Using UPF for Low Power Design and Verification

- Tutorial #2: presented by members of the IEEE P1801 WG

- John Biggs
- Erich Marschner
- Sushma Honnavara-Prasad
- David Cheng
- Shreedhar Ramachandra
- Jon Worthington
- Nagu Dhanwada

Welcome and Introductions

Erich Marschner

Verification Architect

Mentor Graphics

Tutorial #2

- This tutorial presents the latest information on the Unified Power Format (UPF), based on IEEE Std 1801-2013 UPF which was released in late May of last year.
- Beginning with a review of the concepts, terminology, commands, and options provided by UPF, it will cover the full spectrum of UPF capabilities and methodology, from basic flows through advanced applications, with particular focus on incremental adoption of UPF.
- Tutorial attendees will come away with a thorough understanding of UPF usage in low power design and verification and its role in energy aware system design.

Takeaways

- Why power is important - and challenging
- How power affects implementation decisions
- What UPF is and what problems it addresses
- How UPF enables early consideration of power
- UPF concepts, commands, semantics, and usage
- UPF methodology for modeling, design, integration
- How UPF can be adopted most effectively
- What else we need to address related to power

Presenters

- **John Biggs, ARM Ltd.**
  - Chair, IEEE P1801 UPF Work Group
- **Erich Marschner, Mentor Graphics**
  - Vice-Chair, IEEE P1801 UPF Work Group
- **Sushma Honnavara-Prasad, Broadcom**
  - Secretary, IEEE P1801 UPF Work Group
- **David Cheng, Cadence**
  - Member, IEEE P1801 UPF Work Group
- **Shreedhar Ramachandra, Synopsys**
  - Member, IEEE P1801 UPF Work Group

Other Contributors

- **Jon Worthington, Synopsys**
  - Member, IEEE P1801 UPF Work Group
- **Nagu Dhanwada, IBM**
  - Member, Si2 LPC and LPSG

Agenda

- **Welcome & Introductions**
  - Erich Marschner
- **Low Power Design and Verification Challenges**
  - Erich Marschner
- **Introduction to UPF**
  - John Biggs
- **UPF Basic Concepts and Terminology**
  - Shreedhar Ramachandra
- **UPF Semantics and Usage**
  - Erich Marschner
- **BREAK**
- **Hard IP Modeling with Liberty and Verilog**
  - Sushma Honnavara-Prasad
- **Power Management Cell Commands and Power Models**
  - David Cheng
- **Low Power Design Methodology for IP Providers**
  - John Biggs
- **SoC-Level Design and Verification Challenges**
  - Sushma Honnavara-Prasad
- **Adopting UPF**
  - Sushma Honnavara-Prasad
  - Shreedhar Ramachandra
- **Where We Go From Here**
  - John Biggs

Low Power Design and Verification Challenges*

Erich Marschner

Verification Architect

Mentor Graphics

* With slides contributed by Nagu Dhanwada and Sushma Honnavara-Prasad

Low Power Design Challenges

- **Power density is increasing every process node**
  - Higher performance
  - Lower area
  - > Designs are thermally limited
  - > No single technique serves all purposes
  - > Aggressive power gating is used to minimize leakage
- **Number of power domains and supplies on chip is increasing**
  - Early architecture decisions impacts power, so early power exploration is critical

SoC Low Power Design Challenges

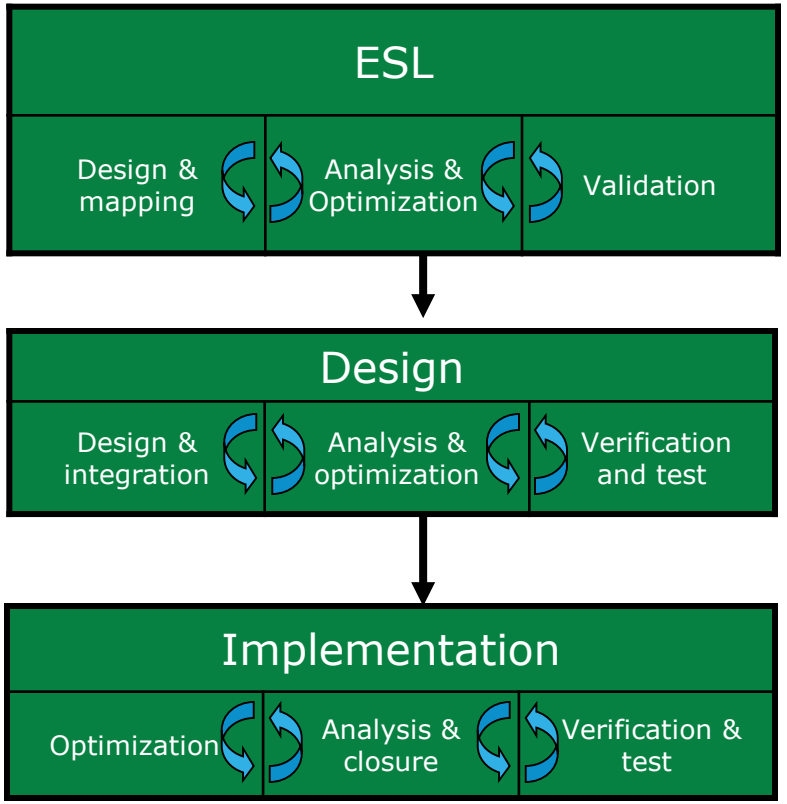
- **Reducing power with:**
  - low area overhead
  - high performance
  - low schedule impact

- **Early power estimation and budgeting**
  - Silicon variation
  - Correlation of models with Si data
  - Lack of use case vectors

- **Power aware flow**
  - Making all phases of design power aware
  - Capturing power intent accurately
  - Ensuring power intent is correctly implemented
  - Verifying power intent with firmware/Software drivers

Three Phases of a Power Aware Flow

- **ESL (algorithm and system models)**
  - Functionality
  - Architecture
  - TLM
  - Firmware
- **Design (RTL and IP)**
  - RTL module design/selection
  - IP selection and Chip integration
- **Implementation**
  - Synthesis
  - Physical design



Power Analysis is Required Throughout*

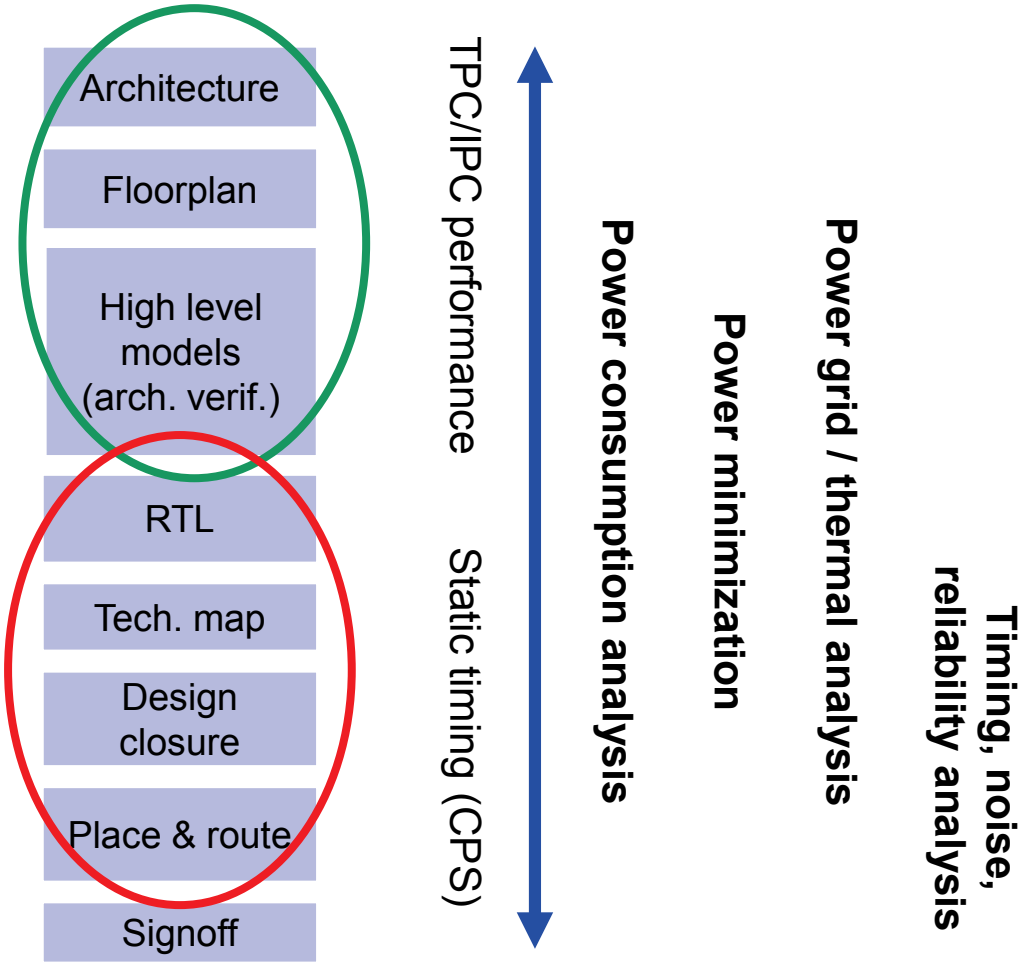
*** But with varying criteria:**

Greatest power savings opportunity
(design exploration)

Need faster analysis, can afford lower accuracy / detail

Automated / Manual power reduction

Need higher accuracy, can afford longer run times

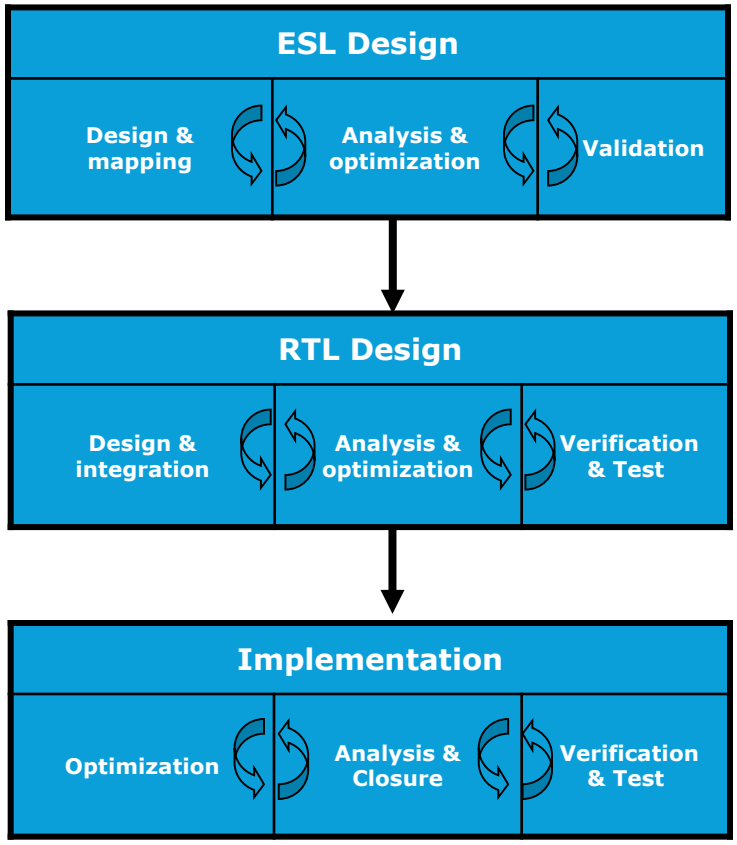


No Single Metric Handles Everything

- **Battery Life**
  –Total chip power over long time period
- **Package Inductance**
  –Total chip power over short time period
- **Reliability / Electromigration**
  –Very local power over a very long time period
- **Static IR Drop**
  –Local power over moderate time period
- **Decap / Transient IR drop**
  –Local power over very short time periods

Power-Awareness Needed in Each Phase

Design Phase



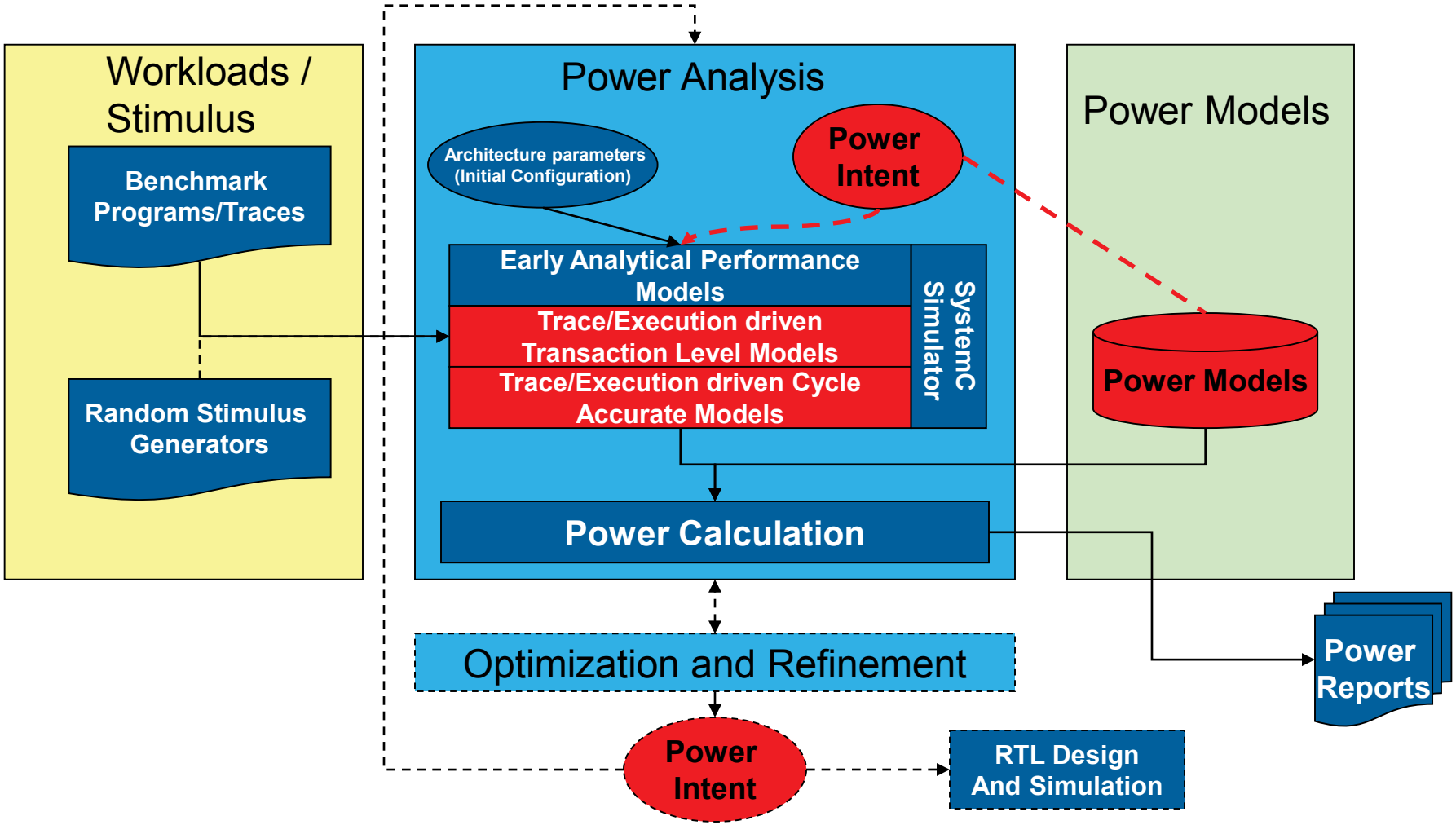
Low Power Design Activities

- Explore architectures and algorithms for power efficiency
- Map functions to sw and/or IP blocks for power efficiency
- Choose voltages and frequencies
- Evaluate power consumption for each operational mode
- Generate budgets for power, performance, area

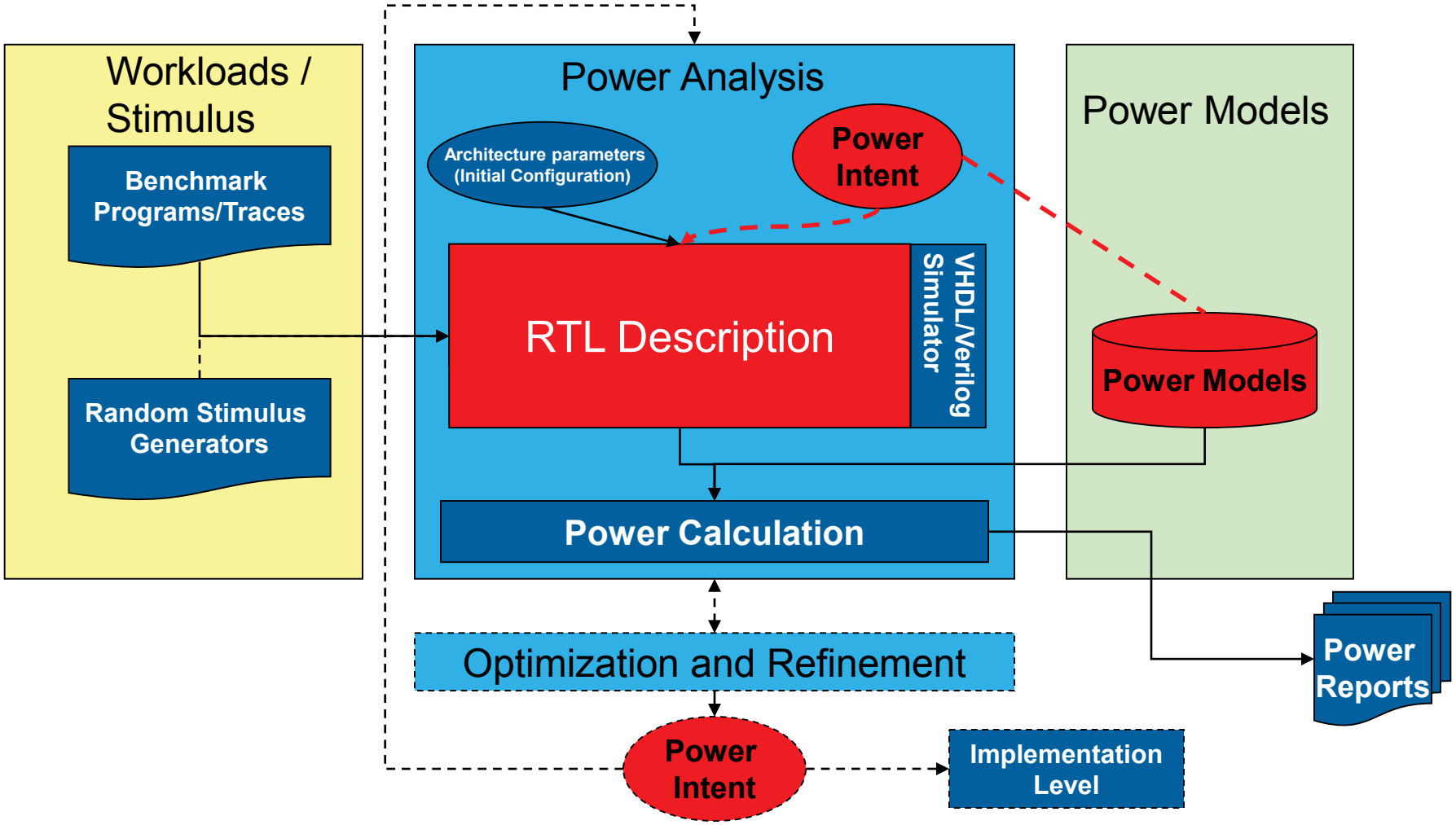
- Generate RTL to match system-level model
- Select IP blocks
- Analyze and optimize power at module and chip levels
- Analyze power implications of test features
- Check power against budget for various modes

- Synthesize RTL to gates using power optimizations
- Floorplan, place and route design
- Optimize dynamic and leakage power
- Verify power budgets and power delivery

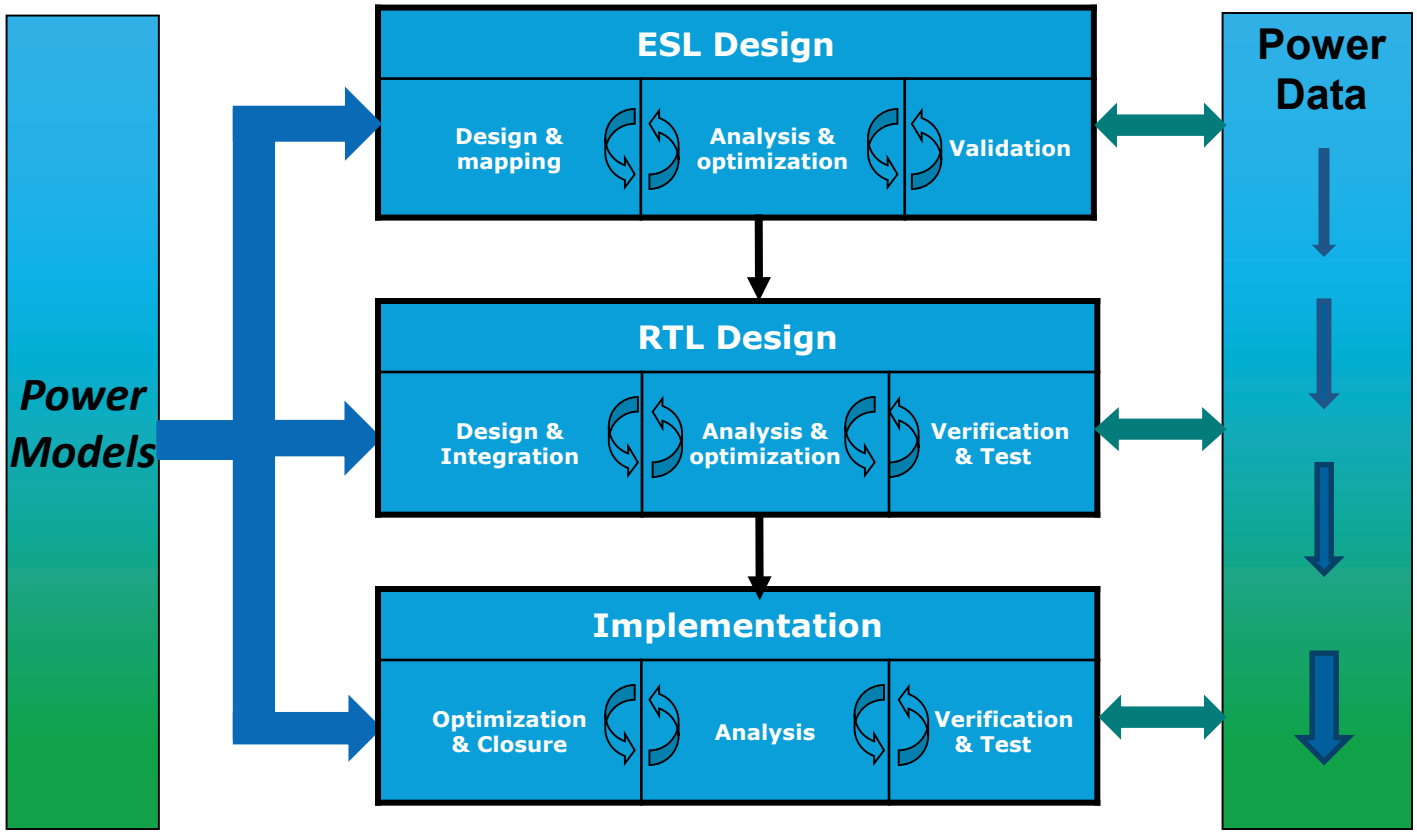
Power Analysis Flow at ESL



Power Analysis Flow at RTL

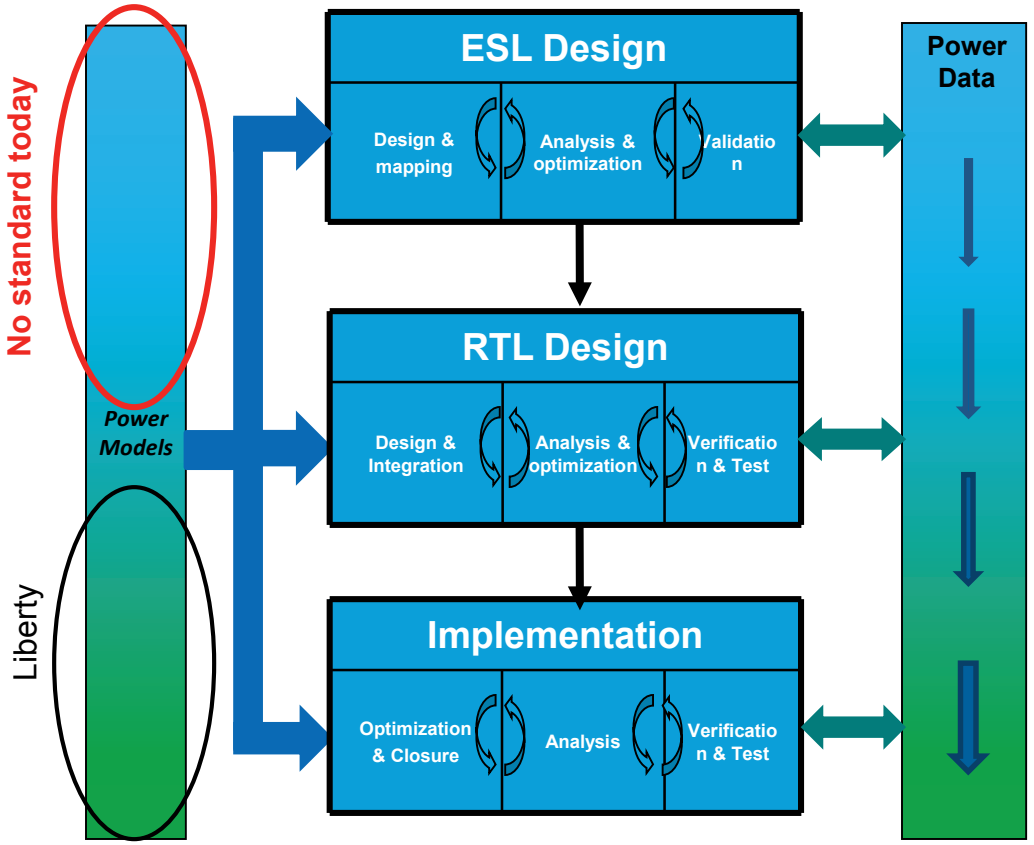


System-to-Silicon Power Aware Design Flow

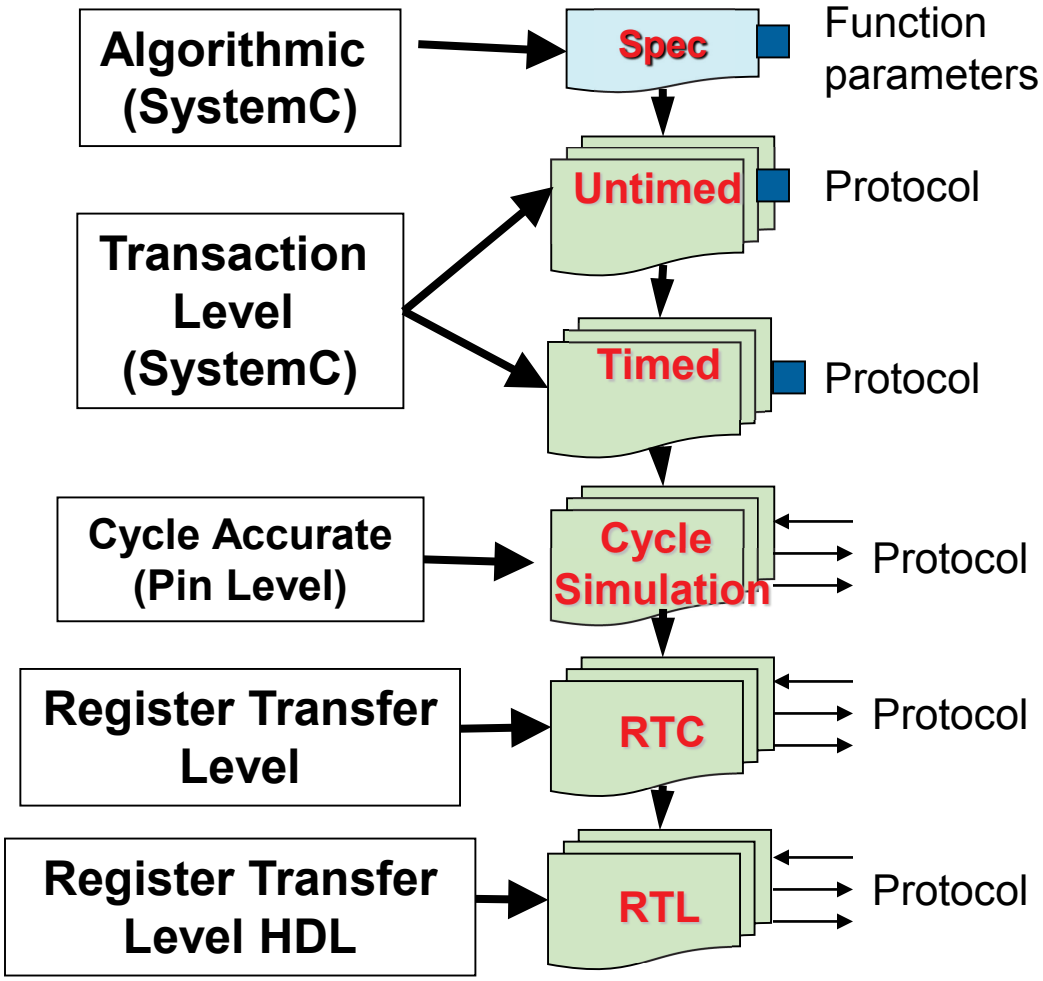


Where Are the Models?

