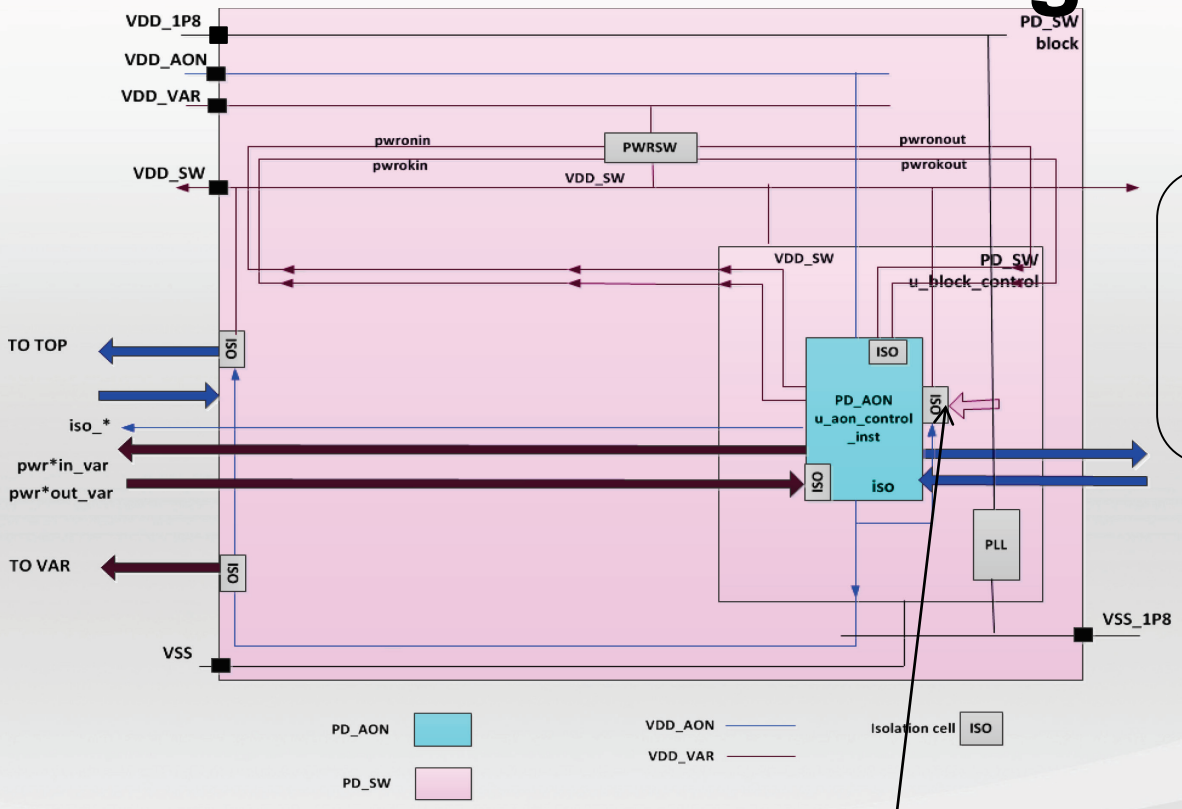


Isolation Strategy




```
set_isolation sw_iso_c0 -domain PD_SW \
  -isolation_supply_set aonss -clamp_value 0 -applies_to outputs -diff_supply_only

set_isolation sw_iso_c0 -domain PD_SW -update \
  -isolation_sense high -location self -isolation_signal iso

use_interface_cell source_isolow -strategy sw_iso_c0 \
  -domain PD_SW  -lib_cells {CELL_ISOLOWX4}
```

Isolation/Level Shifting Strategies




```
set_level_shifter no_ls_sw_to_aon \
  -domain PD_SW -sink aonss \
  -threshold 0.2

set_level_shifter no_ls_aon_to_sw \
  -domain PD_AON -sink swss \
  -threshold 0.2
```

```
set_isolation sw_to_aon_c0 -domain PD_AON \
  -isolation_supply_set swss -clamp_value 0 -applies_to inputs -diff_supply_only

set_isolation sw_to_aon_c0 -domain PD_AON -update \
  -isolation_sense high -location parent -isolation_signal iso1

use_interface_cell source_isolo1 -strategy sw_to_aon_0 -domain PD_AON \
  -lib_cells {CELL_ISOLOWSX4}
```

Supply States

Supply	nom			turbo			offmode		
	VDD	VSS	Nwell	VDD	VSS	Nwell	VDD	VSS	Nwell
aonss	0.8	0	0.8						
varss	0.8	0	0.8	0.9	0	0.9	off	0	off
swss	0.8	0	0.8	0.9	0	0.9	off	0	off
1p8ss	1.8	0							

Note: Power state for swss is similar to varss. Not listed here for brevity.

```
add_power_state aonss -supply  \
  -state { nom     -supply_expr { (power == {FULL_ON 0.8}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.8}) } \
  -state { not_on  -supply_expr { (power != {FULL_ON 0.8}) } -illegal }

add_power_state varss -supply  \
  -state { nom     -supply_expr { (power == {FULL_ON 0.81}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.81}) } \
  -state { turbo   -supply_expr { (power == {FULL_ON 0.90}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.90}) } \
  -state { offmode -supply_expr { (power == {OFF}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {OFF})  -simstate CORRUPT}

add_power_state 1p8ss -supply  \
  -state { nom     -supply_expr { (power == {FULL_ON 1.8}) && (ground == {FULL_ON 0}) } \
  -state { not_on  -supply_expr { (power != {FULL_on 1.8}) } -illegal }
```

Power States

State	aonss	varss	swss	1p8ss
nom	nom	nom	nom	nom
turbo	nom	turbo	turbo	nom
sw1off	nom	nom	offmode	nom
sw2off	nom	turbo	offmode	nom
sw3off	nom	offmode	offmode	nom

```
add_power_state PD_SW -domain  \
  -state { turbo   -logic_expr  { (varss  == turbo)   && (swss  == turbo)}   } \
  -state { nom     -logic_expr  { (varss  == nom)     && (swss  == nom)}     } \
  -state { sw1off  -logic_expr  { (varss  == turbo)   && (swss  == offmode)} } \
  -state { sw2off  -logic_expr  { (varss  == nom)     && (swss  == offmode)} } \
  -state { sw3off  -logic_expr  { (varss  == offmode) && (swss  == offmode)} }
```

Agenda

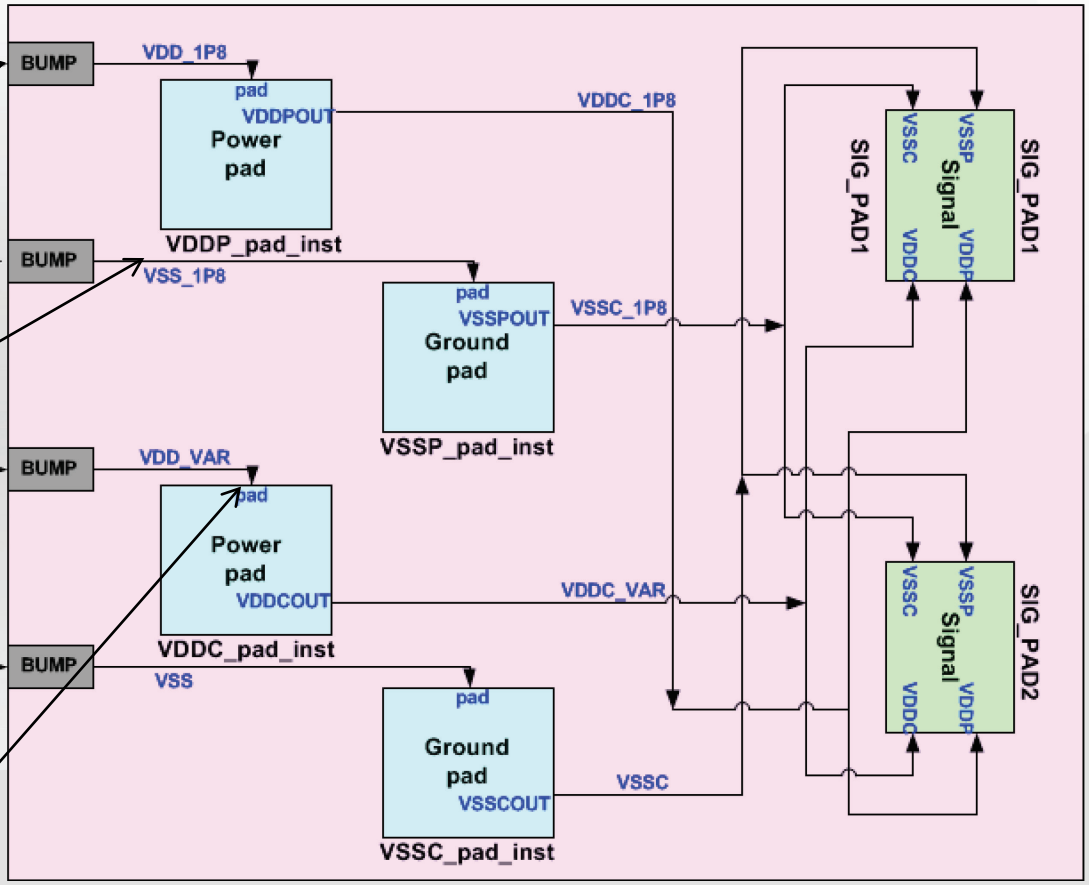
- Introduction
- **Block implementation examples**
  - Hard IP with 2 domains
  - IO pads
- SoC Integration
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