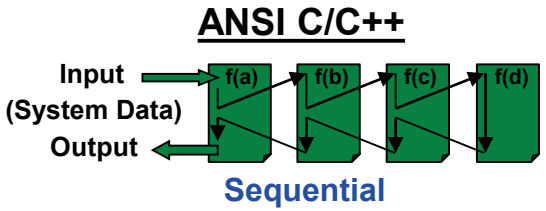
- **Flow execution is limited by model availability**
  - Implementation tasks rely upon gate level models
  - RTL tasks rely upon gate level models (directly or indirectly)
  - ESL? …. Ad hoc solutions…
- **What about IP blocks?**
  - Behavioral models are available for ESL & RTL, but without power
  - Simplistic power models may be available, …accuracy? reliability?
- **We have a problem**
  - Gate level models can be used for IP power simulations, but simulation time and resources are prohibitive

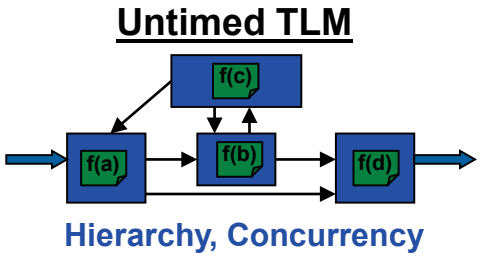


Must Handle Different Kinds of Models



Modeling Paradigms and Use Scenarios





ANSI C/C++ or Algorithmic SystemC

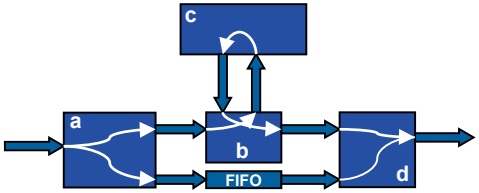
- Used to functionally verify algorithm, with no focus on implementation issues.
- HW and SW algorithms are described in the same way.
- At best design is partitioned into functional blocks, but what lies in software or hardware is not considered.

"Untimed" SystemC Transaction Level Model

- Used to manually partition system.
- No reset, clocks, or timing, but enough synchronization to enable correct functionality.
- Instruction set simulation models / virtual platforms for embedded software development would fall into this category.

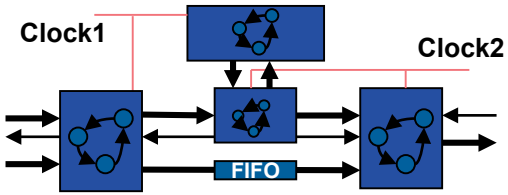
Modeling Paradigms and Use Scenarios

Timed TLM



Hierarchy, Concurrency & Timing

Cycle Accurate Model



Interface Pin & Cycle Accurate

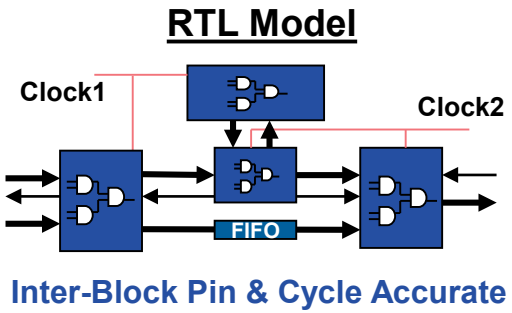
"Timed" SystemC Transaction Level Model

- Used to design and test the "system timing budget".
- Models have timing information, transactions will take correct number of clock cycles to execute but not every clock accounted for within transactions.
- Specialized structures (FIFOs) absorb irregular data rates.
- Budget defines synthesis constraints for each block.

Cycle Accurate "Behavioral" SystemC Model

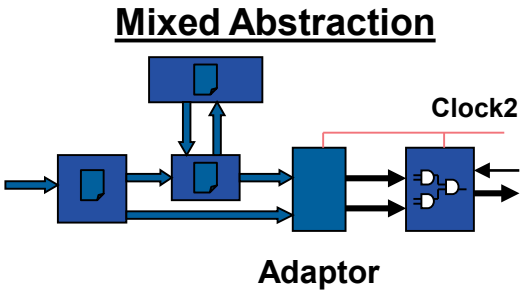
- Used to verify cycle level and system behavior at a low level.
- Models clocking in detail, i.e all clocks ticking.
- Model full pin and cycle timing at block interfaces.
- Behaves as RTL but optimized internally for speed.

Modeling Paradigms and Use Scenarios



RTL Model

- Model used for implementation via RTL synthesis.
- Completely cycle accurate (external and internal).
- Logic between clock ticks is simulated.

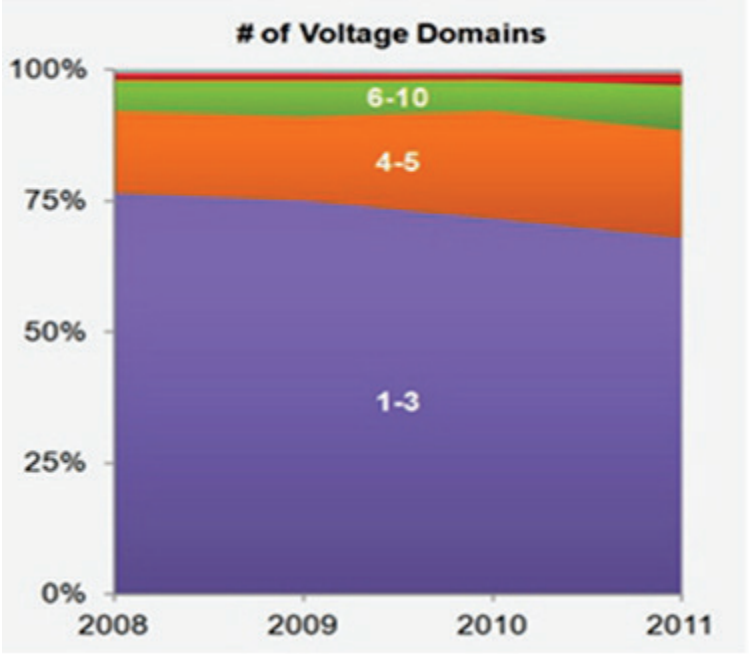


Mixed Abstraction Level Modeling

- Used for mixed mode system/block verification.
- "transactors or adaptors" are used to connect different abstractions within a system.

SoC Low Power Verification Challenges

- **Modeling issues**
  - Are all the low power features correctly represented?
- **Capacity issues**
  - Number of domains
  - Size of the chip
- **Complexity issues**
  - Chip-level power state complexity
  - Complex interactions between blocks
- **Coverage issue**
  - Power state coverage
  - System level use cases
  - H/W-S/W co-verification



Source: Synopsys 2012 global user survey

So Where Does UPF Fit In?

- **UPF models active power management**
  - A form of power optimization
- **UPF Power Intent applies at RTL and GL today**
  - RTL for "early" verification and to drive implementation flow
  - GL for verification of implementation stages
- **UPF concepts can be extended to other models**
  - Power states and transitions in particular
  - Component-based modeling paradigm can also apply

Introduction to UPF

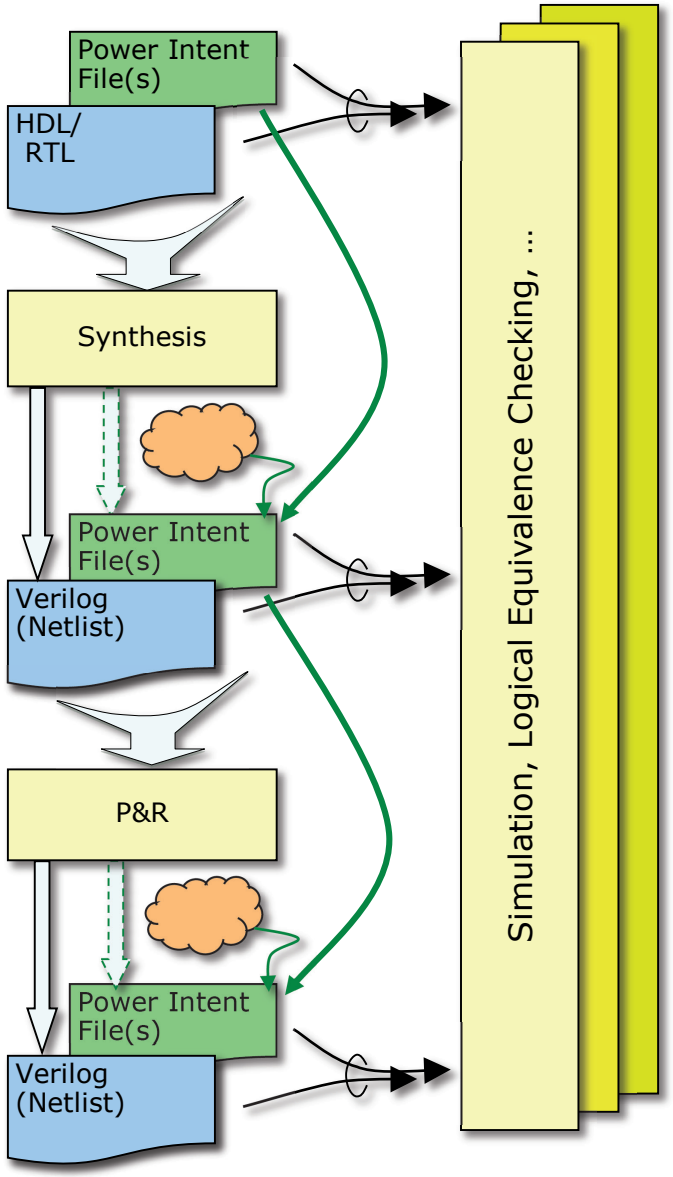
John Biggs

Senior Principal Engineer

ARM

What is UPF?

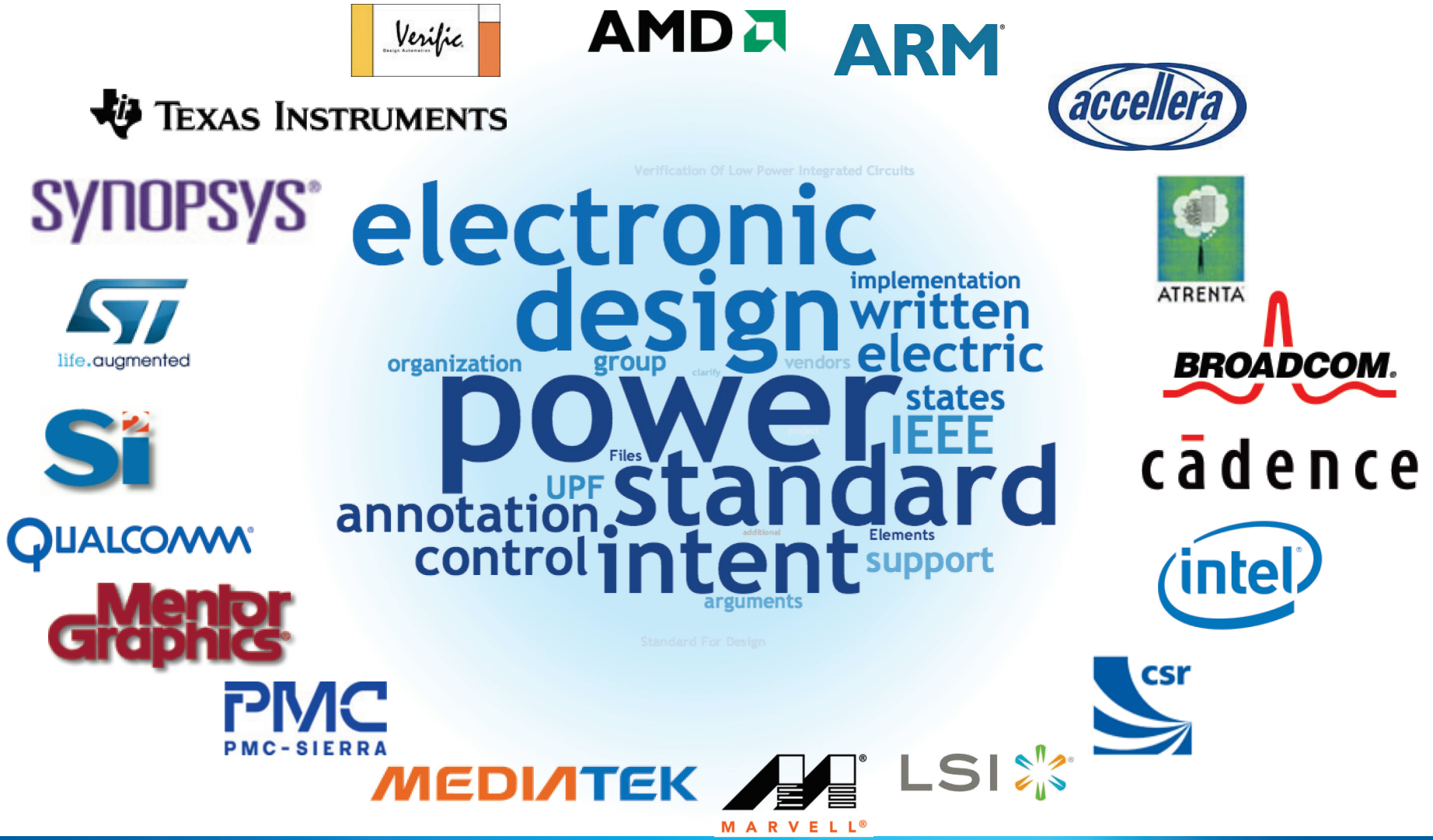
- **An Evolving Standard**
  - Accellera UPF in 2007 (1.0)
  - IEEE 1801-2009 UPF (2.0)
  - IEEE 1801-2013 UPF (2.1)
- **For Power Intent**
  - To define power management
  - To minimize power consumption
- **Based upon Tcl**
  - Tcl syntax and semantics
  - Can be mixed with non-UPF Tcl
- **And HDLs**
  - SystemVerilog, Verilog, VHDL
- **For Verification**
  - Simulation or Emulation
  - Static/Formal Verification
- **And for Implementation**
  - Synthesis, DFT, P&R, etc.



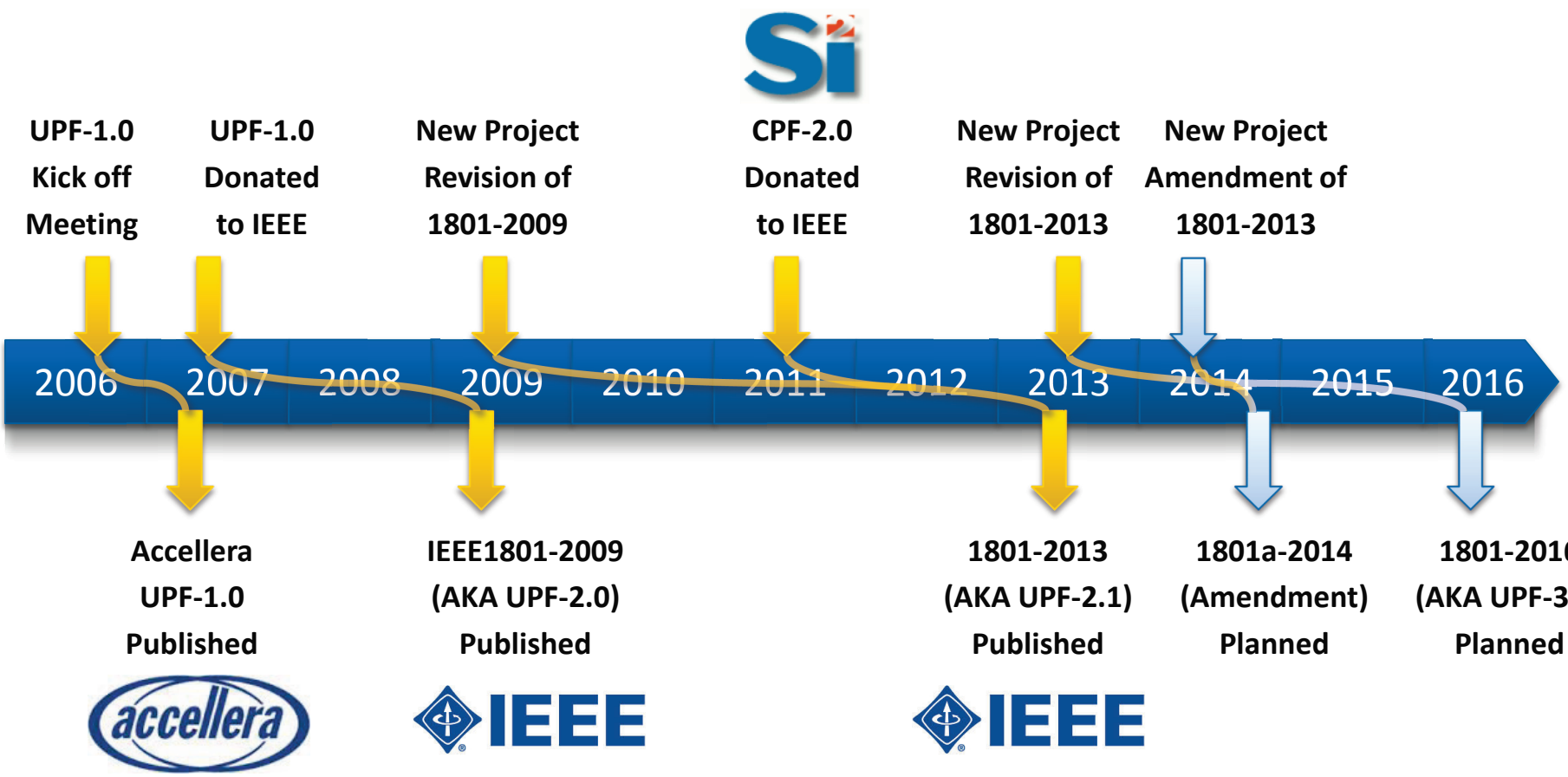
Components of UPF

- **Power Domain:**
  - Groups of elements which share a common set of power supply requirements
- **Power Supply Network**
  - Abstract description of power distribution (ports, nets, sets & switches)
- **Power State Table**
  - The legal combinations of states of each power domain
- **Isolation Strategies**
  - How the interface to a power domain should be isolated when its primary power supply is removed
- **Retention Strategies**
  - What registered state in a power domain should be retained when its primary power supply is removed
- **Level Shifter Strategies**
  - How signals connecting power domains operating at different voltages should be shifted
- **Repeater Strategies**
  - How domain ports should be bufffered

P1801: IEEE-SA Entity Based Work Group



IEEE 1801 (UPF) timeline



Accellera UPF-1.0 (2007)

Key:
- Accellera UPF-1.0 (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-2.0)

Key:
- Accellera UPF-1.0 (2007)
- IEEE 1801-2009 (UPF-2.0)

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-2.1)

Key:
- Accellera UPF-1.0 (2007)
- IEEE 1801-2009 (UPF-2.0)
- IEEE 1801-2013 (UPF-2.1)
- Deprecated/Legacy

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

The IEEE 1801-2013 Standard

- **Motivation**
  - Address known issues with 1801-2009
    - Improve the clarity and consistency
  - Syntax clarifications, semantic clarifications
    - Some restrictions, some additions
  - Include limited number of critical enhancements
    - Improved support for macro cell modeling
    - Attribution library pins/cells with low power meta data

- **Additional contributions:**
  - Cadence: Library Cell Modeling Guide Using CPF
  - Cadence: Hierarchical Power Intent Modeling Guide Using CPF
  - Si2: Common Power Format Specification, Version 2.0

  => Improved methodology convergence with CPF flows

The IEEE 1801-2013 Standard

- **Revisited each and every command**
  - Rewrote the major strategy commands
- **Rewrote many key sections:**
  - Definitions, UPF Concepts, Language Basics, Simulation Semantics
- **Added new sections:**
  - Power management cell commands, UPF processing,
  - Informative Annex on Low Power Design Methodology
- **Final Draft (D14) approved by IEEE-SA March 2013**
  - A 95% (19/20) approval rate on a 95% (20/21) return.
  - One the largest entity base ballot pools in IEEE-SA history
- **IEEE1801-2013 Published May 30th 2013**
  - Available at no charge via the IEEE Get™Program
    - http://standards.ieee.org/findstds/standard/1801-2013.html
    - http://standards.ieee.org/getieee/1801/download/1801-2013.pdf

IEEE STANDARD

1801-2013 - IEEE Standard for Design and Verification of Low-Power Integrated Circuits

Description: A method is provided for specifying power intent for an electronic design, for use in verification of the structure and behavior of the design in the context of a given power management architecture, and for driving implementation of that power management architecture. The method supports incremental refinement of power intent specifications required for IP-based design flows.

STATUS: Active Standard

Working Group: UPF - UPF: Standard for Design and Verification of Low Power Integrated Circuits

Oversight Committee: C/DA - Design Automation

Sponsor: IEEE Computer Society

Get This Standard

Buy an Electronic Copy
Get an Adobe Acrobat PDF version of this standard.

~~Buy~~

Access via the IEEE Get Program

Access with Subscription
Standards Online subscribers can access this standard in IEEE Xplore Digital Library.

Access | Learn More

RELATED MATERIALS

P1801

RELATED STANDARDS

Industry Applications Standards
Computer Technology Standards

ADDITIONAL RESOURCES

Request Interpretation
Related Products
Standards Distributors & Resellers

JOIN IEEE-SA & SAVE!

IEEE-SA Individual Members save 10% (on average)* on most standards purchased through IEEE.

Learn More about IEEE-SA Membership Options

** actual savings vary per standard*

UPF Basic Concepts and Terminology*

Shreedhar Ramachandra

Staff Engineer

Synopsys

* Slides contributed by Jon Worthington

Functional Intent vs. Power Intent
What is the difference?

Functional 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

Captured in RTL

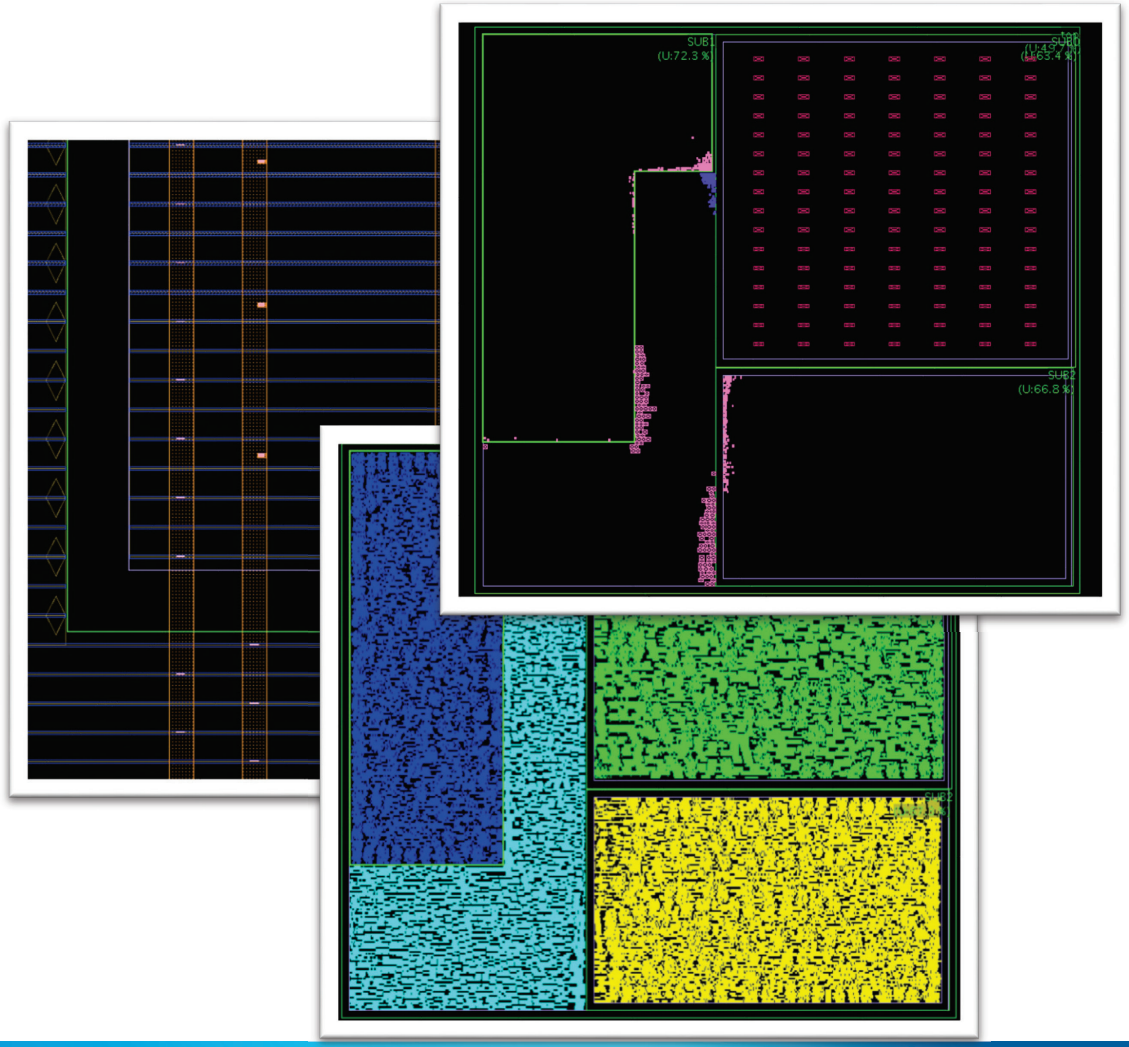
Power intent specifies

- **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 UPF

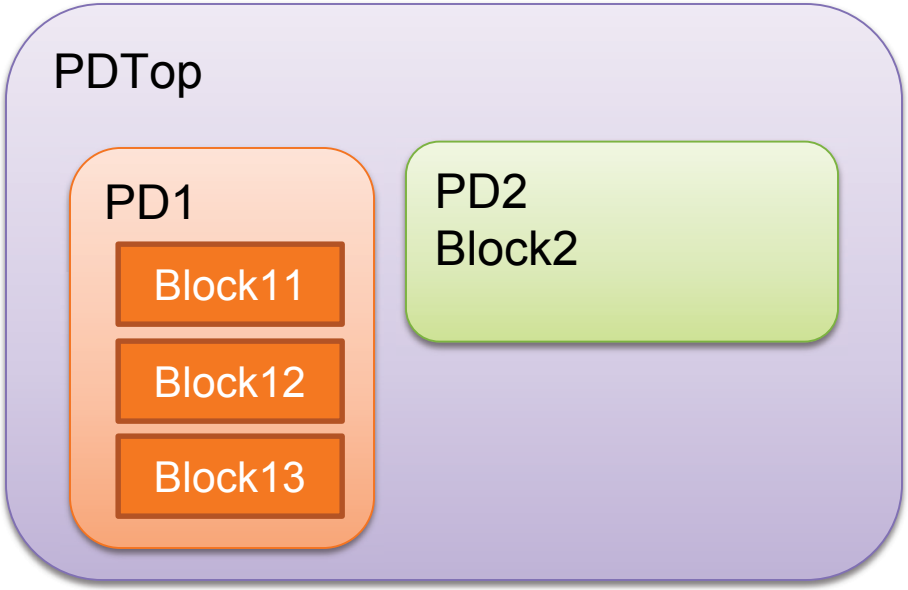
How Power Intent Affects Implementation

- **The power intent will have an impact on the implementation of the design.**
  - Domains may need separate floorplans/regions.
  - Cells may need special power routing.
- **This section will introduce power intent concepts in UPF and how they relate to implementation.**



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. ...