Supply Ports

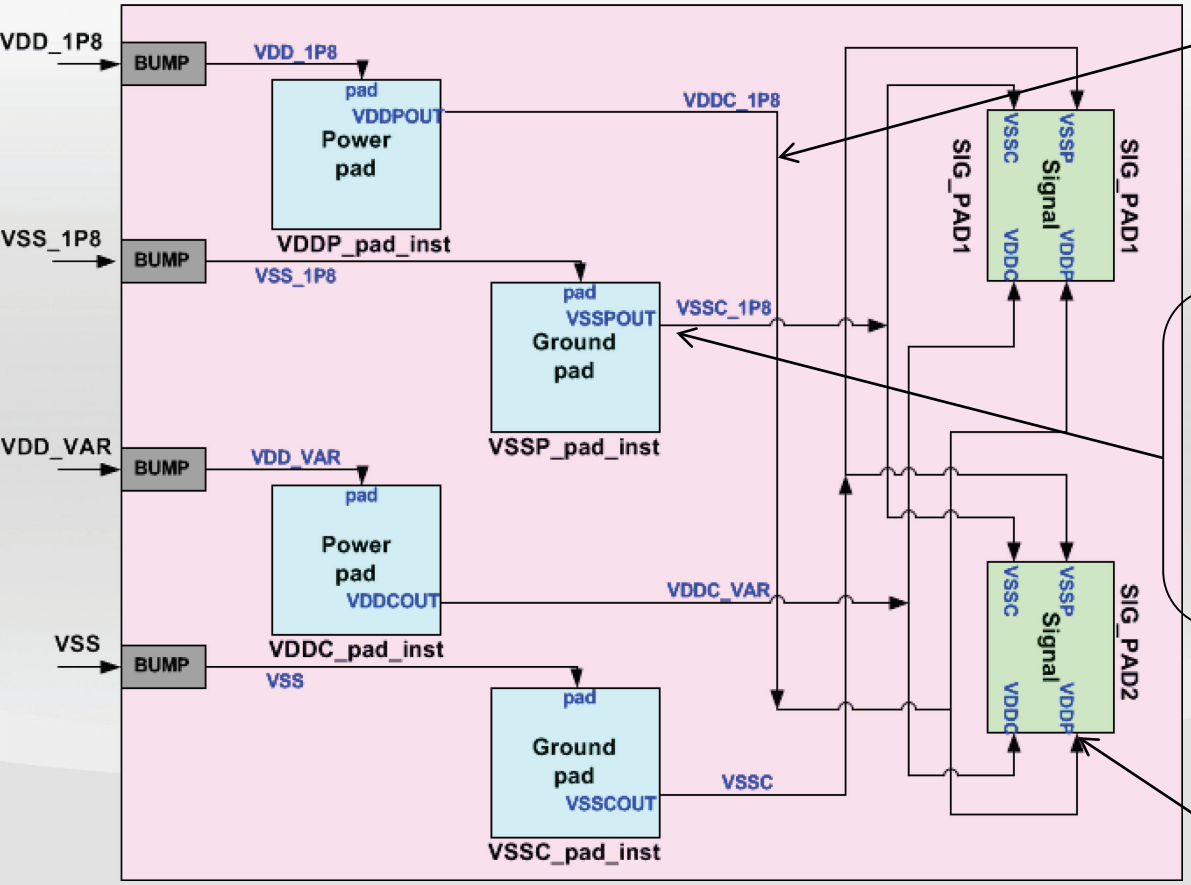
```
create_supply_port VDD_1P8
create_supply_port VSS_1P8
create_supply_port VDD_VAR
create_supply_port VSS
```

```
create_supply_net VDD_1P8 \
  -domain PD_TOP
create_supply_net VSS_1P8 \
  -domain PD_TOP
create_supply_net VDD_VAR \
  -domain PD_TOP
create_supply_net VSS \
  -domain PD_TOP
```

```
connect_supply_net VDD_1P8 \
  -ports "u_pads/VDDP_pad_inst/pad"
connect_supply_net VSS_1P8 \
  -ports "u_pads/VSSP_pad_inst/pad"
connect_supply_net VDD_VAR \
  -ports "u_pads/VDDC_pad_inst/pad"
connect_supply_net  VSS \
  -ports "u_pads/VSSC_pad_inst/pad"
```



Supply Connections




```
create_supply_net VDDC_1P8 \
  -domain PD_TOP
create_supply_net VSSC_1P8 \
  -domain PD_TOP
create_supply_net VDDC_VAR \
  -domain PD_TOP
create_supply_net  VSSC \
  -domain PD_TOP
```

```
connect_supply_net VDDC_1P8 \
-ports "u_pads/VDDP_pad_inst/VDDPOUT"
connect_supply_net VSSC_1P8 \
-ports "u_pads/VSSP_pad_inst/VSSPOUT"
connect_supply_net VDDC_VAR \
-ports "u_pads/VDDC_pad_inst/VDDCOUT"
connect_supply_net  VSSC \
-ports "u_pads/VSSC_pad_inst/VSSCOUT"
```

```
connect_supply_net VDDC_1P8 \
  -ports "u_pads/SIG_PAD1/VDDP"
connect_supply_net VSSC_1P8 \
  -ports "u_pads/SIG_PAD1/VSSP"
connect_supply_net VDDC_VAR \
  -ports "u_pads/SIG_PAD1/VDDC"
connect_supply_net  VSSC \
  -ports "u_pads/SIG_PAD1/VSSC"
```

Supply States

Supply	nom			turbo			offmode		
	VDD	VSS	Nwell	VDD	VSS	Nwell	VDD	VSS	Nwell
varss	0.8	0	0.8	0.9	0	0.9	off	0	off
1p8ss	1.8	0							

```
add_power_state varss -supply  \
  -state { nom     -supply_expr {(power == {FULL_ON 0.81}) && (ground == {FULL_ON 0}) && \
                                 (nwell == {FULL_ON 0.81}) } \
  -state { turbo   -supply_expr {(power == {FULL_ON 0.90}) && (ground == {FULL_ON 0}) && \
                                 (nwell == {FULL_ON 0.90}) } \
  -state { offmode -supply_expr {(power == {OFF}) && (ground == {FULL_ON 0}) && \
                                 (nwell == {OFF})  -simstate CORRUPT}

add_power_state 1p8ss -supply  \
  -state { nom     -supply_expr {(power == {FULL_ON 1.8}) && (ground == {FULL_ON 0}) } \
  -state { not_on  -supply_expr {(power !=  {FULL_on 1.8}) } -illegal }
```

Power States

State	varss	1p8ss
nom	nom	nom
turbo	turbo	nom
offmode	offmode	nom

```
add_power_state PD_SW -domain  \
  -state { turbo   -logic_expr  { (varss  == turbo) } } \
  -state { nom     -logic_expr  { (varss  == nom)   } } \
  -state { offmode -logic_expr  { (varss  == offmode)} -simstate CORRUPT }
```

Agenda

- Introduction
- Block implementation examples
  - Hard IP with 2 domains
  - IO pads
- **SoC Integration**
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

Design/UPF Partitioning

- **Partition design UPF into sub-module UPF**
  - Place and route block boundary
  - Power domain boundary

```
load_upf $env(UPF_PATH)/module1/upf/module1.upf \
  -scope core_inst/module1_inst
load_upf $env(UPF_PATH)/module2/upf/module2.upf \
  -scope core_inst/module2_inst
load_upf $env(UPF_PATH)/module3/upf/module3.upf \
  -scope core_inst/module3_inst
```

- **Iso/Ls inside blocks or at top level**
  - Number of domains < N, Iso/Ls insertion at top
    - Block/sub-module implementation is simplified, all domain crossings at top
  - Number of domains > N, Iso/Ls inside implementation blocks/sub-modules
    - Block/sub-module low power implementation is self contained
    - Top level domain crossings are minimized, simplifying top level implementation

Modularizing Top-Level UPF

- **Break up the contents of top level UPF into multiple files for readability**
  - Top level power ports
  - Top level power nets
  - Top level supply sets
  - Macro connections
  - System power states

```
source   $env(UPF_PATH)/top/upf/top_power_ports.upf
source   $env(UPF_PATH)/top/upf/top_power_nets.upf
source   $env(UPF_PATH)/top/upf/top_macro_connections.upf
source   $env(UPF_PATH)/top/upf/top_system_states.upf
```

IO Modeling

- **IO pad con**
  - Special structure involving multiple power supplies
  - Need many connect_supply_net connections
  - Special IO cells connected to analog constants need additional domains (hierarchies)

```
set pad_cells [[find_objects . -pattern *PAD_SEG2_inst* -object_type inst \
                               -leaf_only -transitive ]
foreach pad $pad_cells {
       connect_supply_net pad_ring_VSS -ports "$pad/VSSP"
}
```

UPF 2.1 feature

- **Supply port/net/set reduction using equivalence**
  - Several IO supplies are functionally equivalent
  - Some supplies might be connected at package level/off-chip

```
set_equivalent -function_only { AVDD VDD1P8 pad_ana_VDD }
set_equivalent -function_only { AVSS pad_AVSS ana_VSS VSS dig_VSS }
```

Handling Macros

- **Macro connections**
  - Analog macros - all non-default connections need to specified with connect_supply_net
  - Analog model should include pg_pin definitions/related_power_pin/ground_pin definitions
  - Special care needs to be taken for macros with internal supplies
    - Does that need additional top level isolation/level-shifting

```
set pll_inst [find_objects . -pattern *u_pll* -object_type inst -leaf_only \
                             -transitive]
connect_supply_net 1p8ss.power -ports "$pll_inst[0]/VDD1P8"
connect_supply_net 1p8ss.ground -ports "$pll_inst[0]/AVSS"
```

Agenda

- Introduction
- Block implementation examples
  - Hard IP with 2 domains
  - IO pads
- **SoC Integration**
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

SoC UPF: An Outline

```
load_upf    $env(UPF_PATH)/core/upf/core.upf -scope u_core
load_upf    $env(UPF_PATH)/iopads/upf/iopads.upf -scope u_pads

if { $env(HIER_MODE) eq "TRUE" } {
  load_upf    $env(UPF_PATH)/hard_block/upf/hard_block.upf -scope u_hard_block
}

create_power_domain PD_TOP -include_scope
source    $env(UPF_PATH)/top/upf/top_power_ports.upf
source    $env(UPF_PATH)/top/upf/top_power_nets.upf
source    $env(UPF_PATH)/top/upf/top_supply_sets.upf
associate_supply_set aonss -handle PD_TOP.primary

source    $env(UPF_PATH)/top/upf/top_submodule_connections.upf
source    $env(UPF_PATH)/top/upf/top_macro_connections.upf
source    $env(UPF_PATH)/top/upf/top_port_attributes.upf
source    $env(UPF_PATH)/top/upf/top_system_states.upf
source    $env(UPF_PATH)/top/upf/top_strategies.upf
```