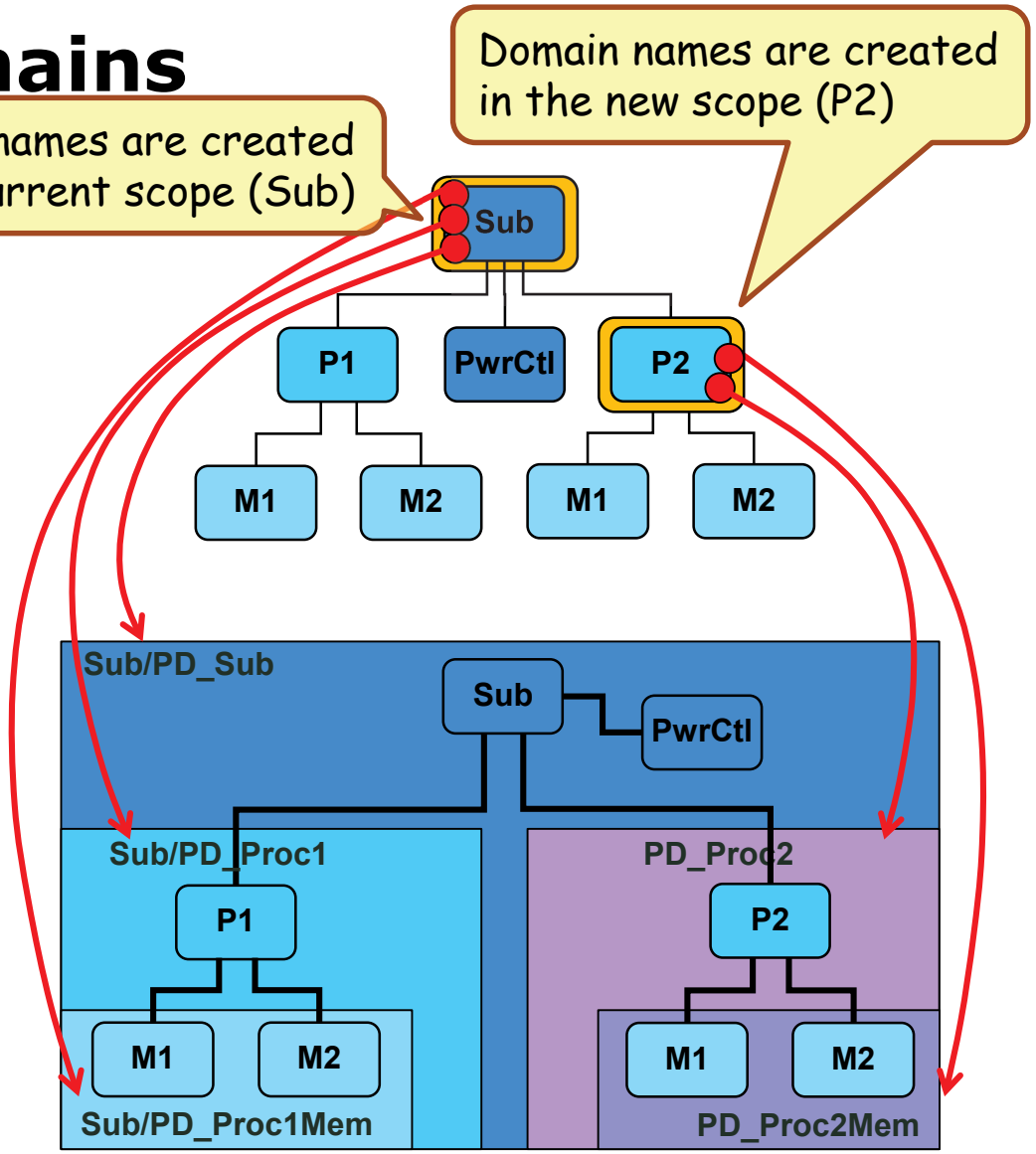


Power Domain PD1 contains 3 instances
Power Domain PD2 contains only Block2

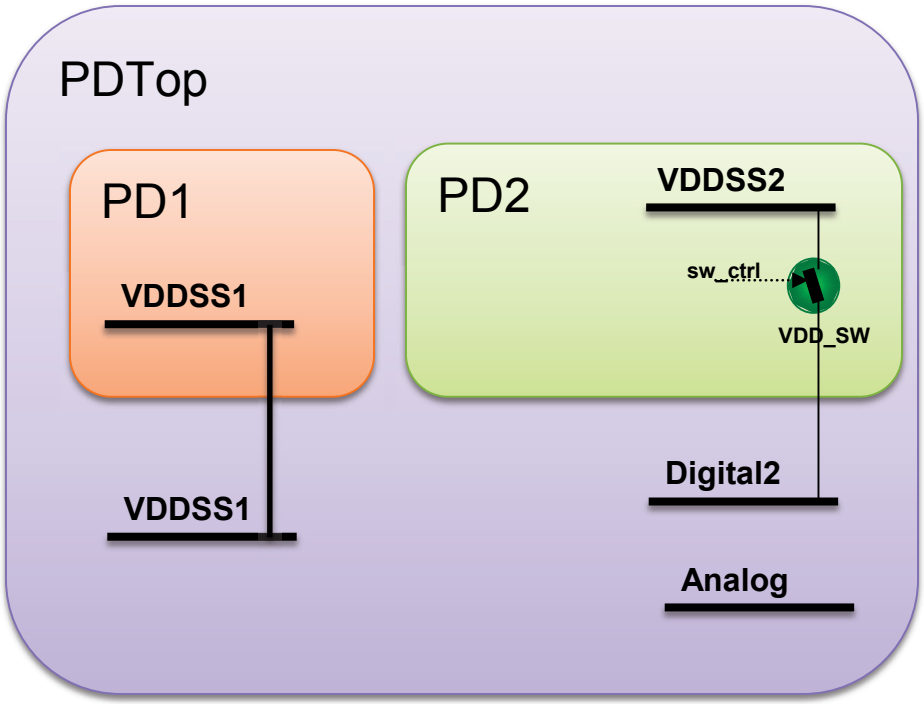
UPF - Power Domains

- **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}

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



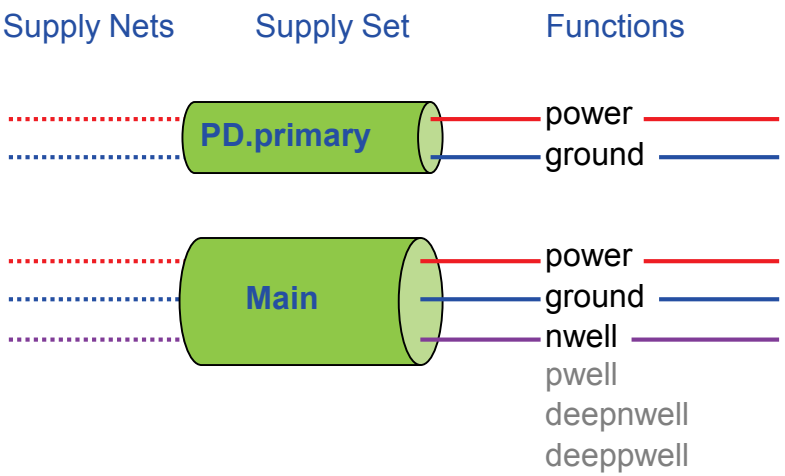
Power Supply Network



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

UPF Power Supply Networks - 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



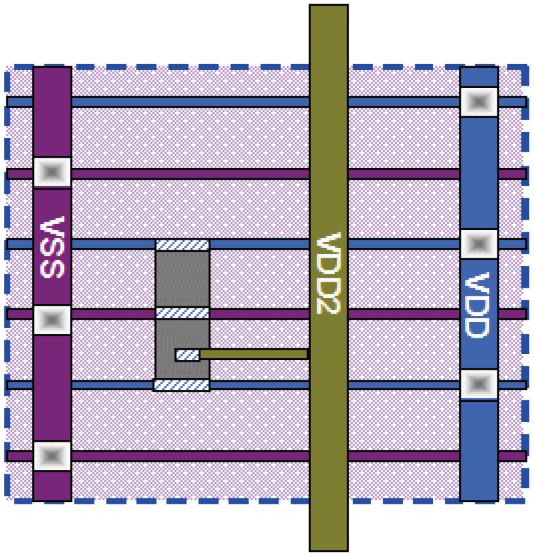

```
create_power_domain PD \
  -supply {primary} \
  -supply {backup}
```
(predefined)
(user-defined)

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

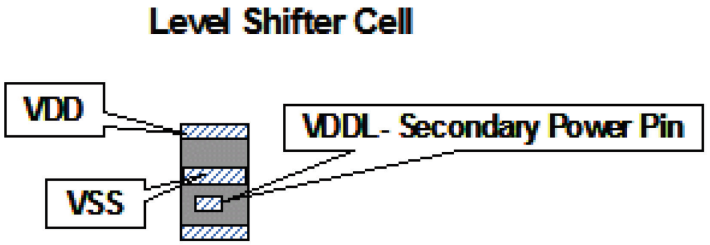
```
set_isolation ISO -domain PD \
  -isolation_supply_set PD.backup
```

How Logical Supply Networks Relate to Real World Connections

- The functions in a supply set will translate into real supply net connections.
  - Often a domain's 'primary' function will be implemented as the rails within a floorplan region, VDD and VSS in this example.
  - The other functions will be routed as secondary supplies and connect to special pins on the cell. The example here shows a levelshifter cell with a secondary pin called VDDL that connects to VDD2.
  - Secondary supplies may be primary supplies of other domains. Therefore the location of special cells may result in supplies from other domains to be available at the location where the cell is inserted.



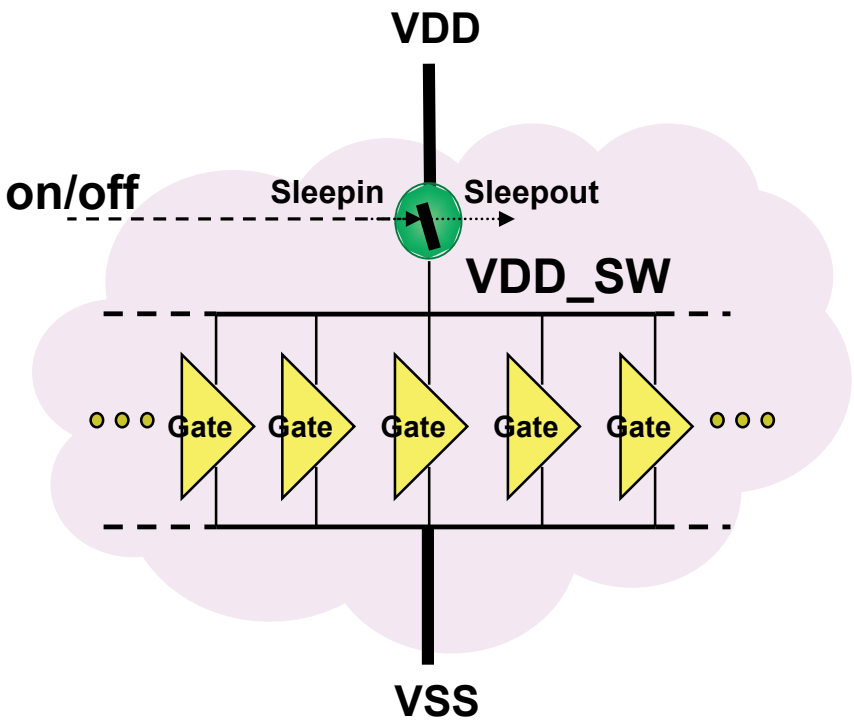

```
create_supply_set top_ss\
  -function {power VDD} \
  -function {ground VSS}

associate_supply_set top_ss\
  -handle PD.primary
```



Power Management Techniques

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

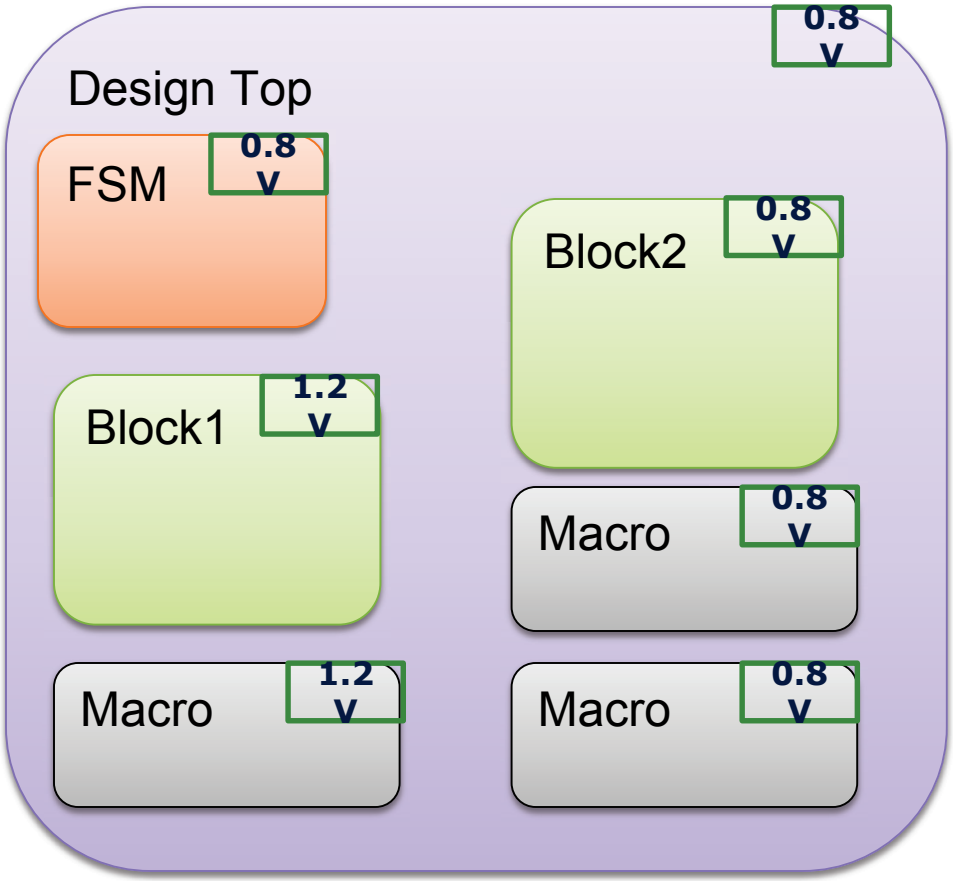
Power Gating



- Power reduction technique to save leakage power by shutting off, or powering down, unnecessary logic
- Enabled by power switch, or MTCMOS, cells
- Requires consideration of isolating and state retention.

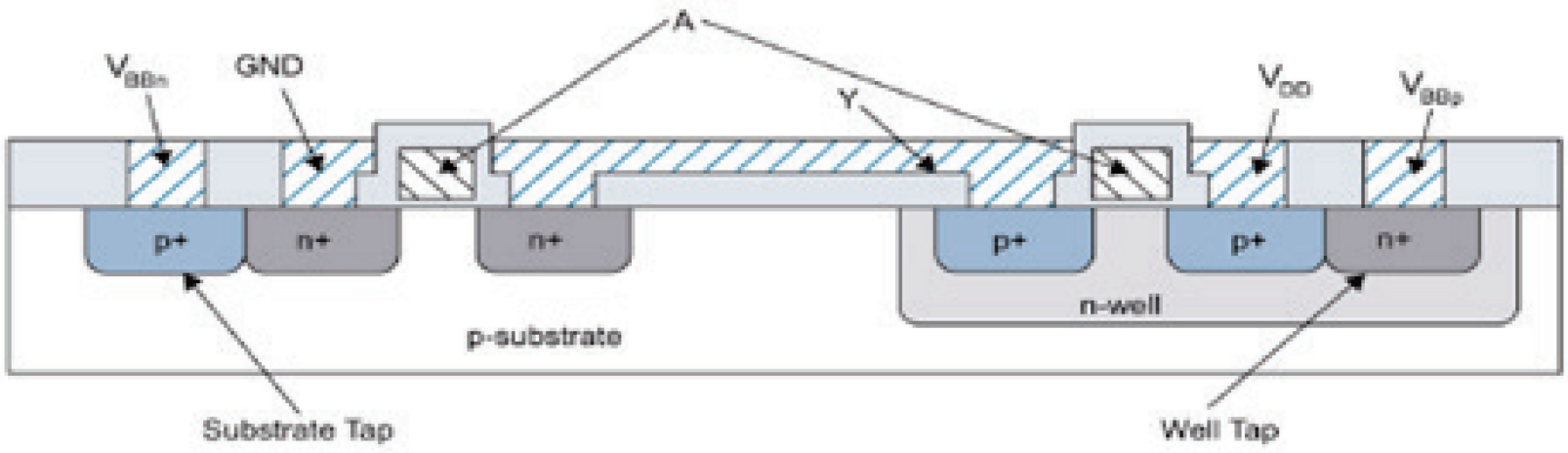
Multi-Voltage

- 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



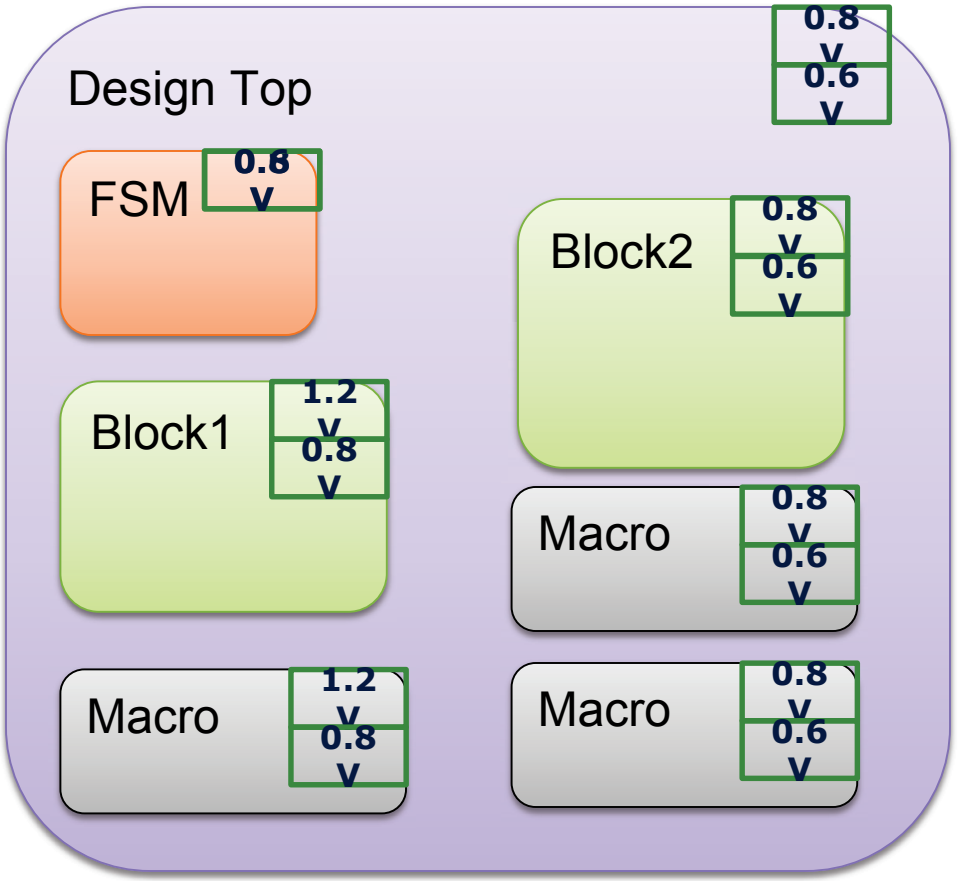
Bias Voltage

- 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

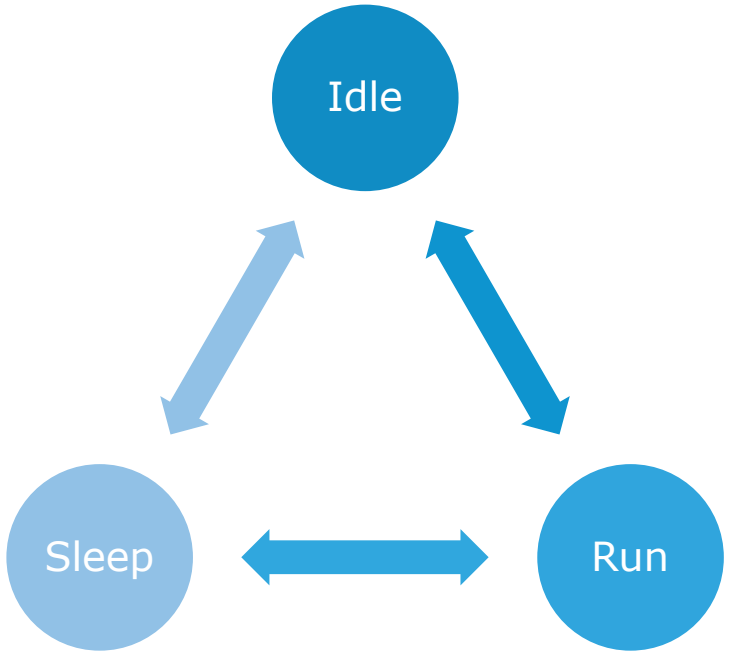


Power Management Architecture

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

Power States and Transitions

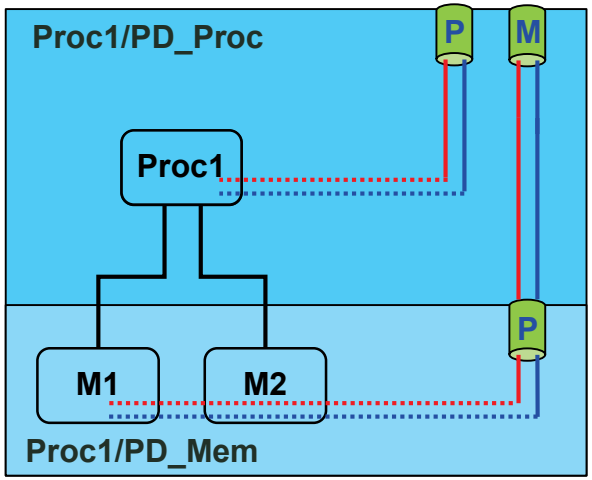
- **A design may have multiple modes of operation that affect the power supplies.**
- **Modelling these modes will allow tools to verify and implement the power intent. For example;**
  - *Simulation*: States can indicate that correct isolation has been specified.
  - *Implementation*: Correct levelshifting can be inserted.
  - *Verification*: Confirm that the design is still correct after optimization.



UPF Power States

- **UPF uses 'add_power_state' to define the states of supplies.**

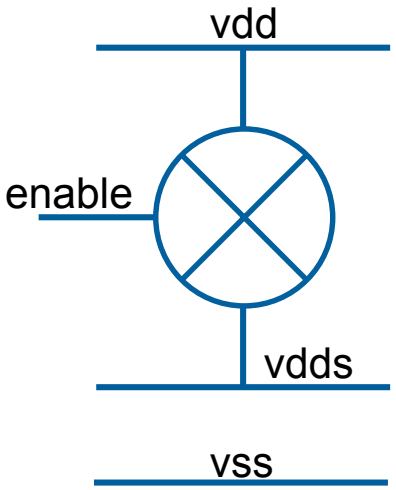
```
add_power_state PD_Mem \
  -state RUN {-logic_expr {primary ==
ON_08}} \
  -state OFF {-logic_expr {primary == OFF}}

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



PD_PROC	primary	memory	PD_MEM
Normal	ON_10	ON_08	RUN
Sleep	OFF	ON_08	RUN
Hibernate	OFF	OFF	OFF

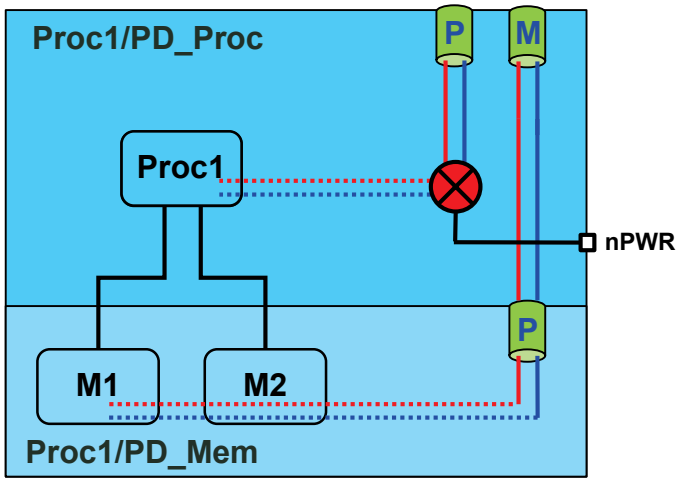
Supply Switching/Power Gating

- Supplies can be switched off to save power when they are not needed. This can be done off or on chip.
- On-chip switching can be implemented by number of methods including fine/course grain switch cells.
- UPF will allow a switch construct to be declared to represent the switching state of the supplies. The supplies, their states and controlling signals are all defined however how it is implemented is not specified.



UPF - Power Gating

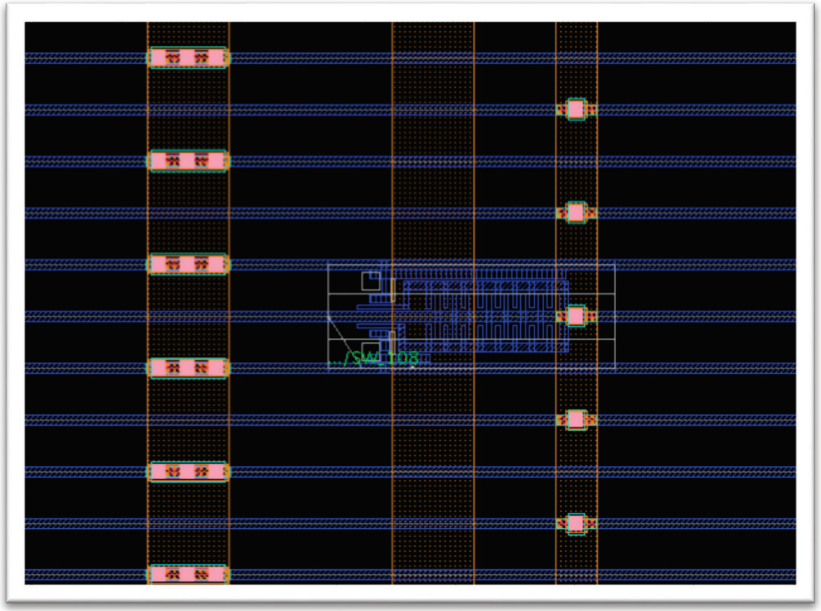
```
create_logic_port nPWR1 -direction in