Top-Level Supply States

Supplies	Type	nom	turbo	offmode
IO supplies	Constant	1.8V		
AON supply	Constant	0.8V		
VAR1 supply	Variable/switchable	0.8V	0.9V	Off
VAR2 supply	Variable/switchable	0.8V	0.9V	Off
VAR3 supply	Variable	0.8V	0.9V	
VAR4 supply	Switchable	0.9V		Off

Note: Some states are left out for brevity

```
add_power_state var1ss -supply  \
  -state { nom     -supply_expr { (power == {FULL_ON 0.8}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.8}) } \
  -state { turbo   -supply_expr { (power == {FULL_ON 0.9}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.9}) } \
  -state { offmode -supply_expr { (power == {OFF}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {OFF})  -simstate CORRUPT}

add_power_state var3ss -supply  \
  -state { nom     -supply_expr { (power == {FULL_ON 0.8}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.8}) } \
  -state { turbo   -supply_expr { (power == {FULL_ON 0.9}) && (ground == {FULL_ON 0}) && \
                                  (nwell == {FULL_ON 0.9}) }
```

Top-Level Power States

State	ioss	aonss	var1ss	var2ss	var3ss	var4ss
nom	nom	nom	nom	nom	nom	nom
state1	nom	nom	turbo	turbo	turbo	nom
state2	nom	nom	turbo	nom	nom	nom
state3	nom	nom	nom	turbo	nom	nom
state4	nom	nom	nom	nom	turbo	nom
state5	nom	nom	off	nom	nom	nom
.....						

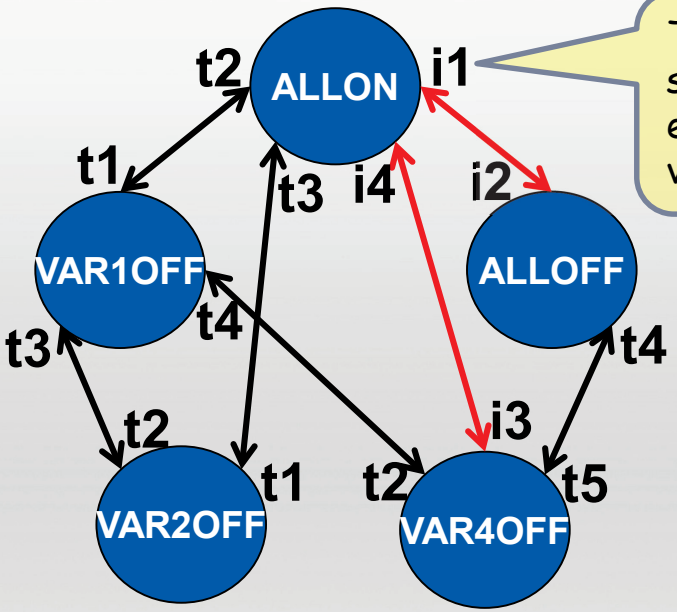
State	ioss	aonss	var1ss	var2ss	var3ss	var4ss
on	nom	nom	!off	!off	nom | turbo	!off
var1off	nom	nom	off	!off | off	nom | turbo	!off
var2off	nom	nom	!off	off	nom | turbo	!off
var4off	nom	nom	off	!off | off	nom | turbo	off
alloff	nom	nom	off	off	nom | turbo	off

Top-Level Power States

State	ioss	aonss	var1ss	var2ss	var3ss	var4ss
on	nom	nom	!off	!off	nom | turbo	!off
var1off	nom	nom	off	- (any)	nom | turbo	!off
var2off	nom	nom	!off	off	nom | turbo	!off
var4off	nom	nom	off	- (any)	nom | turbo	off
alloff	nom	nom	off	off	nom | turbo	off

```
add_power_state PD_TOP -domain  \
  -state { on           \
  -logic_expr  { (var1ss  != offmode) && (var2ss  != offmode)  && \
                 (var3ss  == nom || var3ss==turbo) && (var4ss  != offmode)} }  \
  -state { var1off    \
  -logic_expr  { (var1ss  == offmode) && (var3ss  == nom || var3ss==turbo) && \
                 (var4ss  != offmode)} }  \
  -state { var2off    \
  -logic_expr  { (var1ss  != offmode) && (var2ss  == offmode) && \
                 (var3ss  == nom || var3ss==turbo) && (var4ss  != offmode)} }  \
  -state { var4off    \
  -logic_expr  { (var1ss  == offmode) && (var3ss  == nom || var3ss==turbo) && \
                 (var4ss  == offmode)} }  \
  -state { alloff       \
  -logic_expr  { (var1ss  == offmode) && (var2ss  == offmode) && \
                 (var3ss  == nom || var3ss==turbo) && (var4ss  == offmode)} }
```

Top-Level Power Transitions



Transition name next to arrow stands for transition TO the state, eg: t4 is a transition TO either var1off or alloff from var4off

```
describe_state_transition i1 -object PD_TOP -from {ALLOFF}  -to {ALLON}   -illegal
describe_state_transition i2 -object PD_TOP -from {ALLON}   -to {ALLOFF}  -illegal
describe_state_transition i3 -object PD_TOP -from {ALLON}   -to {VAR4OFF} -illegal
describe_state_transition i4 -object PD_TOP -from {VAR4OFF} -to {ALLON}   -illegal

describe_state_transition t1 -object PD_TOP -from {ALLON}   -to {VAR1OFF VAR2OFF}
describe_state_transition t2 -object PD_TOP -from {VAR1OFF} -to {ALLON VAR2OFF VAR4OFF}
describe_state_transition t3 -object PD_TOP -from {VAR2OFF} -to {VAR1OFF ALLON}
describe_state_transition t4 -object PD_TOP -from {VAR4OFF} -to {VAR1OFF ALLOFF}
describe_state_transition t4 -object PD_TOP -from {ALLOFF}  -to {VAR4OFF}
```

Agenda

- Introduction
- Block implementation examples
  - Hard IP with 2 domains
  - IO pads
- **SoC Integration**
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

SoC Test-bench and Test-bench UPF

- **SOC Test-bench UPF**

```
set_design_top top/chip_tb_inst
load_upf $env(UPF_PATH)/chip/upf/chip.upf -scope u_chip
# Any additional user attributes
```

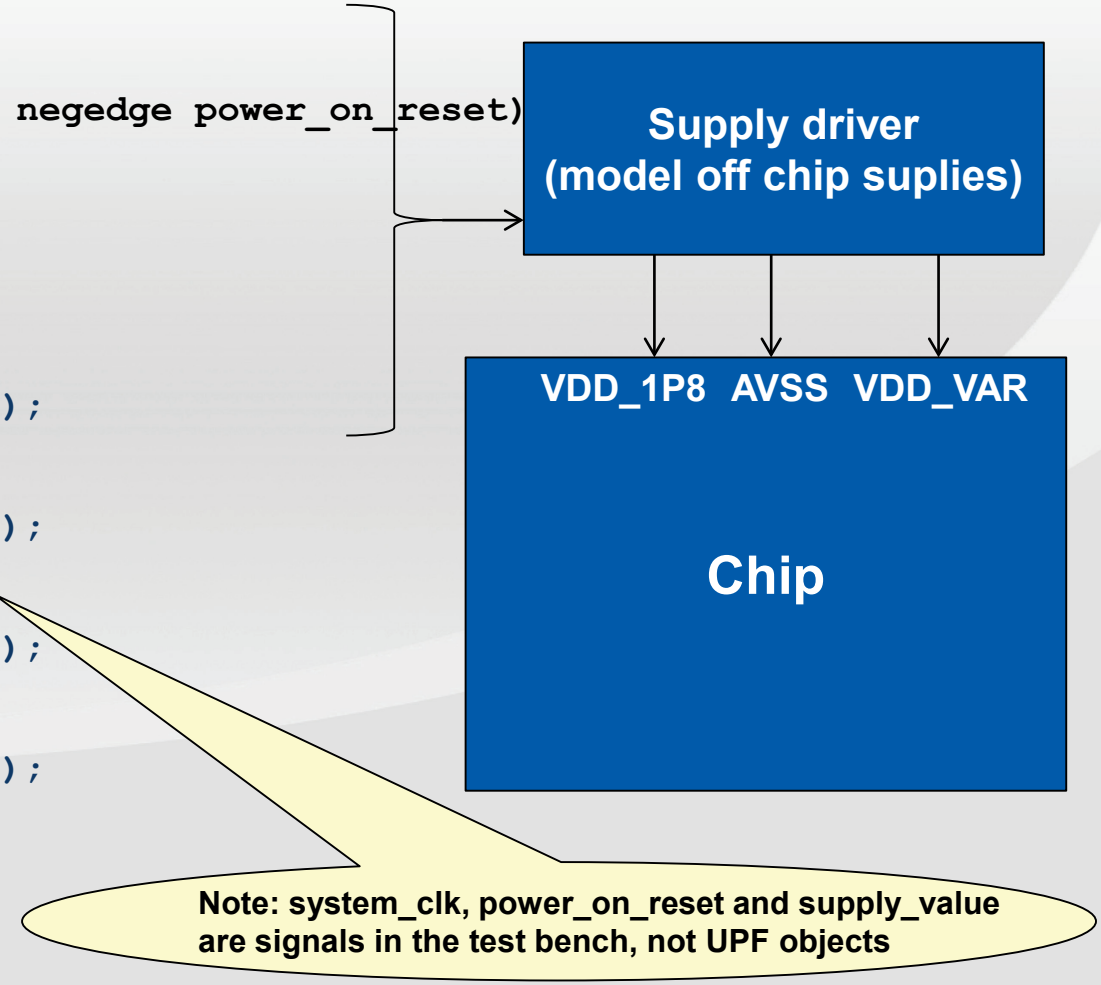
- **Test-bench**

```
module chip_tb;
.......
`ifdef DEFINE_UPF_PKG
   import UPF::*;
`endif
// Constant supplies
initial
begin
  supply_on("VDD_1P8", 1.8);
  supply_on("AVSS", 0);
```