create_power_switch SW -domain PD_Proc
    -input_supply_port    {sw_in    VDDSOC}
    -output_supply_port   {sw_out   VDDPROC1}
    -control_port         {sw_ctl   nPWR}
    -on_state   {on_state   sw_in {!sw_ctl}}
    -off_state {off_state         { sw_ctl}}
```

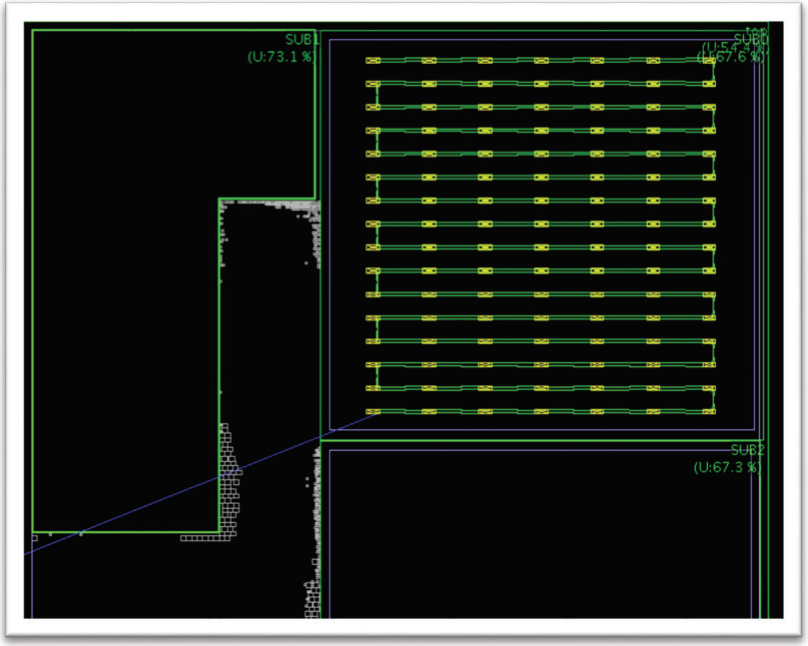


PD_PROC	primary	memory	PD_MEM
Normal	ON_10	ON_08	RUN
Sleep	OFF	ON_08	RUN
Hibernate	OFF	OFF	OFF

Coarse Grain Switch Implementation

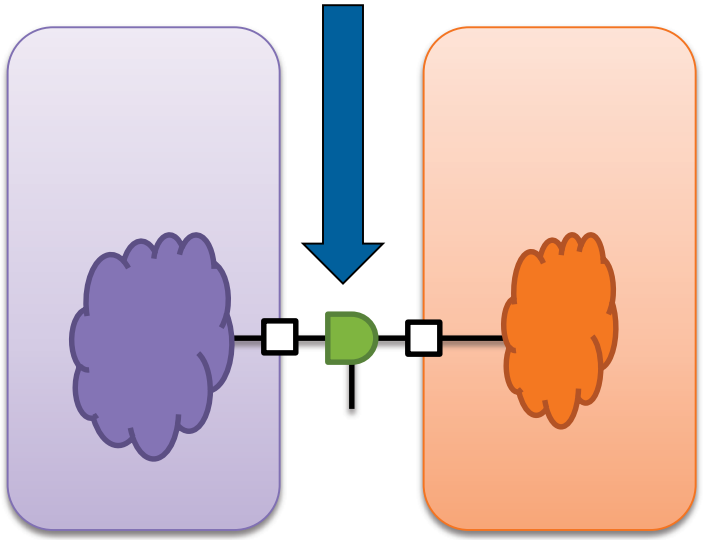
- A UPF switch may be implemented as an array of switches all connected together to switch as one.
- The switches power the standard cell rows and use power meshes to connect the secondary supply.





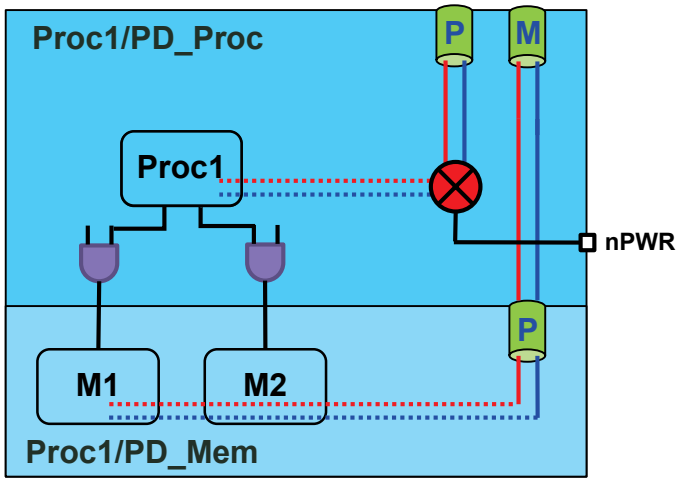
Isolation Cells

- 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



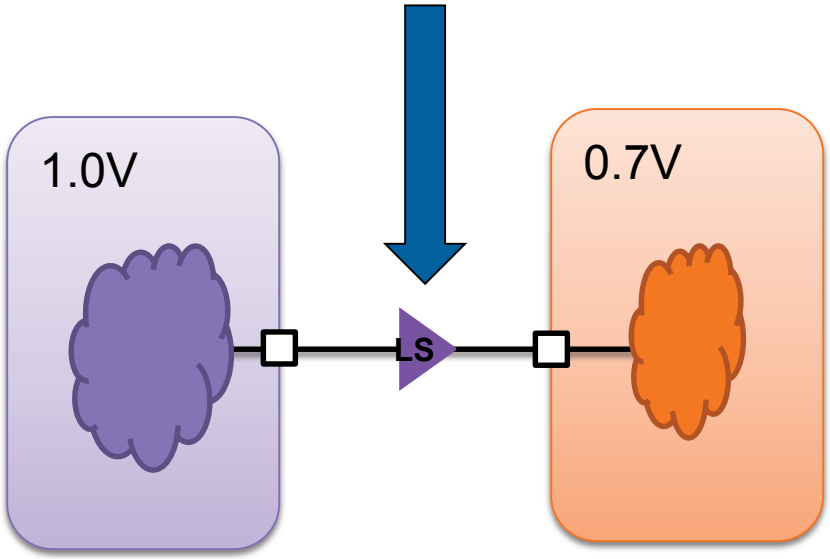
UPF Isolation Strategies

```
set_isolation ISO_Proc \
  -domain PD_Proc \
  -applies_to outputs \
  -clamp_value 0 \
  -isolation_signal mISO \
  -isolation_sense low \
  -location self

use_interface_cell ISOX1 \
  -domain PD_Mem \
  -strategy ISOMem \
  -lib_cells {TechISOX1}
```



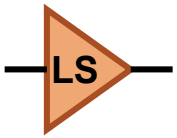
PD_PROC	primary	memory	PD_MEM
Normal	ON_10	ON_08	RUN
Sleep	OFF	ON_08	RUN
Hibernate	OFF	OFF	OFF

Level Shifters

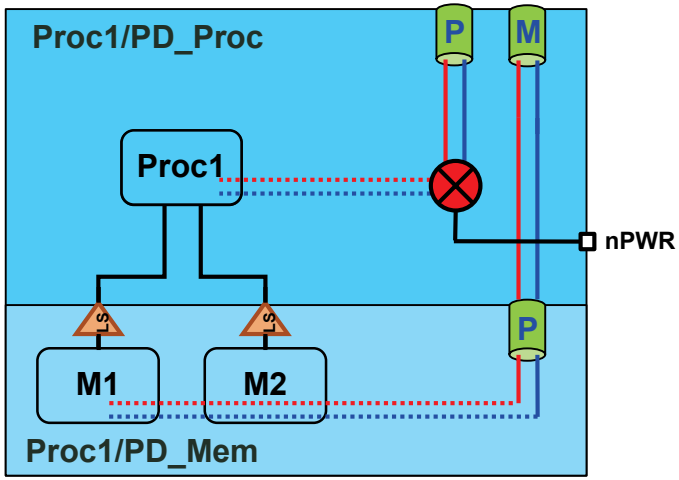


- Changes the voltage from one discrete value to another discrete value
  - A 1'b1 driven by 1.0V logic may be too much for 0.7V logic and likewise a 1'b1 from 0.7V logic may not translate into a 1'b1 for 1.0V logic
  - A level shifter changes a 0.7V 1'b1 to a 1.0V 1'b1 so you are propagating valid digital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

UPF Level Shifting Strategies



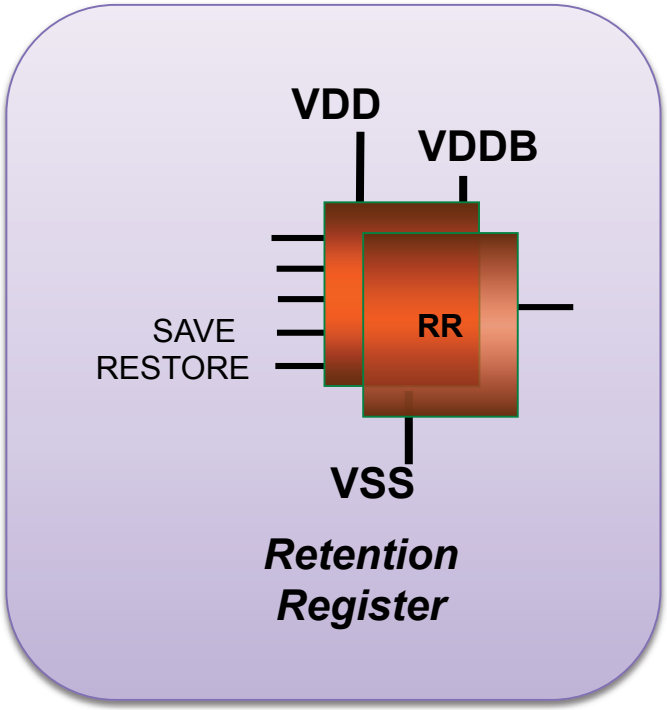

```
set_level_shifter LSmem \
  -domain PD_Mem \
  -applies_to outputs \
  -location self

use_interface_cell LSX2 \
  -domain PD_Mem \
  -strategy LSMem \
  -lib_cells {TechLSX2}
```



PD_PROC	primary	memory	PD_MEM
Normal	ON_10	ON_08	RUN
Sleep	OFF	ON_08	RUN
Hibernate	OFF	OFF	OFF

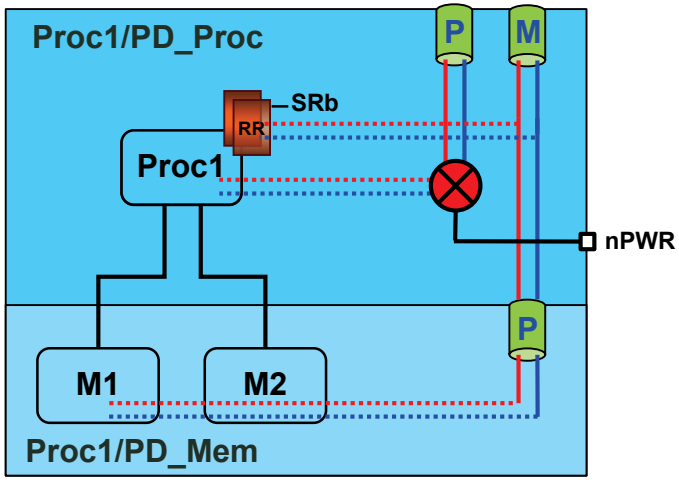
State Retention

- 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



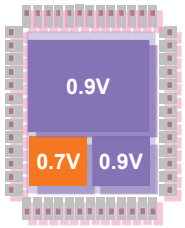
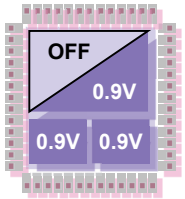
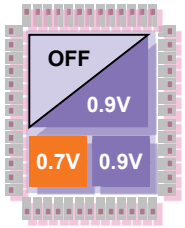
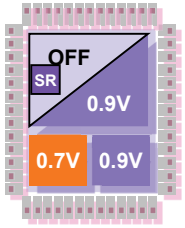
UPF Retention Strategies


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


map_retention_cell RET1 \
   -domain PD_Proc \
   -lib_cells {TechRRX4}
```



PD_PROC	primary	memory	PD_MEM
Normal	ON_10	ON_08	RUN
Sleep	OFF	ON_08	RUN
Hibernate	OFF	OFF	OFF

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	✓	✓	✓	✓	✓

UPF Semantics and Usage

Erich Marschner

Verification Architect

Mentor Graphics

A Deeper Look at UPF Power Intent

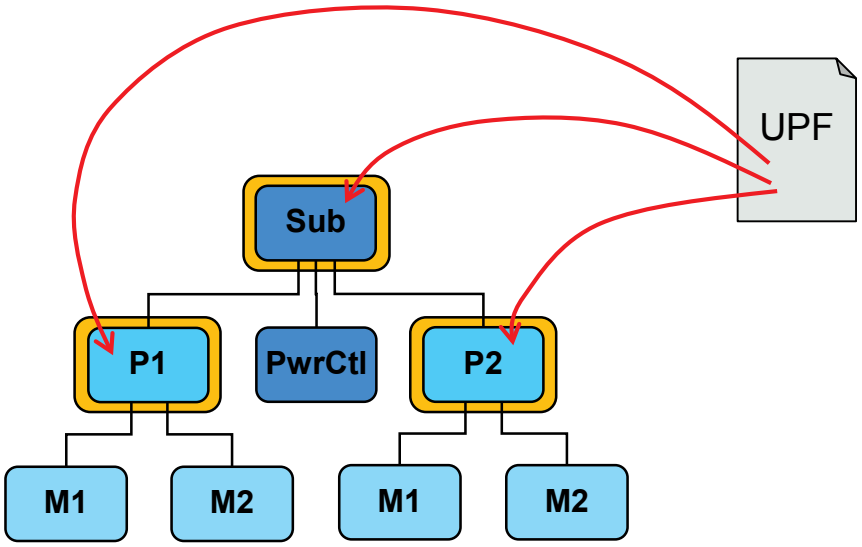
- Logic Hierarchy
- Power Domains
- Power Domain Supplies
- Supply Sets
- Supply Connections
- Power Related Attributes
- Power States and Transitions
- Power Domain State Retention
- Power Domain Interface Management
- Supply Network Construction
- Supply Equivalence

Logic Hierarchy

- **Design Hierarchy**
  - A hierarchical description in HDL
- **Logic Hierarchy**
  - An abstraction of the design hierarchy (instances only)
- **Scope**
  - An instance in the logic hierarchy
- **Design Top**
  - The topmost scope/instance in the logic hierarchy to which a given UPF file applies

Logic Hierarchy

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



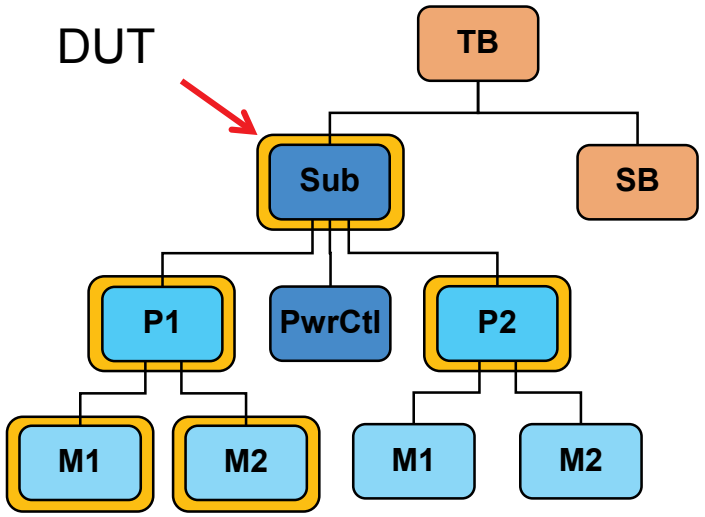
Navigation

- **set_design_top**

```
set_design_top TB/Sub
```

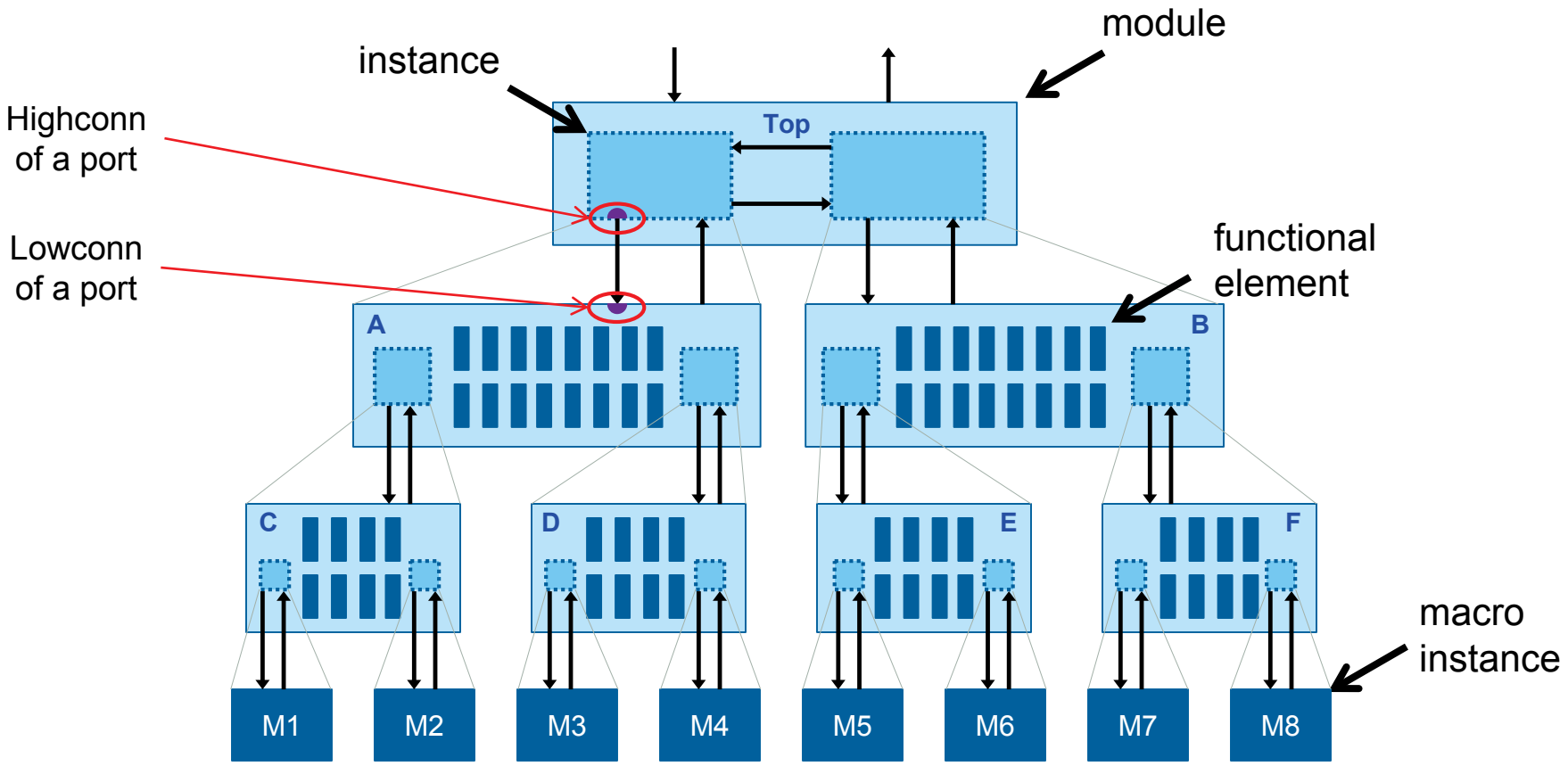
- **set_scope**

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



Note:
These are all instance names

Logic Hierarchy

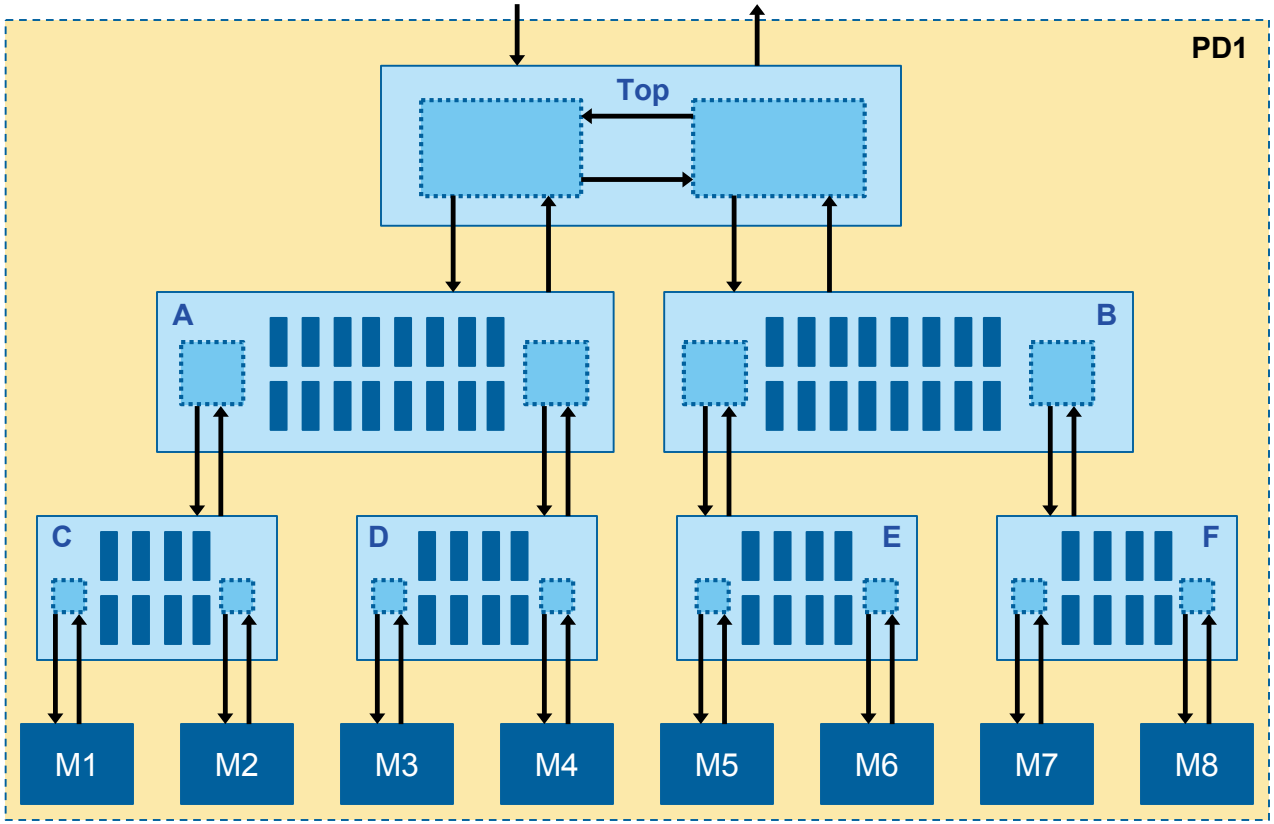


Power Domains

- **Partition the Logic Hierarchy**
  - Every instance must be in (the extent of) exactly one domain
- **Can be further partitioned**
  - A subtree of the design can be carved out as another domain
- **Unless declared "atomic"**
  - Atomic power domains cannot be further subdivided
- **Can be composed into larger domains**
  - If all subdomains have the same primary power supply
- **Have an upper and a lower boundary**
  - Boundaries represent a change in primary supply

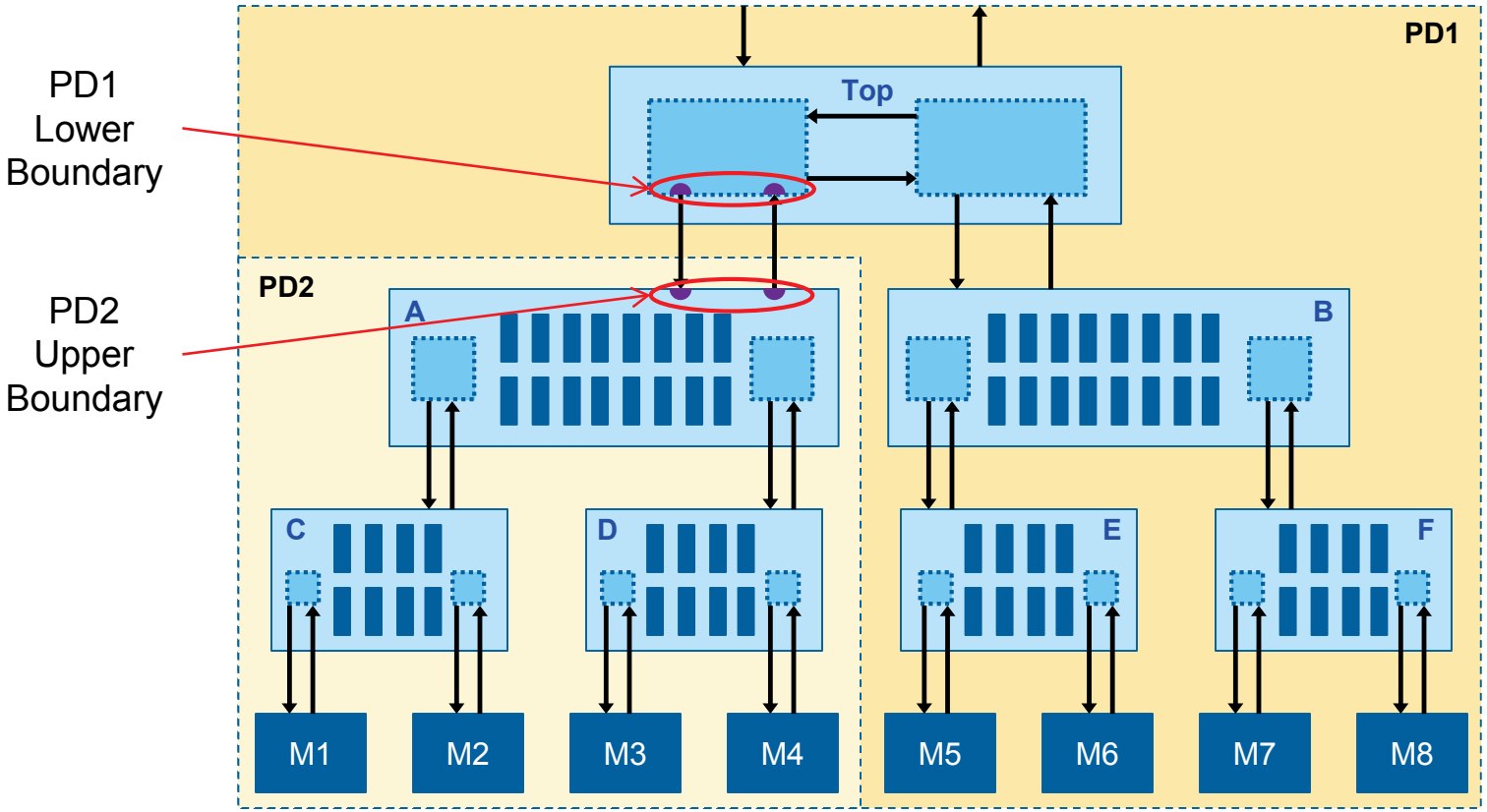
Partitioning the Logic Hierarchy - 1

```
create_power_domain PD1 -elements {.} …
```



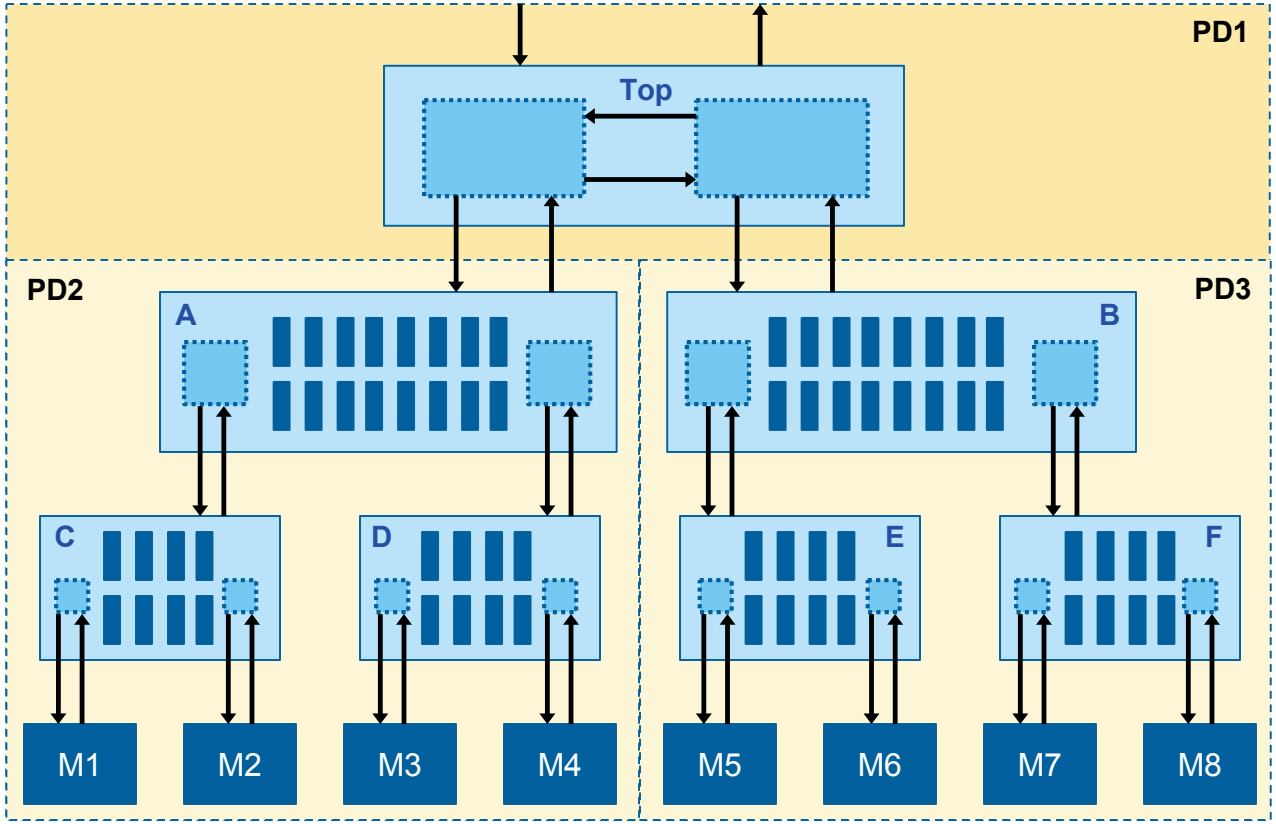
Partitioning the Logic Hierarchy - 2

```
create_power_domain PD2 -elements {A} …
```



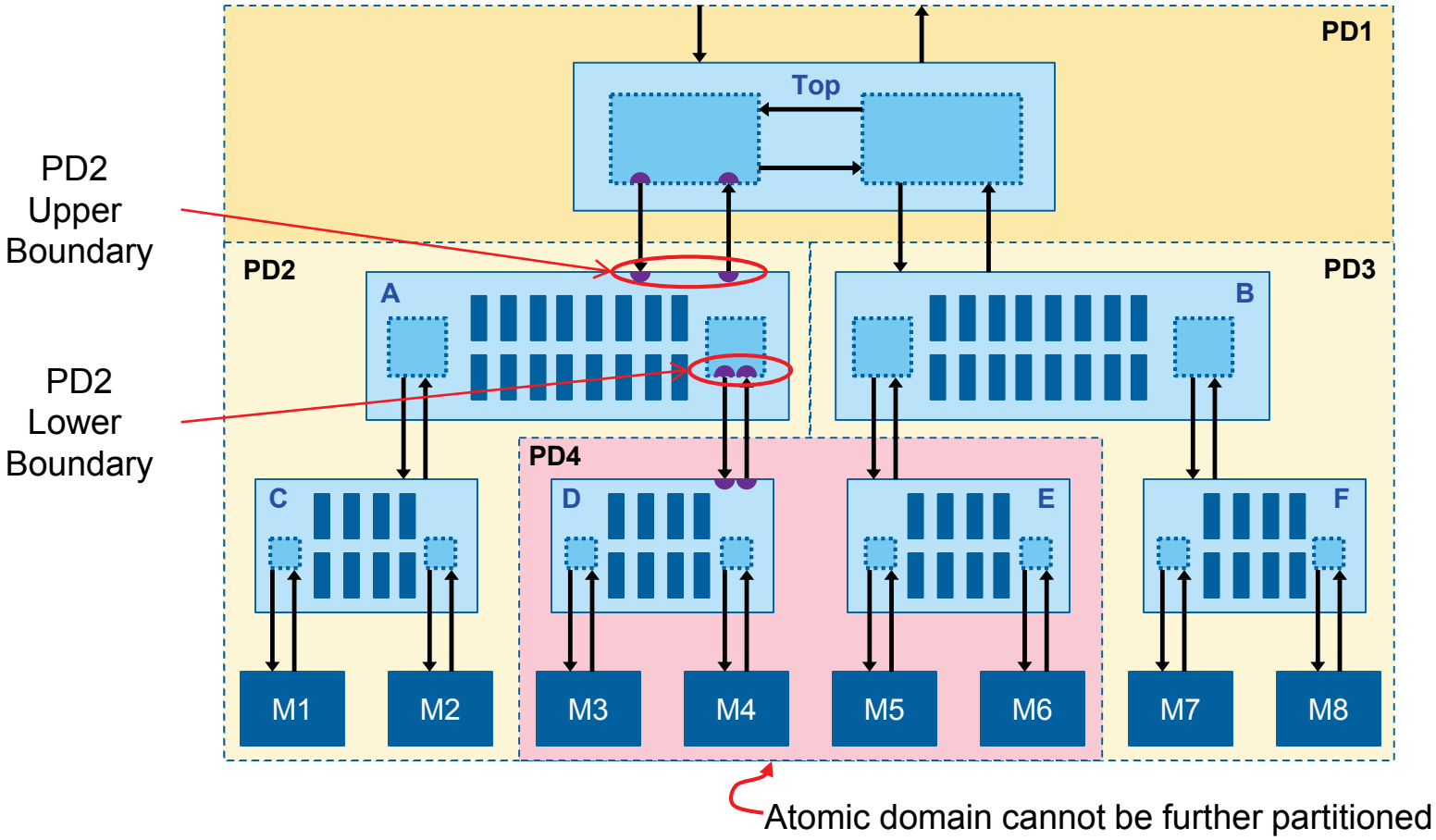
Partitioning the Logic Hierarchy - 3

```
create_power_domain PD3 -elements {B} …
```



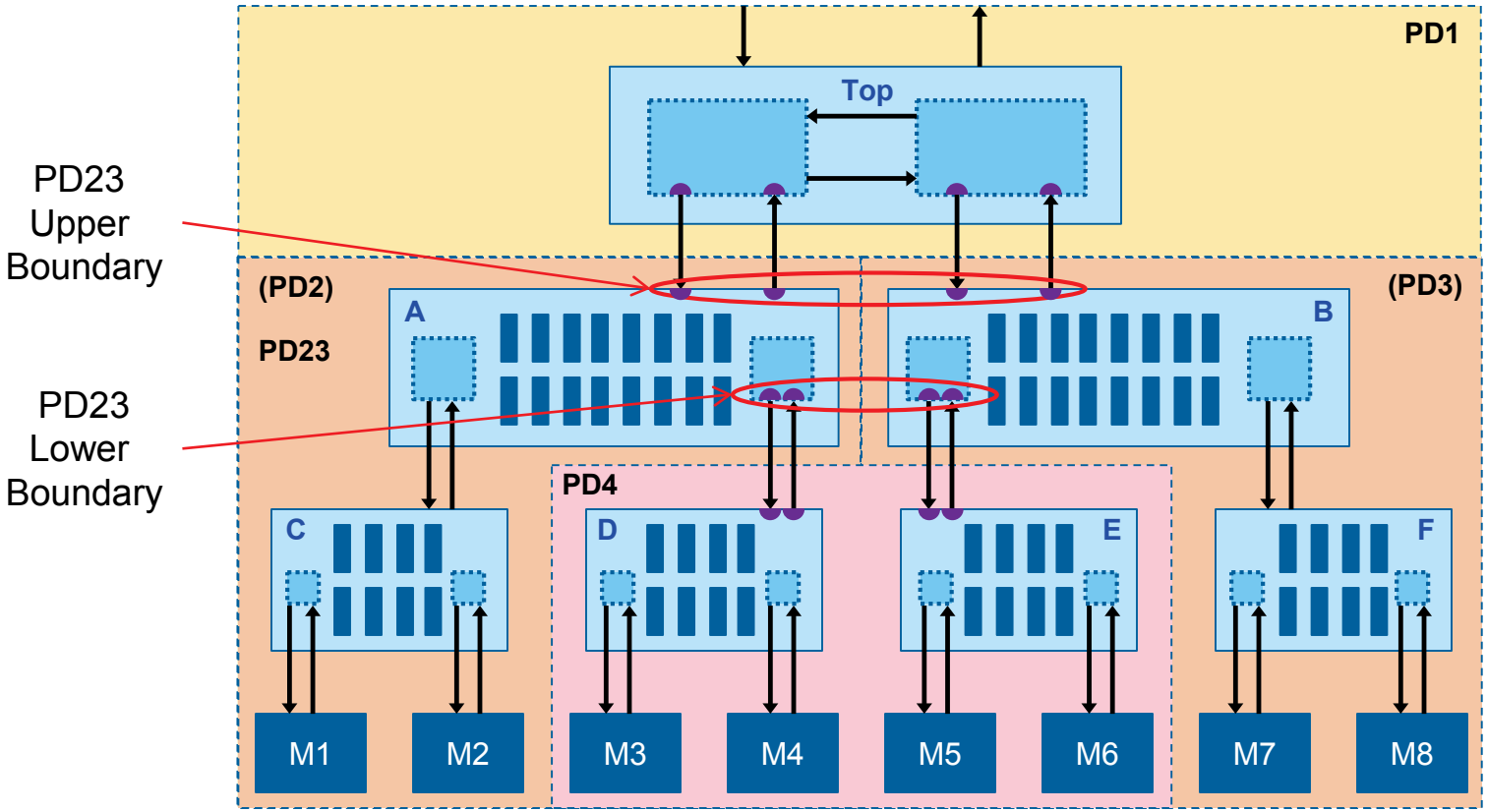
Partitioning the Logic Hierarchy - 4

```
create_power_domain PD4 -elements {A/D B/E} -atomic …
```



Partitioning the Logic Hierarchy - 5

```
create_composite_domain PD23 -subdomains {PD2 PD3} …
```



Power Domain Boundaries

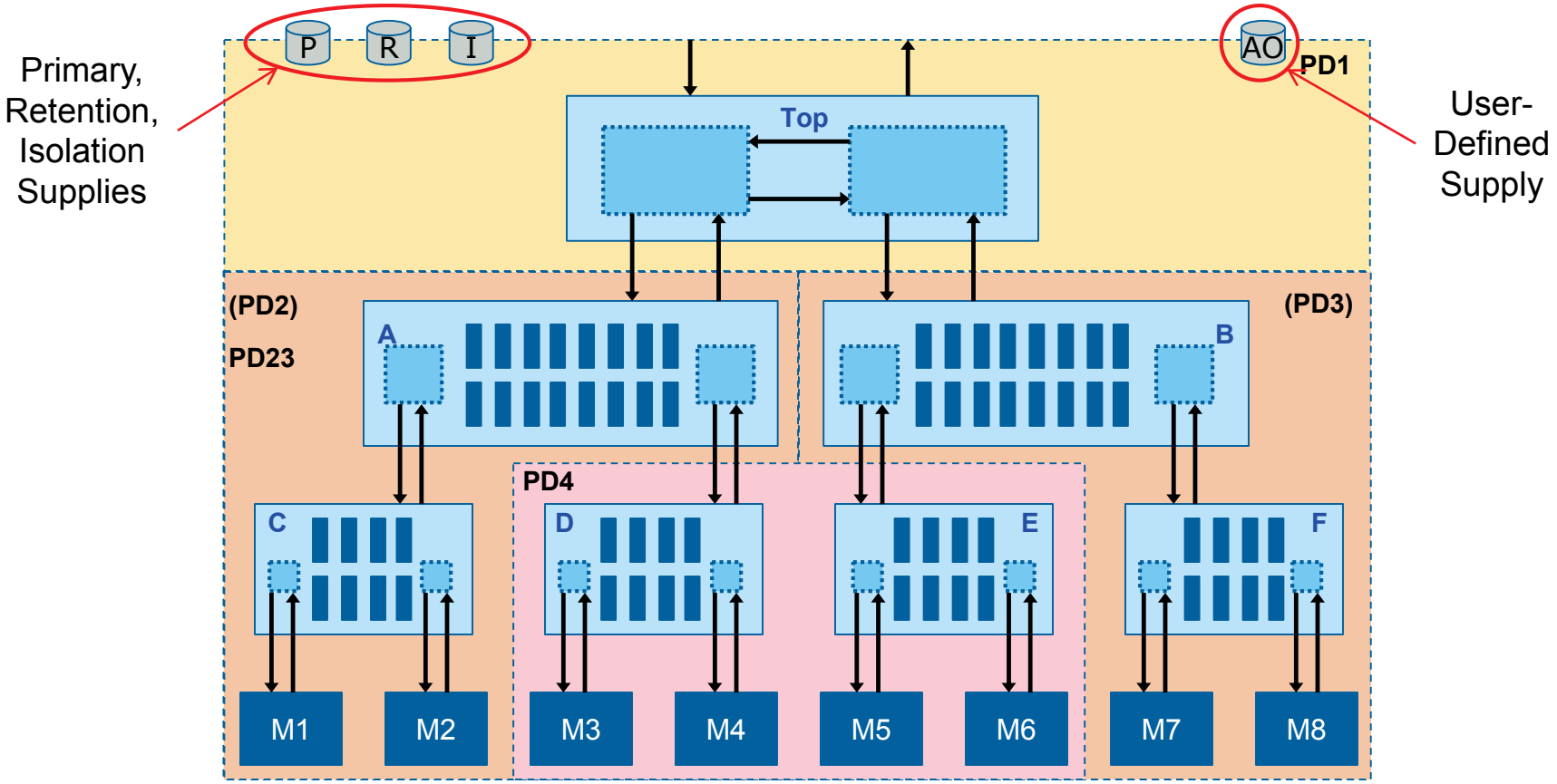
- **Define Domain Interfaces**
  - Isolation/Level shifting are only inserted at power domain boundaries
- **Upper Boundary**
  - Includes lowconn of declared ports of "top-level" instances
- **Lower Boundary**
  - Includes highconn of ports of instances in another domain
  - Includes macro instance ports with different supplies
- **Macro Instances**
  - May have multiple supplies
  - Each port may have a different supply

Domain Supplies

- **Primary supply**
  - Provides the main power, ground supplies for cells in the domain
  - Can also provide additional supplies (nwell, pwell, …)
- **Default retention supply**
  - Provides a default supply for saving the state of registers
- **Default isolation supply**
  - Provides a default supply for input or output isolation
- **Additional user-defined supplies**
  - Can be defined for particular needs (e.g., hard macros)
- **Available supplies**
  - Can be used by tools to power buffers used in implementation

Power Domain Supply Sets

```
create_power_domain PD1 -supply {AO} …
```

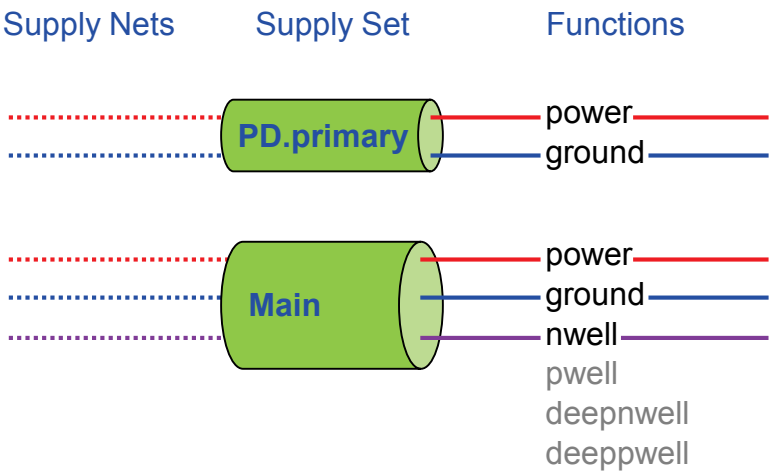


Supply Sets

- **Consists of a set of up to 6 supply "functions"**
  - power, ground, nwell, pwell, deepnwell, deeppwell
- **Represent a collection of supply nets**
  - One supply net per (required) function
- **Can be "global" or "local" to a power domain**
  - Power domains have a few predefined supply set "handles"
- **Can be associated with one another**
  - To model supply connections abstractly
- **Have power states with simstates**
  - Determine domain functionality in power aware simulation

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



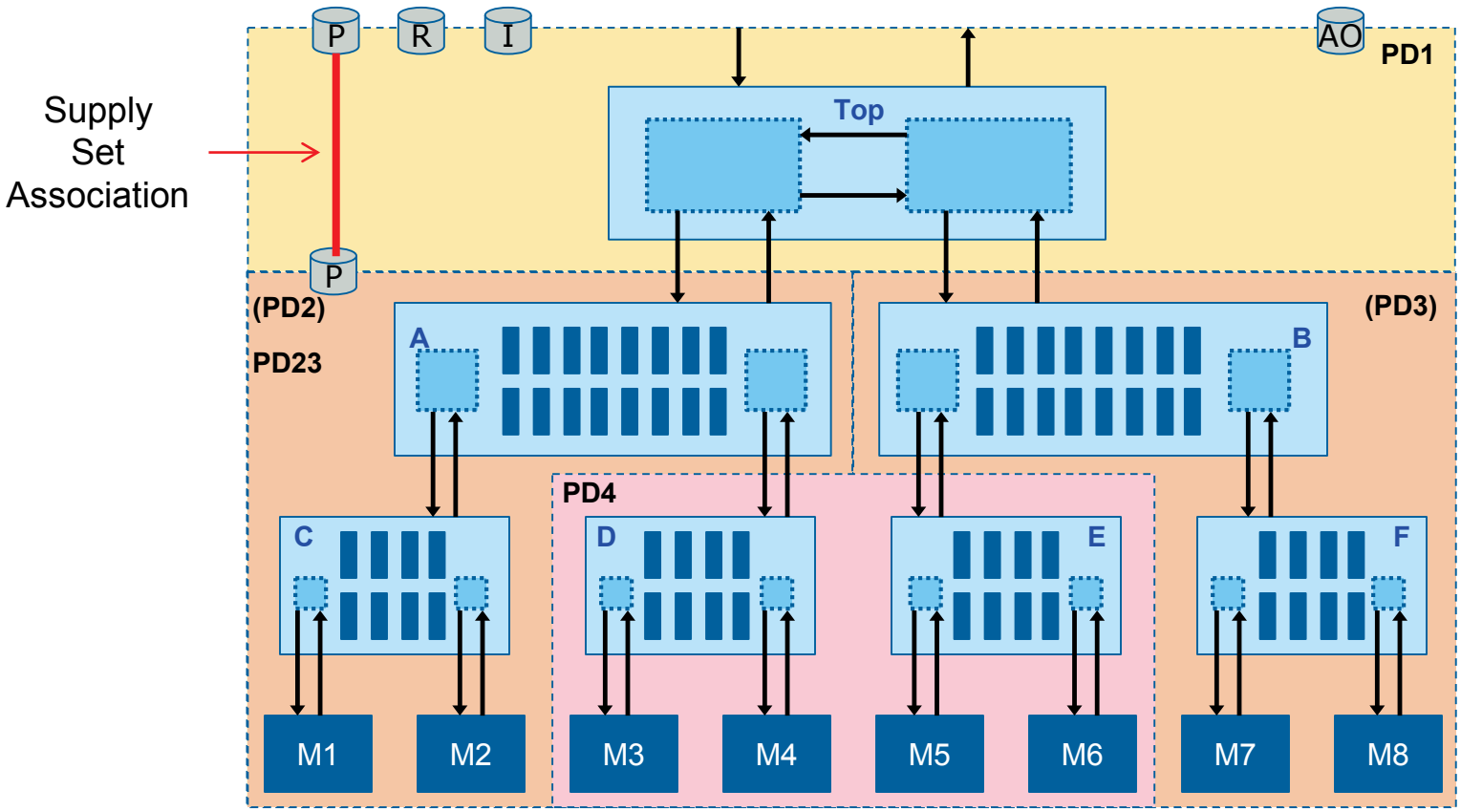

```
create_power_domain PD \
  -supply {primary} \
  -supply {backup}
```
(predefined)
(user-defined)

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

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

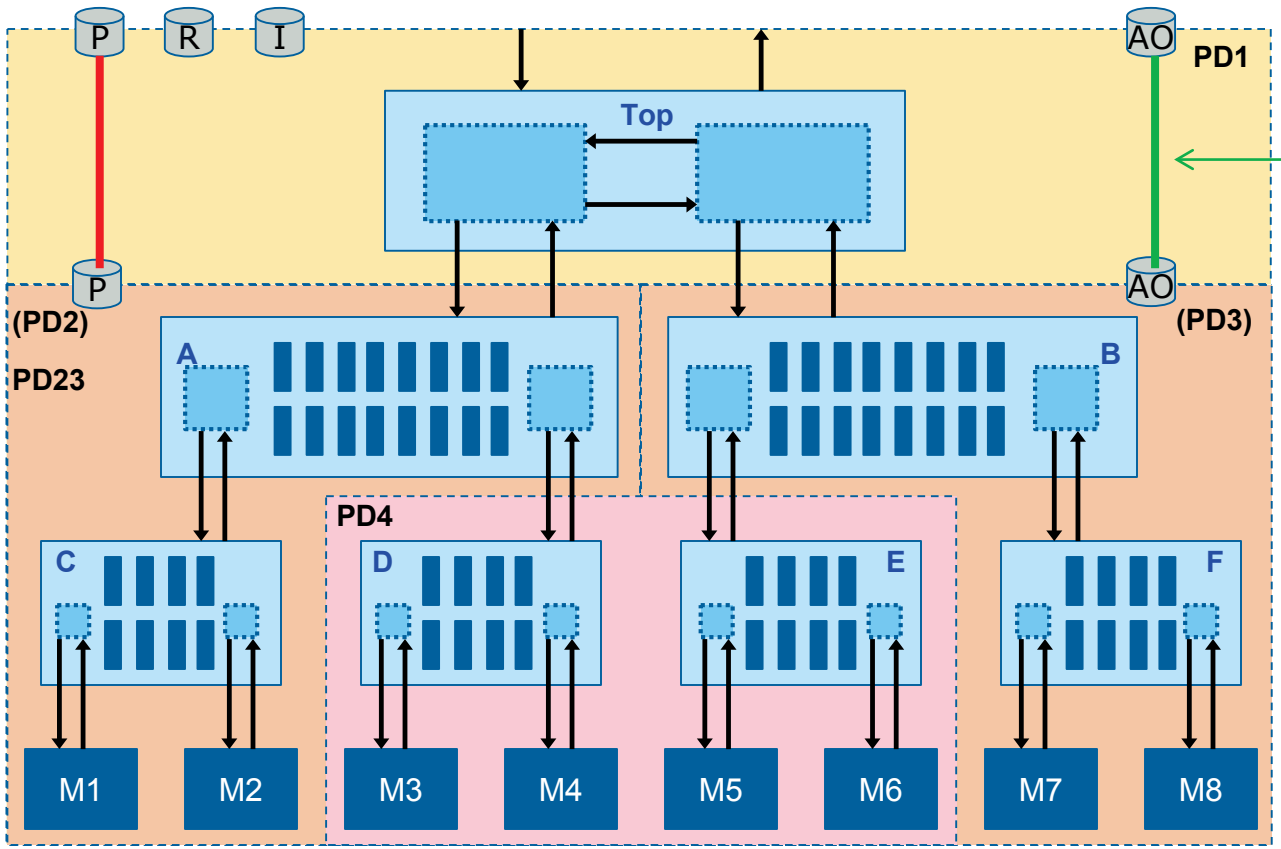
Associating Supply Sets 1

```
associate_supply_set PD1.primary -handle PD2.primary
```



Associating Supply Sets 2

```
associate_supply_set PD1.AO -handle PD3.AO
```



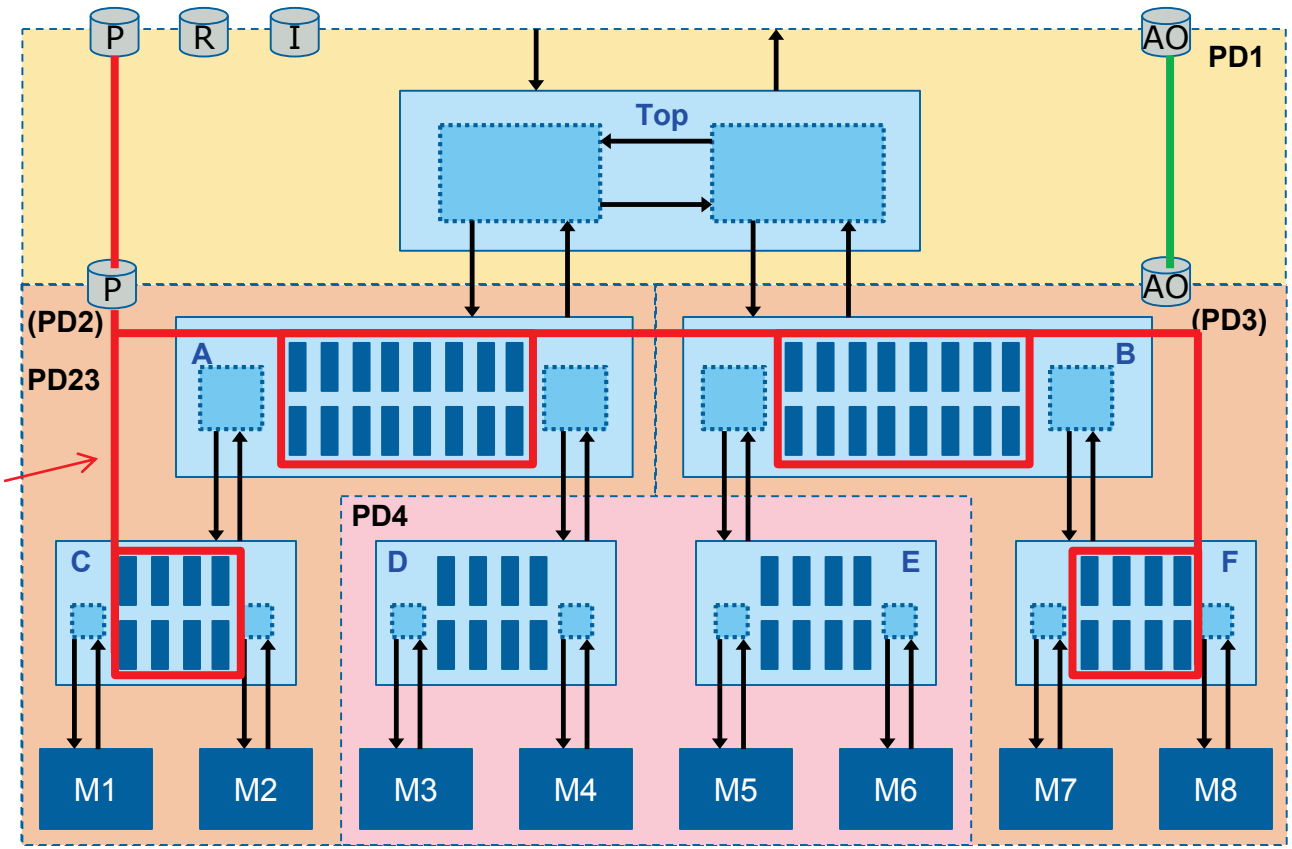
Supply Connections

- **Implicit connections**
  - Primary supply is implicitly connected to std cells
- **Automatic connections**
  - Supplies can be connected to cell pins based on pg_type
- **Explicit connections**
  - Supplies can be connected explicitly to a given pin
- **Precedence rules apply**
  - Explicit overrides Automatic overrides Implicit
- **Supply states determine cell behavior**
  - Cells function when supply is on,
  - Cells outputs are corrupted when supply is off

Implicit Supply Connections

...