SoC Test-bench and Test-bench UPF

- **Test-bench (contd....)**

```
// Dynamic supplies
always @ (posedge system_clk, negedge power_on_reset)
begin
  if(supply_requested)
  begin
    if(supply_value==0x1)
       supply_on("VDD_VAR",0.8);
    else if(supply_value==0x2)
       supply_on("VDD_VAR",0.9);
    else if(supply_value==0x3)
       supply_on("VDD_VAR",1.0);
    else
       supply_on("VDD_VAR",0.7);
  end
  else
    supply_off("VDD_VAR",0);
  end
end
```



Summary

- **Introduction to building system level power intent**
- **Block implementation examples**
  - Hard IP with 2 domains
  - IO pads
- **SoC Integration**
  - Useful tips
  - An outline
  - Top level power states
  - Verification bench and UPF

What's New in UPF 2.1

Qi Wang

Technical Marketing Group Director

Cadence Design Systems, Inc.

cadence®

What's New in IEEE 1801-2013 (UPF2.1)

Sample Features

- Supply Equivalence
- Buffer Insertion
- Power Domain Boundaries
- Modeling Hard Macros
- Defining Power Management Cells
- Methodology Implications

Supply Equivalence

- **Two types of supply equivalence**
  - Electrical equivalence → determined by connection
  - Functional equivalence → determined by connection or declaration
    - if A and B are electrically equivalent, then A and B are functionally equivalent;
    - if A and B are declared functionally equivalent, then A and B are functionally equivalent
  - Electrical equivalence implies functional equivalence, but not vice-versa
- **What does this imply?**
  - Two anonymous supply sets built from equivalent power/ground functions are equivalent;
  - Two supply sets that are functionally equivalent can be used interchangeably;
  - A supply set and any supply set handle it is associated with are always equivalent

Language Impact of Equivalence

Supply equivalence affects various commands:

- **`set_isolation/set_level_shifter/set_repeater`**
  - whether a driver/receiver supply set matches a **`-source/-sink`** filter
- **`set_isolation`**
  - whether a driver/receiver supply satisfies the **`-diff_supply_only`** filter
- **`create_power_domain`**
  - whether a port of a macro instance is on a power domain's (lower) boundary
- **`create_composite_domain`**
  - whether two domains can be composed

Buffer Insertion

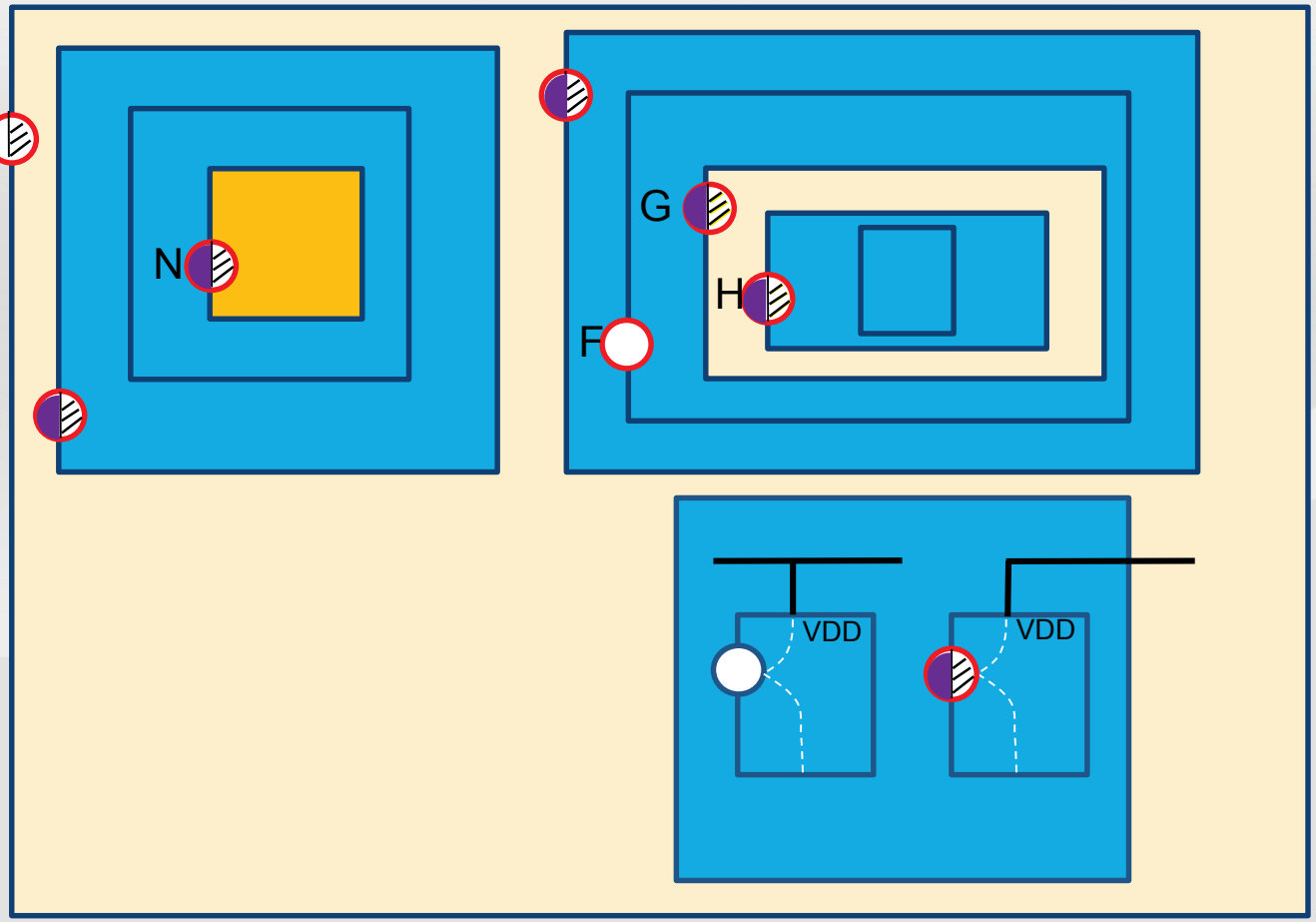
- The `-repeater_supply` of `set_port_attributes` is deprecated
- New command introduced to handle "always-on" buffer insertion

```
set_repeater Buff -domain PD1 -elements {DataIn, AddrIn, DataOut}

set_repeater Buff -domain PD2 -source PD1 -sink PD3 -applies_to inputs

set_repeater Buff -domain PD2 -update -instance {buf1 buf2}
```