Implicit Connections: power, ground only

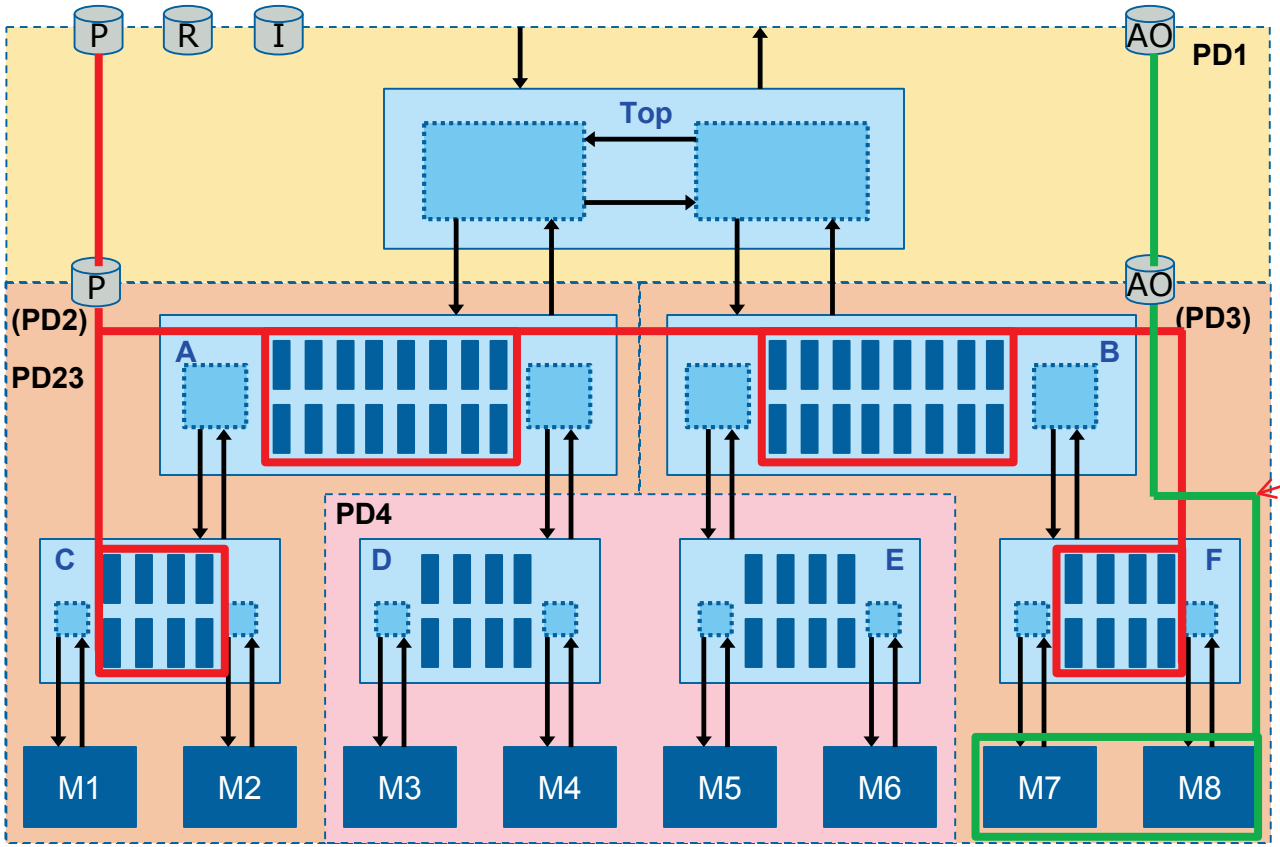


PG Types

- **Describe the usage of supply pins of cells, macros**
  - primary_power, primary_ground
  - backup_power, backup_ground
  - internal_power, internal_ground
  - nwell, pwell, deepnwell, deeppwell
- **Typically defined in Liberty library models**
  - primary power/ground are common to all
- **Can also be defined in HDL or UPF**
  - Using attributes …
- **Drive implicit and automatic supply connections**
  - Each function of a domain supply set maps to a pg_type

Automatic Supply Connections

```
connect_supply_set PD3.AO -elements {B/F/M7 B/F/M8}
```



PG Type-Driven Connections

- **Automatic connection**

```
connect_supply_set PD3.AO -elements {B/F/M7 B/F/M8} \
  -connect {power primary_power} \
  -connect {ground primary_ground}

connect_supply_set PD3.AO -elements {B/F/M7 B/F/M8} \
  -connect {power backup_power} \
  -connect {ground backup_ground}
```

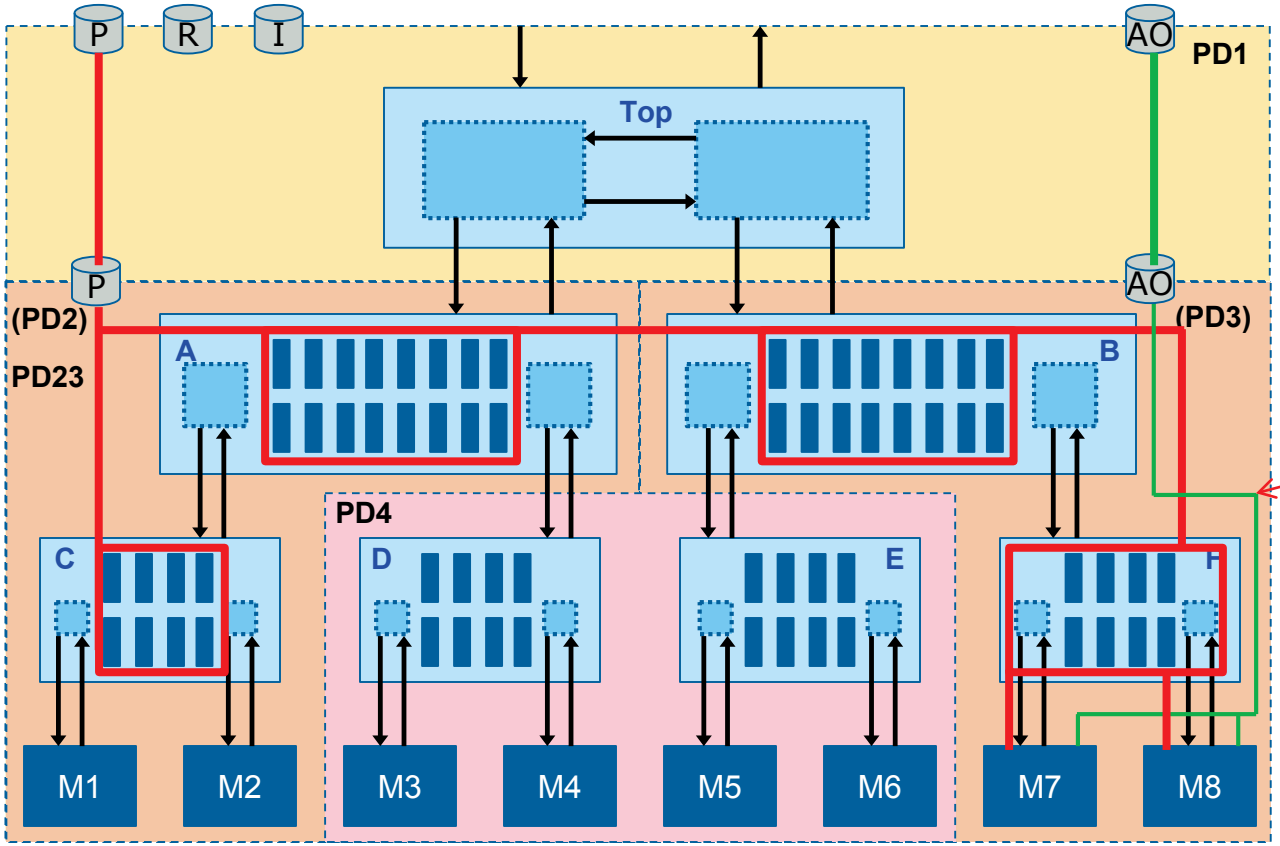
- **Implicit connection**

– Equivalent to

```
connect_supply_set PD2.primary -elements {.} \
  -connect {power primary_power} \
  -connect {ground primary_ground}
```

Explicit Supply Connections

```
connect_supply_net PD3.AO.power -ports {B/F/M7/VDDB …}
```



Power Attributes

- **Characteristics of a port or design element**
  - That relate to power intent or implementation
- **Defined in UPF, HDL, or Liberty**
  - Liberty and HDL attributes are imported into UPF
- **Used to identify power supplies for ports**
  - Related supplies for ports and cell pins
- **Used to specify constraints for IP usage**
  - Clamp value constraints for isolation of ports
- **Used to specify structure and behavior information**
  - Hierarchy leaf/macro cells, net connections, simstate use

Predefined UPF Attributes

- **Supply Attributes**
  - UPF_pg_type
  - UPF_related_power_port
  - UPF_related_ground_port
  - UPF_related_bias_ports
  - UPF_driver_supply
  - UPF_receiver_supply
- **Isolation Attributes**
  - UPF_clamp_value
  - UPF_sink_off_clamp_value
  - UPF_source_off_clamp_value
- **Structural Attributes**
  - UPF_is_leaf_cell
  - UPF_is_macro_cell
  - UPF_feedthrough
  - UPF_unconnected
- **Behavioral Attributes**
  - UPF_retention
  - UPF_simstate_behavior

Attribute Definitions

UPF

```
set_port_attributes -ports Out1 \
  -attribute \
    {UPF_related_power_port "VDD"

set_port_attributes -ports Out1 \
  -attribute \
    {UPF_related_ground_port "VSS"


set_port_attributes -ports Out1 \
  -related_power_port "VDD" \
  -related_ground_port "VSS"
```

HDL

SystemVerilog or Verilog-2005

```
(* UPF_related_power_port = "VDD",
   UPF_related_ground_port = "VSS" *)

  output Out1;
```

VHDL

```
attribute UPF_related_power_port of
  Out1: signal is "VDD";

attribute UPF_related_ground_port of
  Out1: signal is "VSS";
```

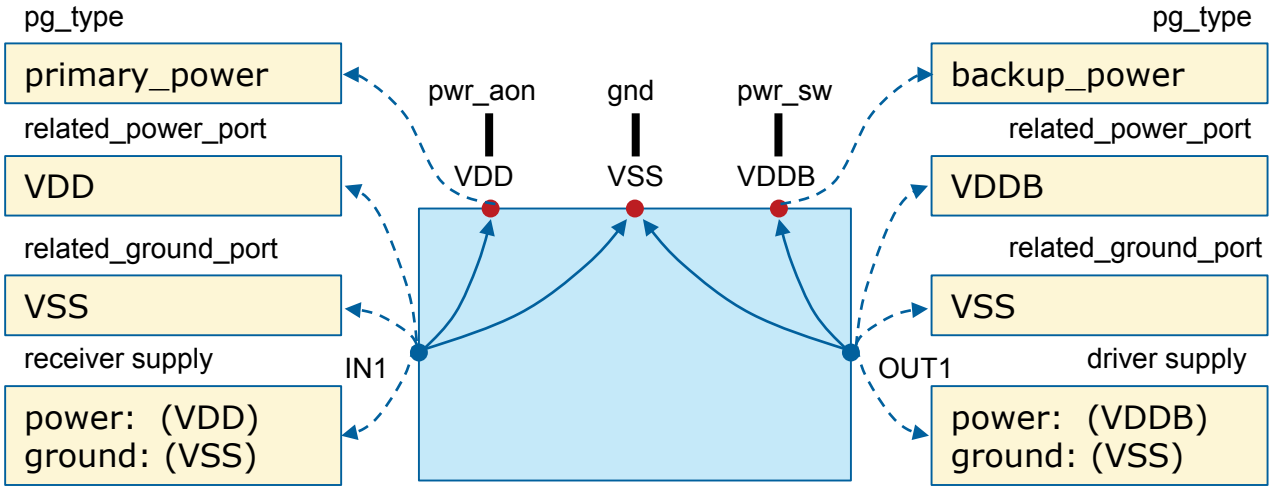
Liberty

- related_power_pin, related_ground_pin
- pg_type, related_bias_pins, is_macro_cell, etc.

UPF Attribute Usage

- **Supply Attributes**
  - UPF_pg_type
  - UPF_related_power_port
  - UPF_related_ground_port
  - UPF_related_bias_ports
  - UPF_driver_supply
  - UPF_receiver_supply

- **Used to specify**
  - cell/macro supply port types
  - logic port related supplies
  - primary IO port related supplies
  - driver/receiver supply sets
    - can be defined only in UPF
    - no supply set data type in HDL or Liberty



UPF Attribute Usage

- **Supply Attributes**
  - UPF_pg_type
  - UPF_related_power_port
  - UPF_related_ground_port
  - UPF_related_bias_ports
  - UPF_driver_supply
  - UPF_receiver_supply

- **Used to specify**
  - cell/macro supply port types
  - logic port related supplies
  - primary IO port related supplies
  - driver/receiver supply sets
    - can be defined only in UPF
    - no supply set data type in HDL or Liberty

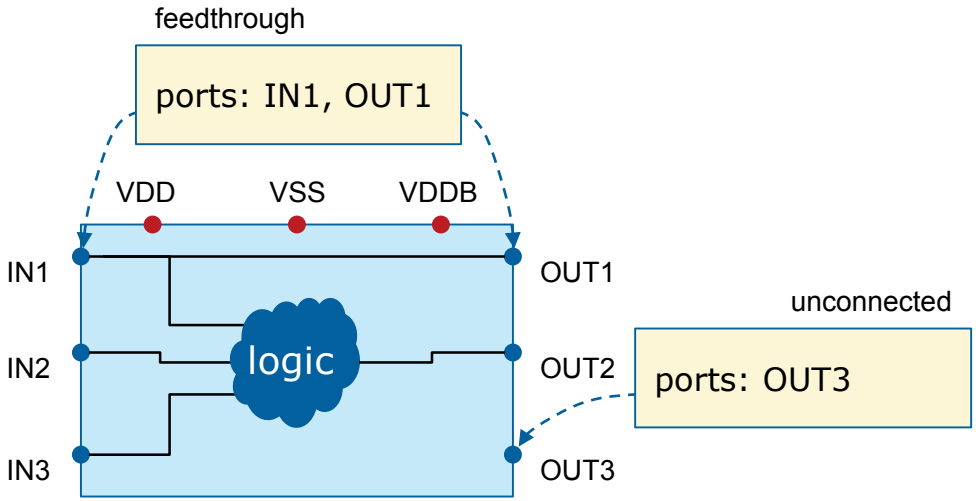
- **Isolation Attributes**
  - UPF_clamp_value
  - UPF_sink_off_clamp_value
  - UPF_source_off_clamp_value

- **Used to specify**
  - clamp value requirements in case source is powered off when sink is powered on
    - used to define power constraints for IP blocks

UPF Attribute Usage

- **Used to identify**
  - leaf cells in the hierarchy
  - macro cells in the hierarchy
  - feedthrough paths through a macro cell
  - unconnected macro ports

- **Structural Attributes**
  - UPF_is_leaf_cell
  - UPF_is_macro_cell
  - UPF_feedthrough
  - UPF_unconnected



UPF Attribute Usage

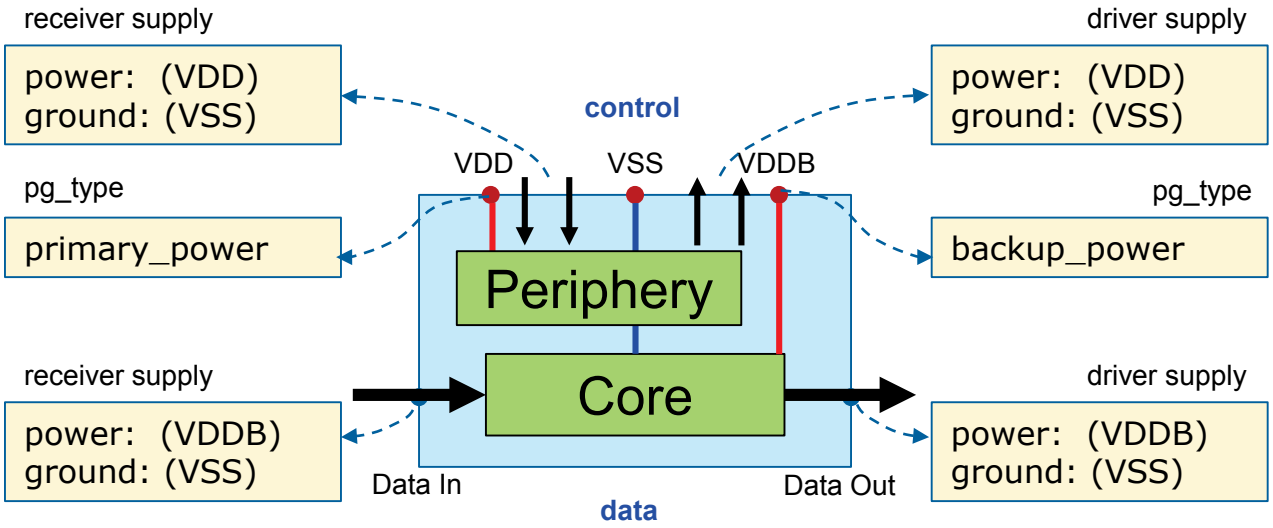
- **Used to identify**
  - leaf cells in the hierarchy
  - macro cells in the hierarchy
  - feedthrough paths through a macro cell
  - unconnected macro ports

- **Structural Attributes**
  - UPF_is_leaf_cell
  - UPF_is_macro_cell
  - UPF_feedthrough
  - UPF_unconnected

- **Used to define**
  - whether state retention is required for a given element
  - whether simstates determine power aware behavior (i.e., corruption)

- **Behavioral Attributes**
  - UPF_retention
  - UPF_simstate_behavior

Hard Macro Supplies



- **Modeled with Attributes**
  - Attributes of cell pins:
    - PG type attributes
    - Related supply attributes
    - In UPF, HDL, or Liberty
  - Imply anonymous supply sets

In a memory cell with separate supplies for peripheral logic and memory core, different ports will have different driver supplies or receiver supplies.

Supply Power States

- **Defined on supply sets**
  - In particular, power domain primary supply
- **Represent how cells behave in various situations**
  - When / whether cell outputs are corrupted
- **Defined by a logic expression**
  - State holds when logic expression is TRUE
- **Also may include a supply expression**
  - Defines the legal values of supply set fns when in that state
- **Also includes a simstate**
  - Simstate defines precise simulation semantics in this state
- **Not necessarily mutually exclusive!**

Supply Set Power State Definition

- **Simple**

```
add_power_state PD1.primary -supply \
  -state {ON  -logic_expr {PwrOn}  -simstate NORMAL} \
  -state {OFF -logic_expr {!PwrOn} -simstate CORRUPT}
```

- **More Complex**

```
add_power_state PD1.primary -supply \
  -state {RUN -logic_expr {PwrOn && !Sleep && Mains} \
    -simstate NORMAL} \
  -state {LOW -logic_expr {PwrOn && !Sleep && Battery} \
    -simstate NORMAL} \
  -state {SLP -logic_expr {PwrOn && Sleep} \
    -simstate CORRUPT_ON_CHANGE} \
  -state {OFF -logic_expr {!PwrOn} \
    -simstate CORRUPT}
```

Simstates - Precedence and Meaning

lower

- **NORMAL**
- **CORRUPT_STATE_ON_CHANGE**
- **CORRUPT_STATE_ON_ACTIVITY**
- **CORRUPT_ON_CHANGE**
- **CORRUPT_ON_ACTIVITY**
- **CORRUPT**

higher

- **NORMAL**
  - Combinational logic functions normally
  - Sequential logic functions normally
  - Both operate with characterized timing
- **CORRUPT_STATE_ON_CHANGE**
  - Combinational logic functions normally
  - Sequential state/outputs maintained as long as outputs are stable
- **CORRUPT_STATE_ON_ACTIVITY**
  - Combinational logic functions normally
  - Sequential state/outputs maintained as long as inputs are stable
- **CORRUPT_ON_CHANGE**
  - Combinational outputs maintained as long as outputs are stable
  - Sequential state/outputs corrupted
- **CORRUPT_ON_ACTIVITY**
  - Combinational outputs maintained as long as inputs are stable
  - Sequential state/outputs corrupted
- **CORRUPT**
  - Combinational outputs corrupted
  - Sequential state/outputs corrupted

Domain Power States

- **Defined on power domains**
  - In particular, power domains representing an IP block
- **Represent aggregate state of supplies, subdomains**
  - Abstract functional/power modes of a component
- **Defined by a logic expression (like supply set states)**
  - Typically refers to power states of other objects
- **Does NOT include a supply expression**
  - Supply expressions are only for supply set power states
- **Does NOT include a simstate**
  - Simstates are only for supply set power states
- **Not necessarily mutually exclusive!**

Domain Power State Definition

Examples

```
add_power_state PD_TOP -domain \
  -state {Normal \
    -logic_expr \
      {primary == ON && \
       backup  == ON && \
       PD_mem  == UP} } \
 -state {Sleep \
    -logic_expr \
      {primary == OFF && \
       backup  == ON && \
       PD_mem  == UP} } \
 -state {Off\
    -logic_expr \
      {primary == OFF && \
       backup  == OFF && \
       PD_Mem  == DOWN} }
```

Examples

```
add_power_state PD_Mem -domain \
  -state {UP \
    -logic_expr {primary == ON}} \
  -state {RET \
    -logic_expr {retention == ON}} \
  -state {DOWN \
    -logic_expr {retention == OFF}}
```

PD_TOP	.primary	.backup	PD_MEM
Normal	ON	ON	UP
Sleep	OFF	ON	RET
Off	OFF	OFF	DOWN

Power Management Strategies

- **Retention strategies**
  - Identify registers to retain, controls/conditions, and supplies
  - Must satisfy any retention constraints (clamp value attributes)
- **Repeater strategies**
  - Identify ports to be buffered and their supplies
  - Input and output ports can be buffered
- **Isolation strategies**
  - Define how to isolate ports where required - control, supplies
  - Actual isolation insertion is driven by source/sink power states
- **Level shifter strategies**
  - Define how to level-shift ports where required - supplies
  - Actual level shifter insertion is driven by threshold analysis

Retention Strategies

Balloon Latch

```
set_retention BL -domain PD1 \
  -elements {…} \
  -save_signal … \
  -restore_signal … \
  -retention_supply …
```

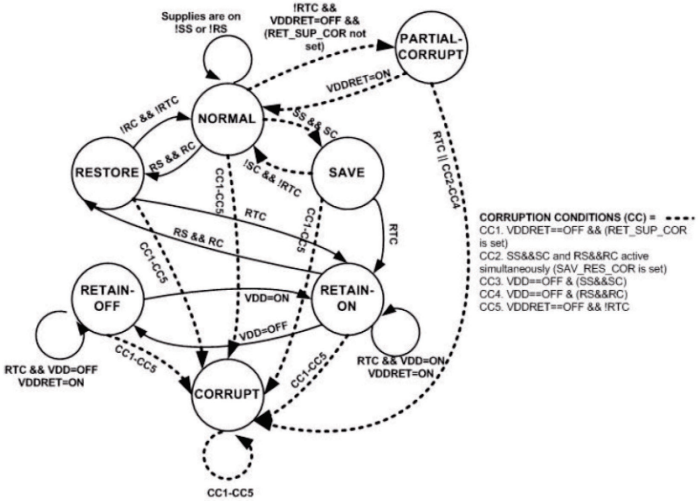


Figure 1—Retention state transition diagram for balloon-style retention

Live Slave Latch

```
set_retention LSL -domain PD1 \
  -elements {…} \
  -retention_condition … \
  -retention_supply …
```

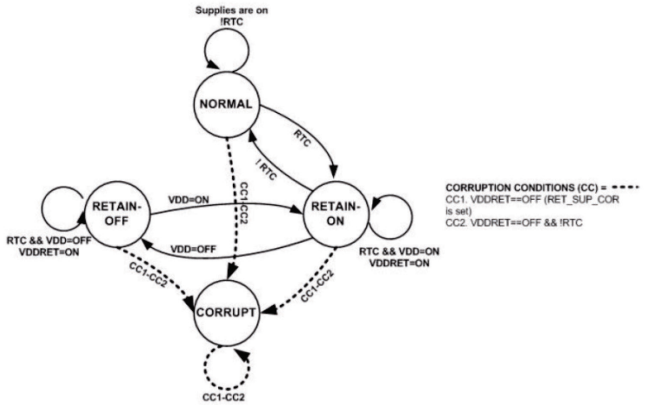


Figure 2—Retention state transition diagram for master/slave-alive style retention

Isolation Strategies

- **Specifying Ports**
  - Using `-elements` and `-exclude_elements`
  - Using filters: `-applies_to`, `-diff_supply_only`, `-sink`, `-source`
- **Precedence Rules**
  - More specific rules take precedence over more generic rules
- **Specifying Location**
  - Using locations `self`, `parent`, `other`, and `fanout`
- **Handling Fanout to Different Domains**
  - Using `-sink` to isolate different paths
- **Isolation Supplies and Cells**
  - Location affects default isolation supply and usable cell types

Specifying Ports

- **Elements list includes ports**

  `-elements { <port name> }`

  `-elements { <instance name> }`

  `-elements { . }`

  [if no -elements list, default is all ports of domain]

- **Exclude elements list excludes ports**

  `-exclude_elements { … }`

- **Filters further limit the set of ports**

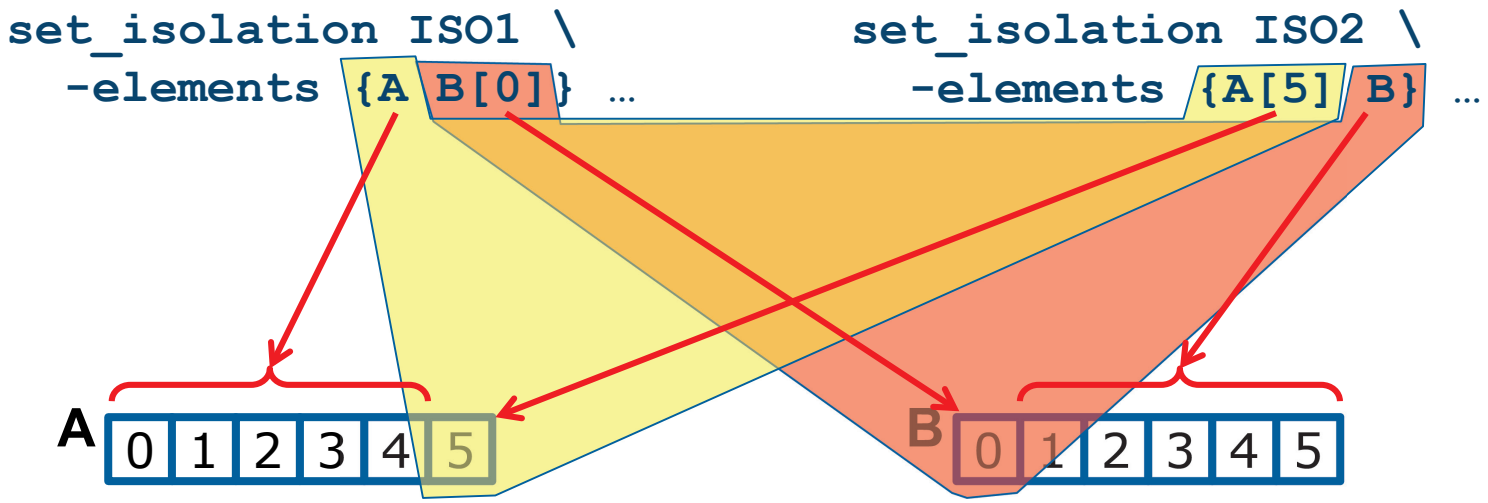
  `-applies_to <inputs | outputs | both>`

  `-source <domain name> | <supply set name>`

  `-sink <domain name> | <supply set name>`

  `-diff_supply_only`

What Happens if Multiple Strategies?



Precedence Rules for Strategies

lower

- Strategy for all ports of a specified power domain

```
set_isolation ISO1 -domain PD \
  …
```

- Strategy for all ports of a specified power domain with a given direction

```
set_isolation ISO2 -domain PD \
  -applies_to inputs …
```

- Strategy for all ports of an instance specified explicitly by name

```
set_isolation ISO3 -domain PD \
  -elements {i1} …
```

- Strategy for a whole port specified explicitly by name

```
set_isolation ISO4 -domain PD \
  -elements {i1/a i1/b} …
```

- Strategy for part of a multi-bit port specified explicitly by name

```
set_isolation ISO5 -domain PD \
  -elements {i1/a[3] i1/b[7]} …
```

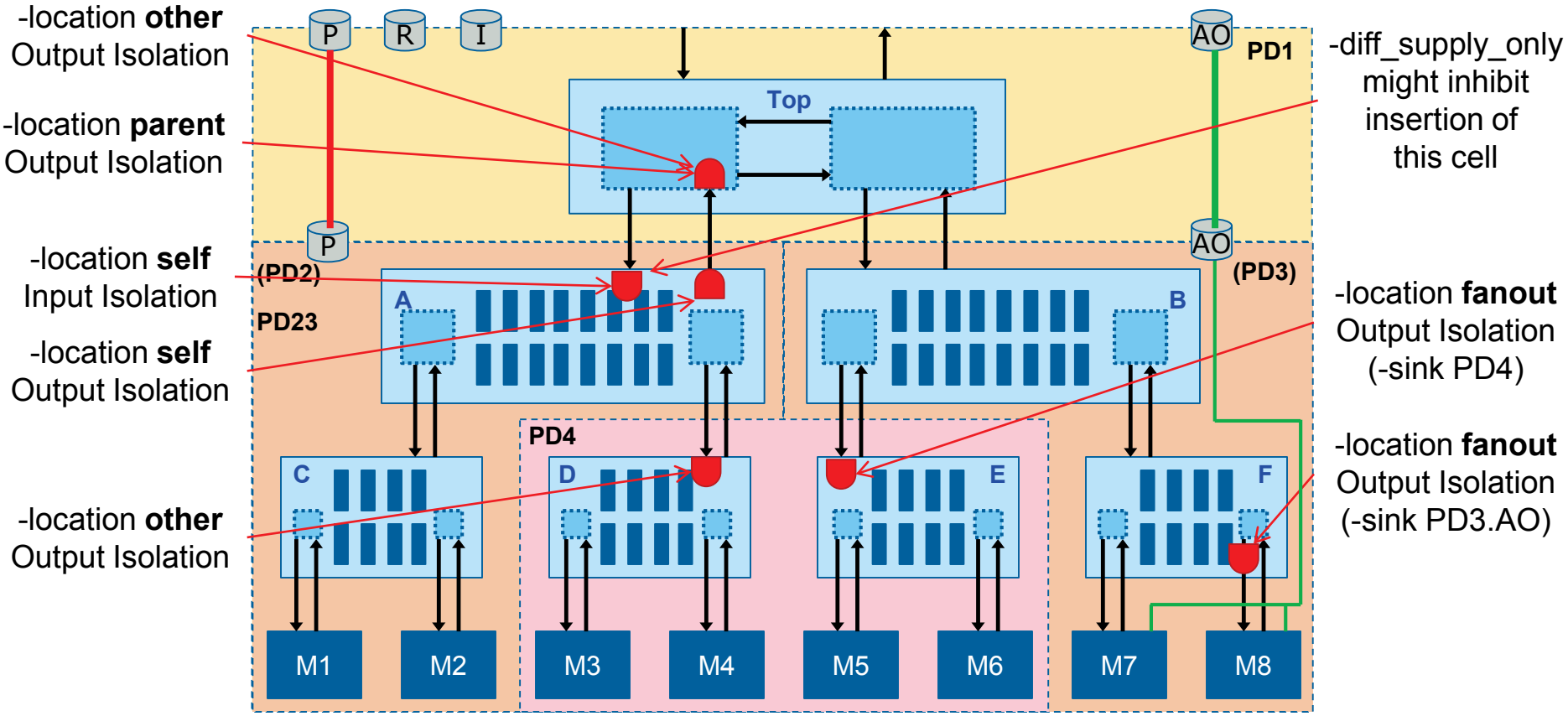
higher

Interface Cell Locations

- **Self**
  - The domain for which the strategy is defined
- **Parent**
  - The domain "above" the self domain
- **Other**
  - The domains "above" and "below" the self domain
- **Fanout**
  - The domain in which the receiving logic is contained

Isolation Cell Locations

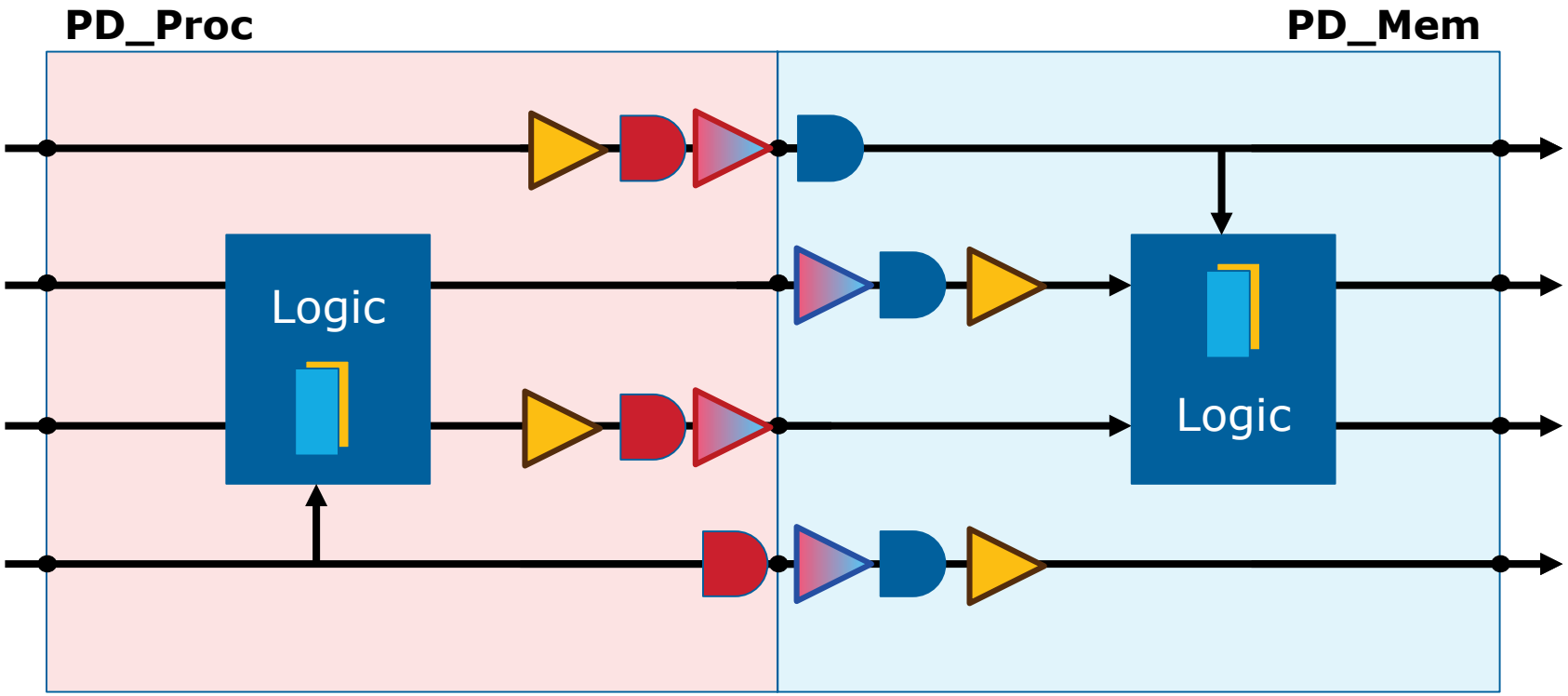
```
set_isolation ISO1 -domain PD2 -location …
```



Other Isolation Cell Parameters

- **Clamp Value**
  - specified with
    ```
    -clamp_value < 0 | 1 | any | Z | latch >
    ```
- **Control**
  - specified with
    ```
    -isolation_signal <signal name>
    -isolation_sense <high | low>
    ```
- **Supply**
  - specified with
    ```
    -isolation_supply <supply set name>
    ```
  - if not specified, uses default_isolation supply of location
    - can be a single-rail cell if containing domain is always on when enabled
    - otherwise typically requires a dual-rail cell

Strategy Execution Order



Retention, then **Repeater**, then **Isolation**, then **Level Shifter**

Strategy Interactions

	Retention	Repeater	Isolation	Level Shifter
Retention	--	affects	affects	affects
Repeater	affected by	--	affects	affects
Isolation	affected by	affected by	--	affects
Level Shifter	affected by	affected by	affected by	--

Strategies may change driver and/or receiver supplies of a port

This may affect `-source/-sink` filters of subsequently executed strategies

Supply Ports/Nets

- **Represent supply ports, pins, and rails**
  - Primary supply inputs, supply pins of cells, nets in between
- **Are connected together to create supply network**
  - Together with power switches to control power distribution
- **Deliver power/ground/etc. supplies to domains**
  - Delivered values determine how domain functions
- **Have and propagate {state, voltage} values**
  - States are UNDETERMINED, OFF, PARTIAL_ON, FULL_ON
  - Voltages are fixed-point values with microvolt precision
- **Examples**
  - {FULL_ON 1.2} {PARTIAL_ON 0.81} {OFF}

"Power" (Supply) Switches

- **Have one or more supply inputs**
  - Defined with `-input_supply_port`
- **Have one supply output**
  - Defined with `-output_supply_port`
- **Have one or more control inputs**
  - Defined with `-control_port`
- **Have one or more control states**
  - Defined with `-on_state` or `-on_partial_state`
  - Also can include `-error_state` and/or `-off_state`
- **Conditionally propagate input supply values to output**
  - Based on which control states are active

Power Switches

- **Examples**

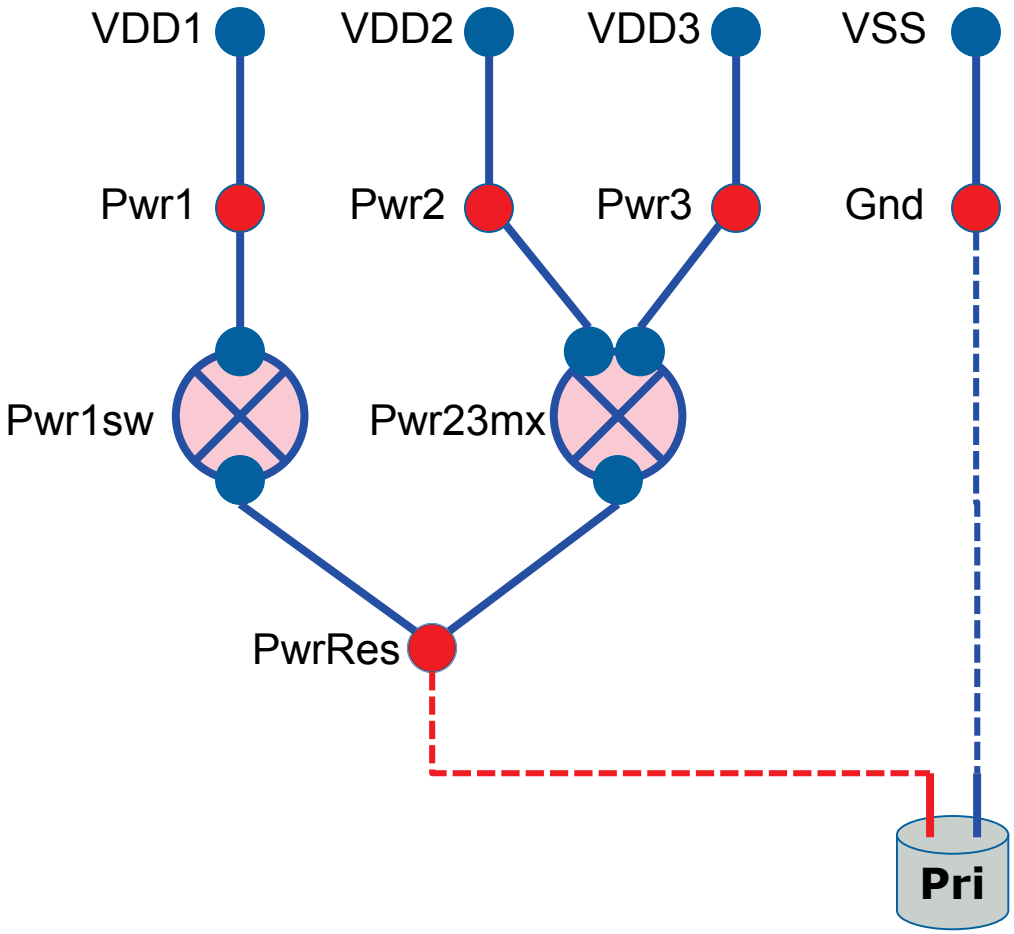
```
create_power_switch Simple \
  -output_supply_port {vout} \
  -input_supply_port {vin} \
  -control_port {ss_ctrl} \
  -on_state {ss_on vin { ss_ctrl }} \
  -off_state {ss_off { ! ss_ctrl }}

create_power_switch TwoStage \
  -output_supply_port {vout} \
  -input_supply_port {vin} \
  -control_port {trickle_ctrl} \
  -control_port {main_ctrl} \
  -on_partial_state {ts_ton vin { trickle_ctrl }} \
  -on _state {ts_mon vin { main_ctrl }} \
  -off_state {ts_off { ! trickle_ctrl && ! main_ctrl }}
```

Input and Output Supply Ports

Control Port

Switch Input States and Output State

Supply Network Construction



Commands

```
create_supply_port …
create_supply_net …
connect_supply_net …

create_power_switch …
connect_supply_net …

create_supply_net \
  -resolved …
connect_supply_net …

create_supply_set \
  -update
```

Supply Equivalence

- **Supply Ports/Nets/Functions**
  - **Electrically equivalent** if same/connected/associated
  - **Functionally equivalent** if
    - they are electrically equivalent, or
    - they are declared functionally equivalent
      - example: outputs of two switches that have same input and control
- **Supply Sets**
  - **Functionally equivalent** if
    - both have the same required functions, and corresponding required functions are electrically equivalent; or
    - both are associated with the same supply set; or
    - they are declared functionally equivalent
      - Declaration works for verification only; must be explicitly connected for implementation

A Deeper Look at UPF Power Intent

- Logic Hierarchy
- Power Domains
- Power Domain Supplies
- Supply Sets
- Supply Connections
- Power Related Attributes
- Power States and Transitions
- Power Domain State Retention
- Power Domain Interface Management
- Supply Network Construction
- Supply Equivalence

For more details, read the IEEE 1801-2013 UPF spec, especially Clause 4, UPF Concepts

BREAK

Hard IP Modeling with Liberty and Verilog

Sushma Honnavara-Prasad

Principal Engineer

Broadcom

Leaf Cells vs Macros (IPs)

- **Leaf cell**
  - An instance that has no descendants, or an instance that has the attribute UPF_is_leaf_cell associated with it.
  - In a typical ASIC flow,
    - Used to denote a standard cell/IO/Analog IP etc in the design
    - Have simulation models/.libs associated with them
- **Macros**
  - Also called IPs, a piece of functionality optimized for power/area/performance
    - Soft macros – handed off as synthesizable HDL (technology agnostic)
    - Hard macros – handed off as LEF/GDS (technology specific)
      - Also a leaf cell
  - UPF_is_macro_cell attribute allows the model to be recognized as part of lower boundary of the domain containing the instance

Models Dealing with Power

- **Why do we need them:**
  - Power models provide a compact abstraction of the design while preserving the structural/functional power related properties of the design
  - Type of models we use change based on the design/verification phase
- **UPF Power models**
- **Simulation models**
  - Power aware RTL/Gate models (Verilog/VHDL/SV)
- **Implementation models**
  - Liberty
  - LEF etc.

Liberty Attributes and UPF

- **Leaf cell/macro cell power intent may be contained in liberty (.lib models)**
- **.lib can describe power/power management cell attributes**

.lib	UPF	Purpose
pg_pin	-	Supply pin definition
pg_type	UPF_pg_type	Function of a supply
pg_function	-	Supply expression for generated supplies
related_power_pin	UPF_related_power_port	Power associated with a port
related_ground_pin	UPF_related_ground_port	Ground associated with a port

Supply Attributes

Liberty Attributes and UPF

.lib	UPF	Purpose
is_macro_cell	UPF_is_macro_cell	Identify a hard-macro
-	UPF_is_leaf_cell	Identify a leaf-cell (all cells in .lib are considered leaf cells)
always_on	define_always_on_cell	Cells that can remain on while the domain is off
antenna_diode_type	define_diode_clamp	Define/describe diode clamps
is_isolation_cell	define_isolation_cell	Define isolation cell
isolation_cell_enable_pin	-enable	Identifies isolation cell enable
always_on	-always_on_pins	Always on Pin attribute
is_level_shifter	define_level_shifter_cell	Define level shifter cells
level_shifter_enable_pin	-enable	Enabled level shifter control

Power management cell attributes

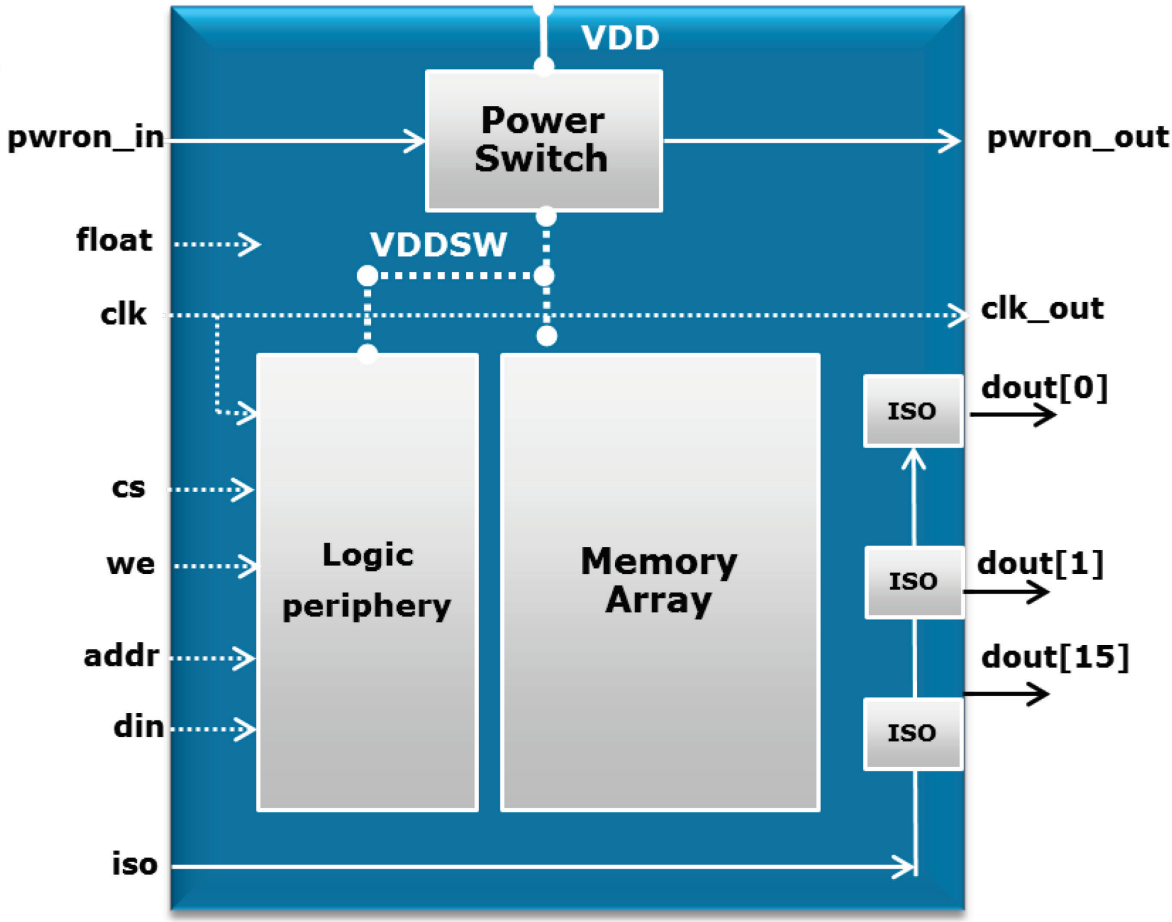
Liberty Attributes and UPF

.lib	UPF	Purpose
switch_cell_type	create_power_switch_cell	Define a power-switch cell
user_pg_type	gate_bias_pin	Identifies supply pin associated with gate input
retention_cell	define_retention_cell	Define a retention cell
retention_pin	-	Identifies the retention control(s)
save_action	save_function	Save pin function that enables save action
restore_action	restore_function	Restore pin function that enables restore action

Power management cell attributes

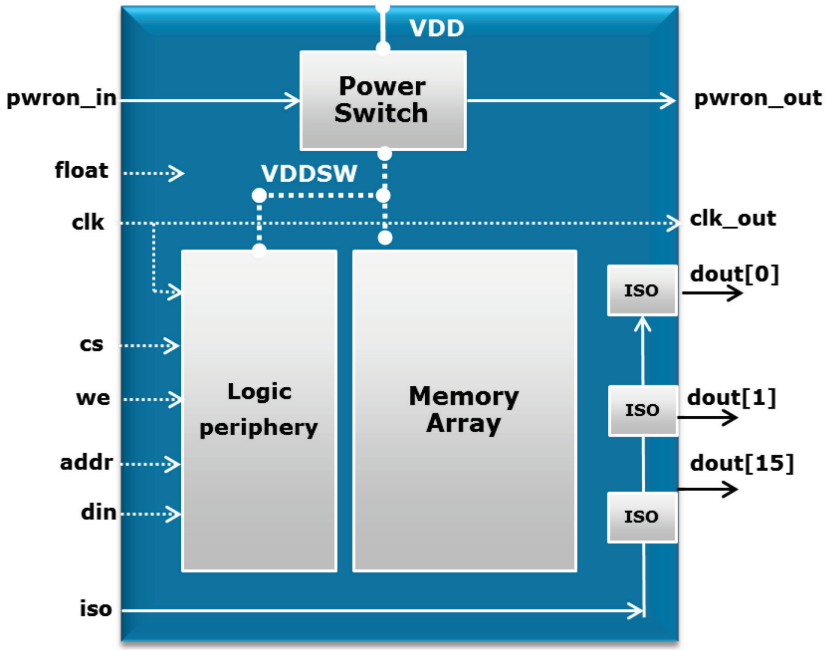
Example: Embedded SRAM

```
module RAM (cs, clk, addr,
din, dout, we, pwron_in,
pwron_out, iso, float,
clk_out);

input cs, clk, we;

input float;

input [3:0] addr;

input [31:0] din;

output [31:0] dout;

input pwron_in, iso;

output pwron_out;

output clk_out;

endmodule
```



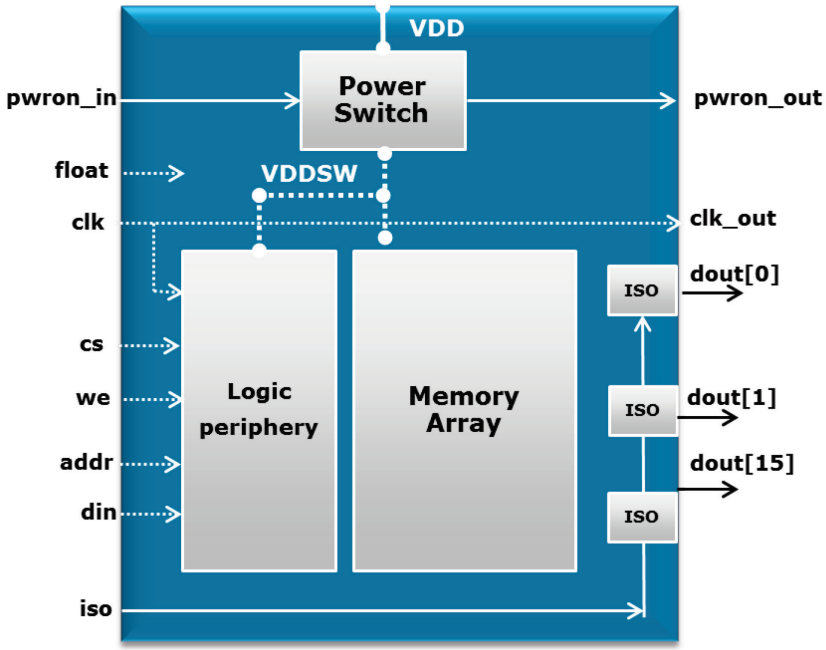
Embedded SRAM: Liberty

```
cell (SRAM) {
  is_macro_cell : true;
  switch_cell_type : fine_grain;
  pg_pin (VDD) {
    pg_type: primary_power;
    voltage_name: VDD;
    direction: input;
  }
  pg_pin (VSS) {
    pg_type: primary_ground;
    voltage_name: VSS;
    direction: input;
  }
```



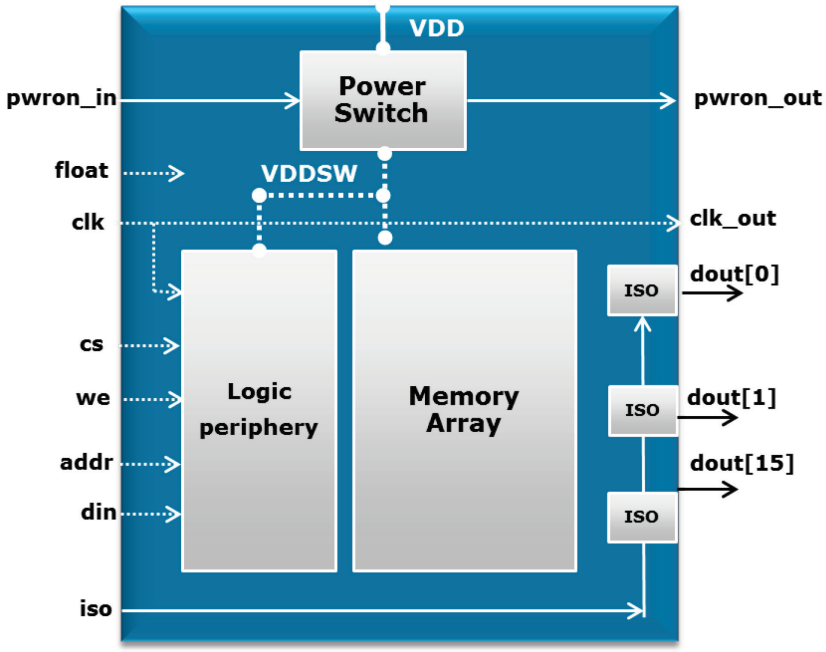
Embedded SRAM: Liberty

```
  pg_pin (VDDSW) {
    pg_type: internal_power;
    voltage_name: VDDSW;
    direction: internal;
    pg_function: VDD;
    switch_function: pwron_in;
  }
  pin (iso) {
    related_power_pin : VDD;
    related_ground_pin : VSS;
  }
}
```



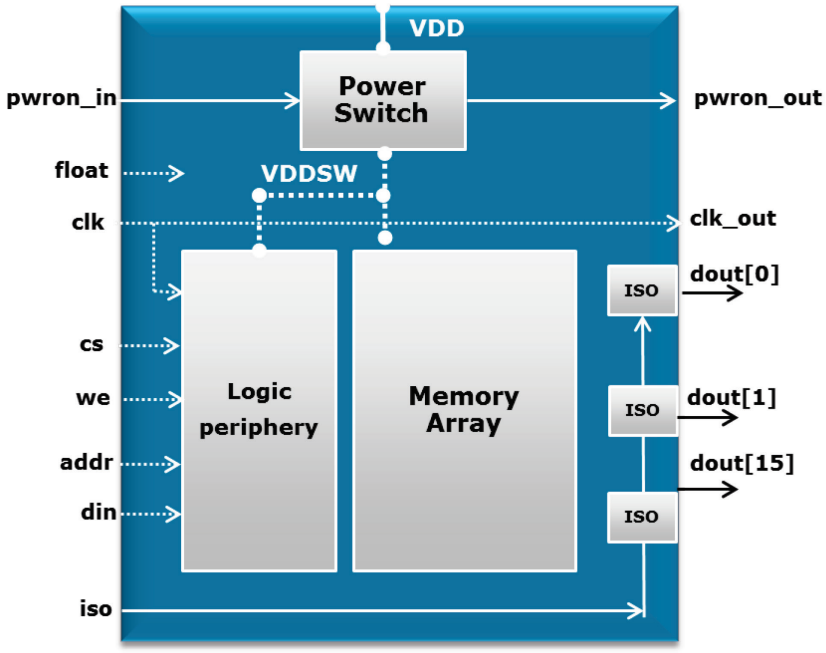
Embedded SRAM: Liberty

```
pin (pwron_in) {
  related_power_pin : VDD;
  related_ground_pin : VSS;
  switch_pin: true;
}
pin (clk) {
  related_power_pin : VDDSW;
  related_ground_pin : VSS;
}
pin (cs) {
  related_power_pin : VDDSW;
  related_ground_pin : VSS;
}
```