- Semantics on the interaction of this command versus other commands are well defined
  - The command will have an impact on the driver/receiver supply analysis for strategies

Power Domain Boundaries

not a boundary  lower boundary  upper boundary



Modeling Hard Macros

- **A structured way to model power intents for hard IP**
- **Three new commands**
  - 2 commands for model definition

    ```
    begin_power_model
    end_power_model
    ```

  - 1 command for model application

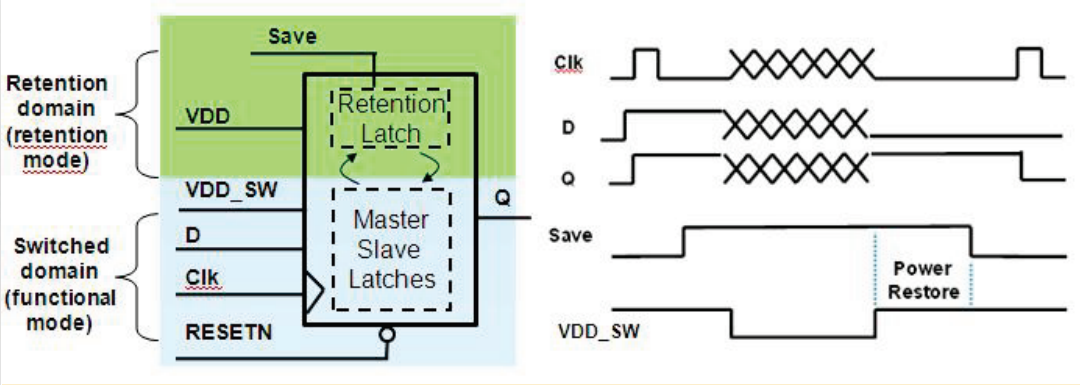
    ```
    apply_power_model
    ```

- **Benefits**
  - Enforce the hierarchical modeling methodology
  - Enable complex hard IP modeling with power states specification

Defining Power Management Cells

Be able to define special power management cells within 1801

- `define_always_on_cell`
- `define_isolation_cell`
- `define_level_shifter_cell`
- `define_power_switch_cell`
- `define_retention_cell`
- `define_diode_clamp`




```
define_retention_cell -cells SR1 \
  -clock_pin Clk \
  -save_function {save posedge} \
  -restore_check !Clk -save_check !Clk \
  -power_switchable VDD_SW \
  -power VDD -ground VSS
```

Summary of IEEE 1801-2013

- **Deletions and Deprecations**
  - many UPF 1.0 commands and options
  - some annexes
- **Restrictions**
  - supply set functions
  - supply_expr, logic_expr
- **Extensions/Additions**
  - precedence rules
  - set_equivalent
  - set_repeater
  - `find_objects` for module type
  - new `CORRUPT_ON_CHANGE` simstate
  - new options for existing commands
  - new power intent commands
  - power mgmt cell commands
  - annexes on
    - low power design methodology
    - supporting hard IP
    - UPF power mgmt commands & Liberty
    - power mgmt cell modeling
- **Syntax Clarifications**
  - UPF and Tcl
  - list syntax
  - Boolean expressions
  - `set_port_attributes`
  - `set_design_attributes`
- **Semantic Clarifications**
  - upf_version
  - set_design_top
  - hierarchical names
  - supply set semantics
  - source/sink filters
  - repeater/retention/iso/ls order
  - domain/system power states
  - UPF attributes
  - UPF processing stages
  - power state simulation semantics
  - power switch simulation semantics