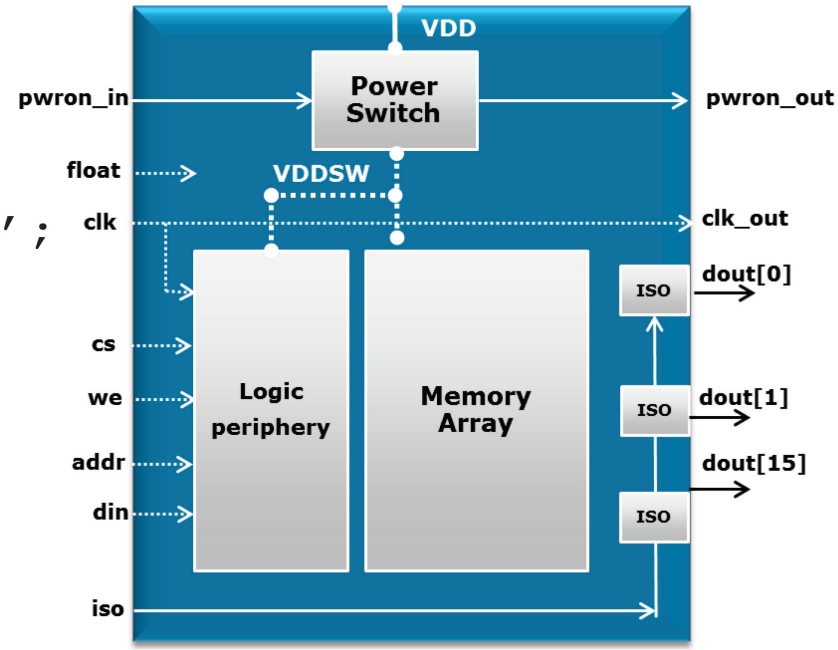
Embedded SRAM: Liberty

```
pin (pwron_out) {
  related_power_pin : VDD;
  related_ground_pin : VSS;
  function: pwron_in;
}
pin (dout[0]) {
  related_power_pin : VDD;
  related_ground_pin : VSS;
}
pin (dout[1]) {
  related_power_pin : VDD;
  related_ground_pin : VSS;
}
```



Embedded SRAM: Power Aware HDL

```
assign VDDSW = pwron_in && VDD;
assign supply_on = VDDSW && !VSS;

//Corrupt all the inputs associated
// with VDDSW when pwron_in is 0
assign clk_int = (supply_on)?clk: 'x';
assign cs_int = (supply_on)?cs: 'x';

//Corrupt array if iso !=1 when off
if (!pwron_in && !iso)
      array[M] <= 16{1'bx};

//Corrupt outputs when VDDSW is off
// and not isolated
assign dout = (iso)? 16{1'b0}: (supply_on)? dout_int:16{1'bx};
```



Power Model and Power Management Cell Commands

David Cheng

Architect

Cadence Design Systems

cadence™

Power Model

- For static tools (where the block content is a black box)
  - Liberty cell modeling provides basic modeling
  - *Power model* provides additional modeling:
    - Power states (power modes)
    - Feedthrough and floating ports
    - Detailed isolation picture of boundary ports
- For dynamic tools (where the block content is visible)
  - Similar to the block's design UPF
  - *Power model* provides a clear boundary that prevents higher-scope UPF constructs from "coming into" the block

Commands for Power Model

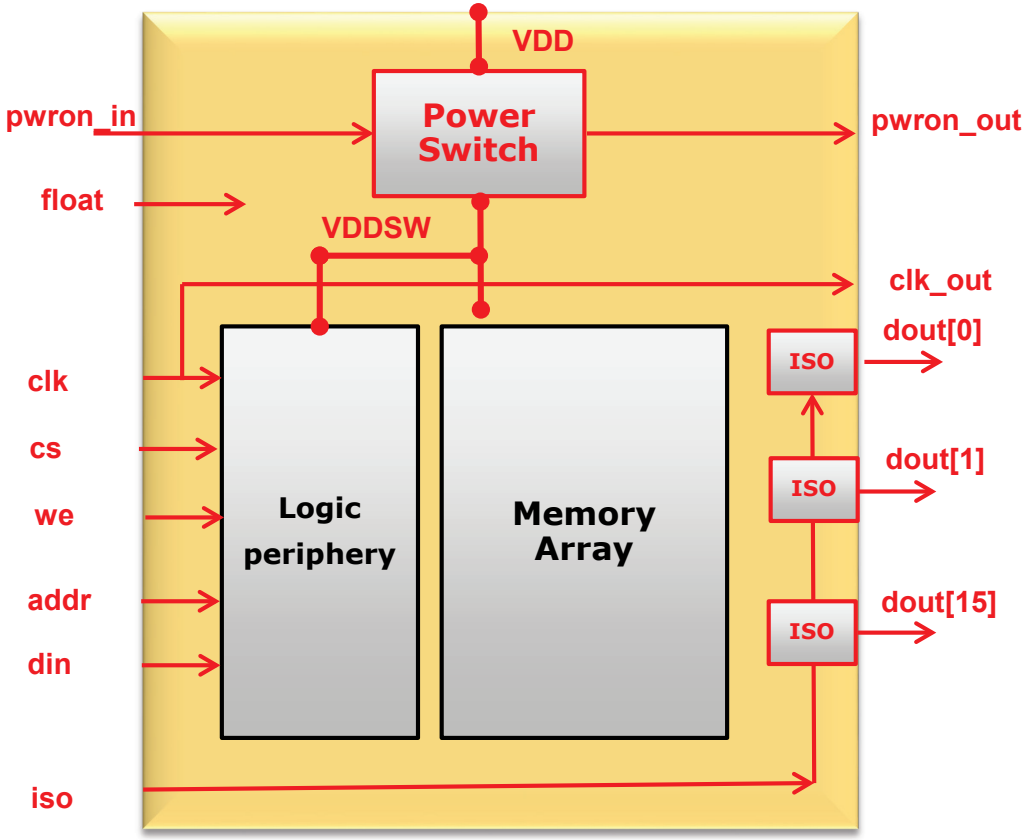
- Commands to define a power model containing other UPF commands

```
begin_power_model power_model_name [-for model_list]
   <UPF commands>
end_power_model
apply_power_model power_model_name
   [-elements instance_list]
   [-supply_map {
      {lower_scope_handle upper_scope_supply_set}*
   }]
```

- Certain commands cannot be used within a power model definition
  - **name_format**
  - **save_upf**
  - **save_scope**
  - **load_upf -scope**
  - **begin_power_model/end_power_model/apply_power_model**
  - Any deprecated/legacy commands/options

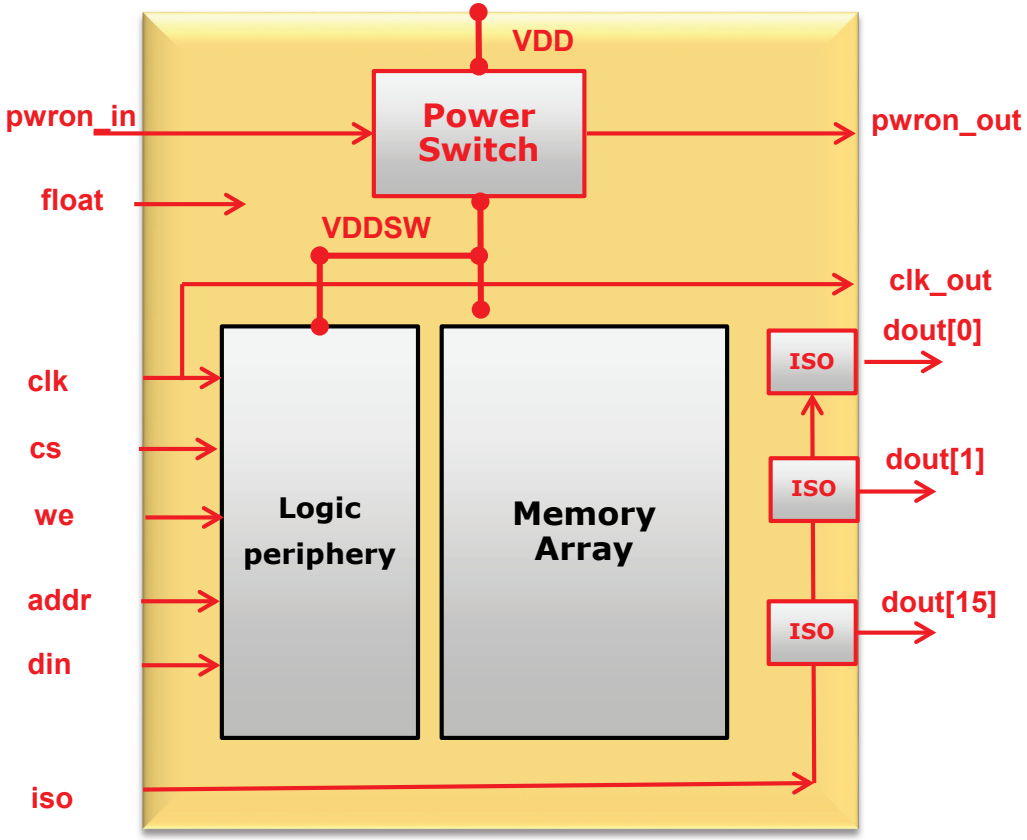
Same Example - Different Colors

- ***For static tools:***
  - *Only describes the boundary (the red part)*
  - *Internal is a black box*
  - *Similar to Liberty model*



Possible Checks in Mind

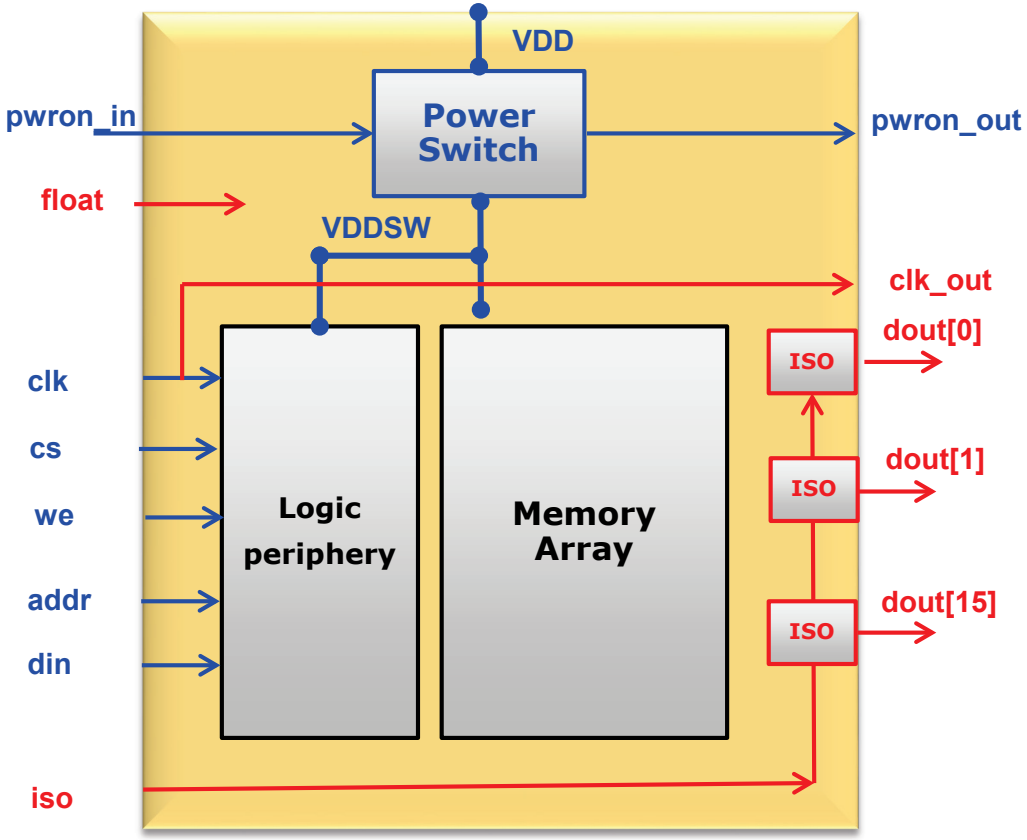
- Do I always assert *iso* before I assert *pwron_in*?
- Are my power states/ modes consistent with the block's?
- Is my logic compatible with the asserted clamp values?
- Are *clk* and *clk_out* treated the same?
- No check needed for *float*



Power Model Description

Blue: Liberty can also describe
Red: Liberty weaker

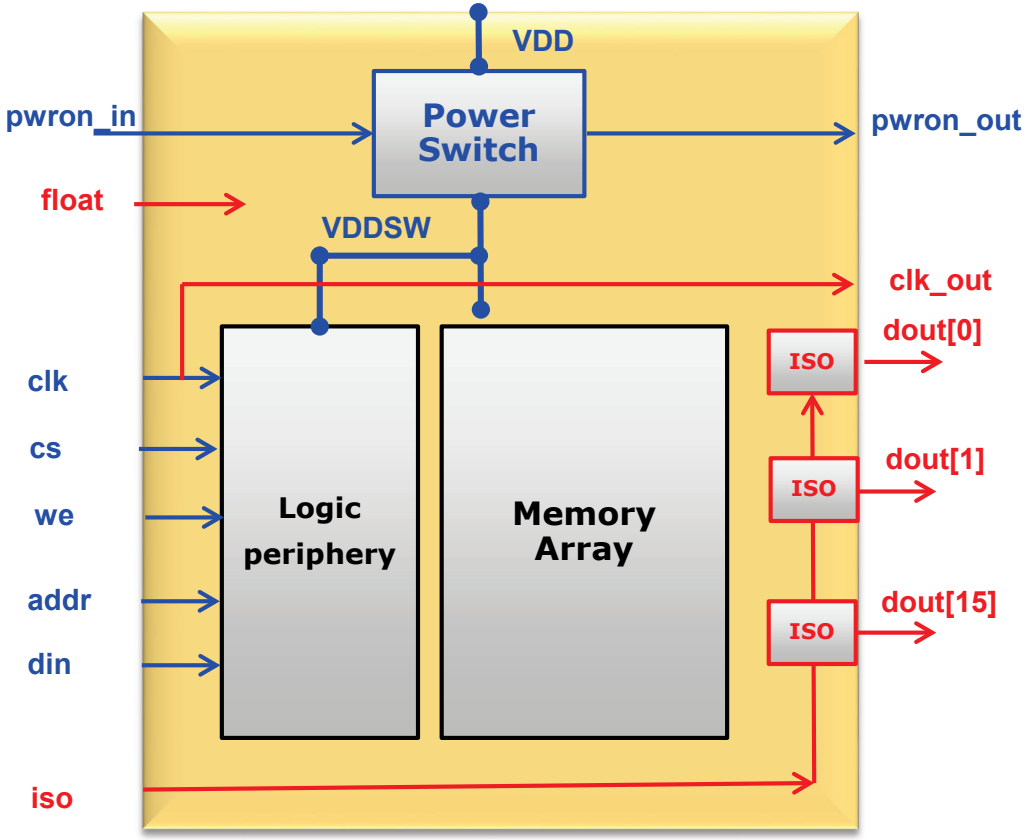
```
begin_power_model

create_power_domain PD1 \
   -include_scope

create_supply_set SS_VDD ...

create_supply_set SS_VDDSW ...

set_port_attributes \
   -elements . \
   -applies_to inputs \
   -receiver_supply SS_VDDSW

set_port_attributes \
   -ports {pwron_in iso}
   -receiver_supply  SS_VDD

set_port_attributes \
   -elements . \
   -applies_to outputs \
   -driver_supply SS_VDD

...
```



Power Model Description

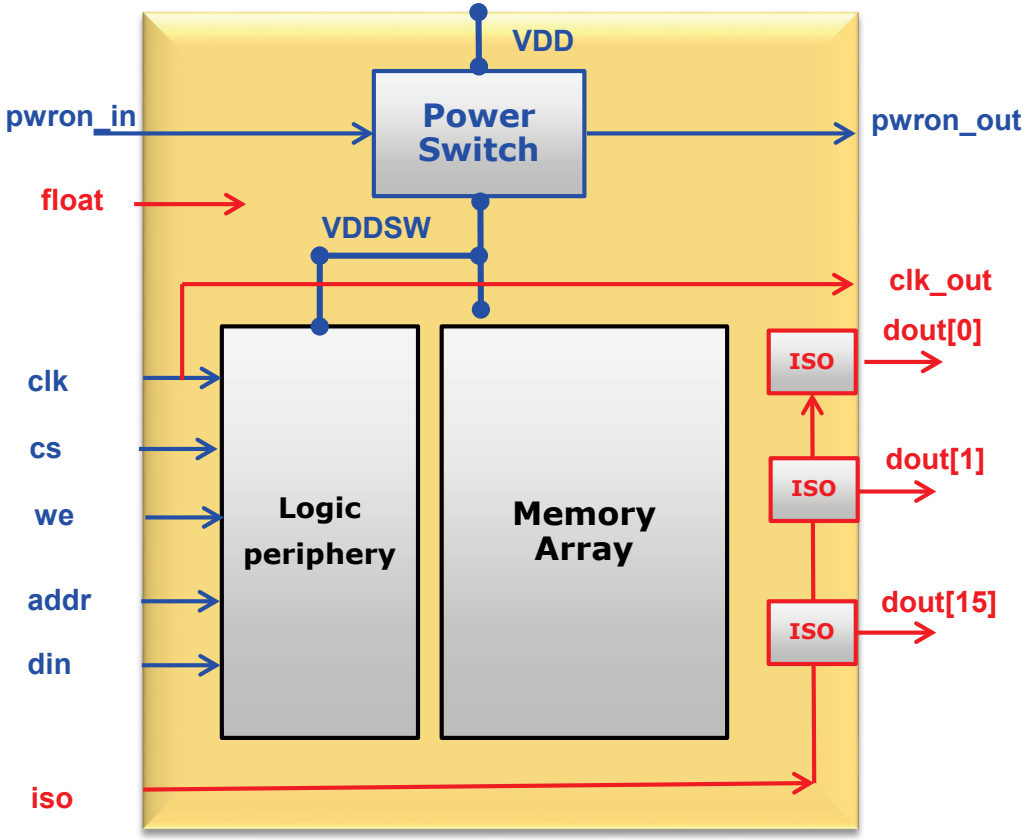
Blue: Liberty can also describe
Red: Liberty weaker

```
...
set_port_attributes \
   -ports {float}
   -unconnected

set_port_attributes \
   -ports {clk clk_out} \
   -feedthrough

set_isolation -domain PD1 \
   -elements {dout} \
   -clamp_value 1 \
   -isolation_signal iso ...

create_pst PST1 \
   -supplies {VDD VDDSW...}

end_power_model
```



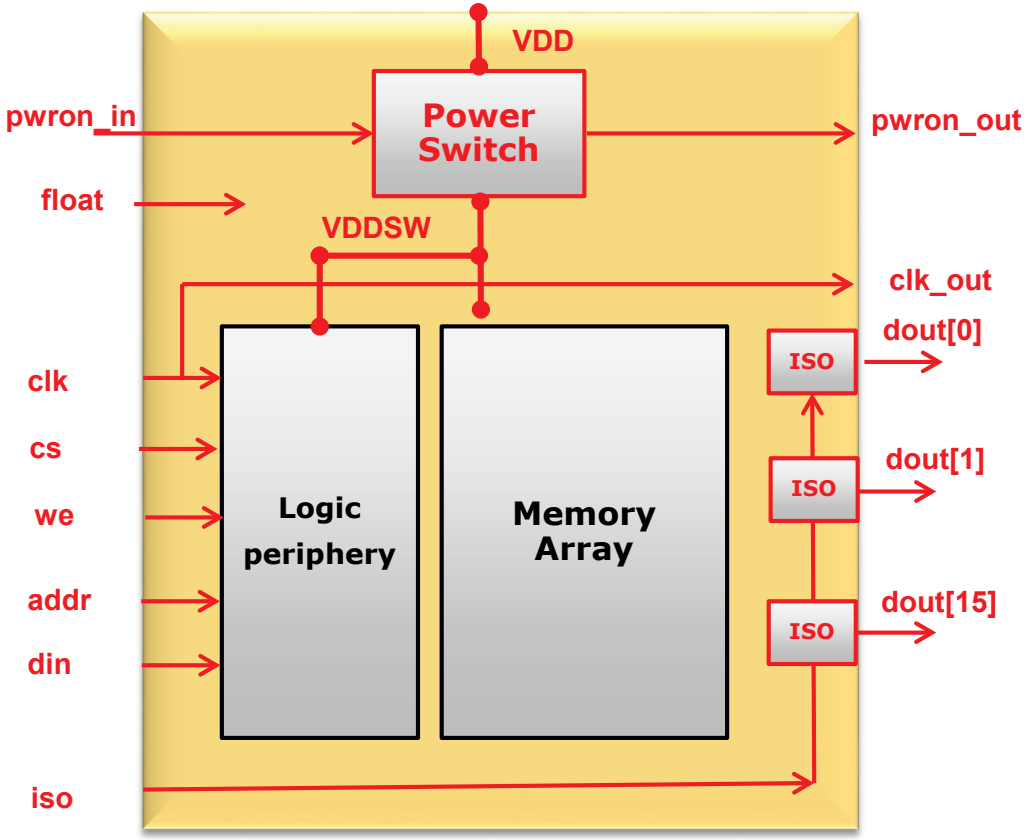
Liberty Description

```
pin (clk_out) {
  function: clk;
}
pin (float) {
}
pin (dout[0]) {
  related_power_pin : VDD;
  related_ground_pin : VSS;
  is_isolated: true;
  (isolation_enable_condition:
    iso;)
}
```



Possible Checks in Mind

- **Do I always assert *iso* before I assert *pwron_in*?**
  - Now possible with Liberty's isolation_enable_condition
- **Are my power states/ modes consistent with the block's?**
  - Liberty has no PSTs
- **Is my logic compatible with the asserted clamp values?**
  - Liberty has no "clamp_value"
- **Are *clk* and *clk_out* treated the same?**
  - Liberty has no "feedthrough"
- **No check needed for *float***
  - Liberty has no "unconnected"



Two Flavors of Dynamic Simulation for a Block

- Power aware
  - HDL alone is enough
  - Often “hard macros”
  - (Review Sushma’s slide)
- Power unaware
  - HDL + UPF
  - Often “soft macros”
    - E.g., an HDL block that goes through implementation and is now “hardened”
  - Without power model, the UPF looks just like your regular design

Power Unaware HDL

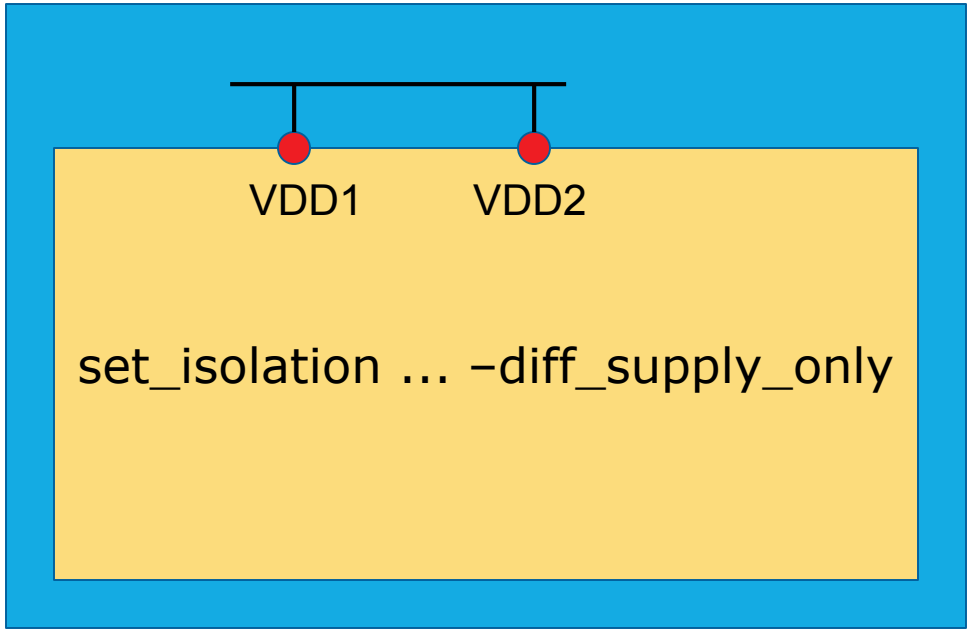
```
assign VDDSW = pwron_in && VDD;
assign supply_on = VDDSW && !VSS;

//Corrupt all the inputs associated
// with VDDSW when pwron_in is 0
assign clk_int = (supply_on)?clk: 'x';
assign cs_int = (supply_on)?cs: 'x';

//Corrupt array if iso !=1 when off
if (!pwron_in && !iso)
      array[M] <= 16{1'bx};

//Corrupt outputs when VDDSW is off
// and not isolated
assign dout = (iso)? 16{1'b0}: (supply_on)? dout_int:16{1'bx};
```

- Note: many details abstracted out, e.g.:
  - always @ (addr or cs or...)
  - if (we) { write } else { read }
  - assign dout_int = ...

Only the **RED** code is required

Power Model (for Dynamic Tools)

- A power model is similar to the block's design UPF, except a clear boundary that prevents higher-scope UPF constructs from "coming into" the block
- Example: two supply ports in a block are shorted at higher scope
  - With the clear boundary of a power model, the simulation tool would treat the two supplies still as "different supplies"
- Other examples exist



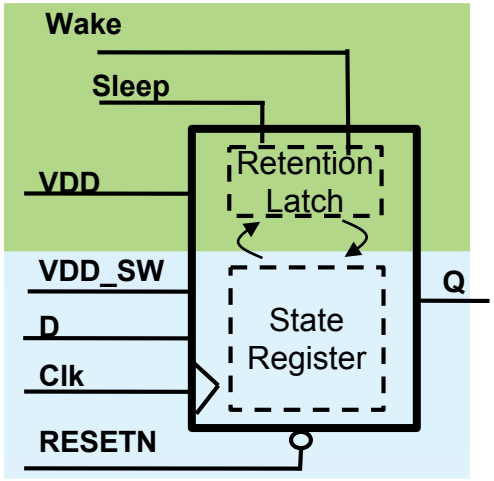
Summary on Power Model

- In short, power model indicates this block is already done
- For static tools (where the block content is a black box)
  - Liberty cell modeling provides basic modeling
  - *Power model* provides additional modeling:
    - Power states (power modes)
    - Feedthrough and floating ports
    - Detailed isolation picture of boundary ports
- For dynamic tools (where the block content is visible)
  - Similar to the block's design UPF
  - *Power model* provides a clear boundary that prevents higher-scope UPF constructs from "coming into" the block

Modeling Power Management Cells

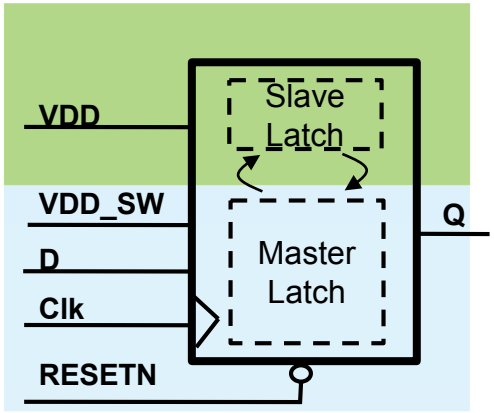
- **1801-2013 provides commands to model the following power management cells**
  - State retention cells
  - Always-on cells
  - Isolation cells
  - Level shifter cells
  - Power switch cells
  - Diode cells
- **An alternative to modeling low power management cells using Liberty**

Example: State Retention Cells



Retention flops with both save and restore

```
define_retention_cell -cells SR1 \
        -save_function {Sleep high} \
        -restore_function {Wake high} \
        -restore_check !Clk -save_check !Clk \
        -power_switchable VDD_SW -power VDD \
        -ground VSS
```

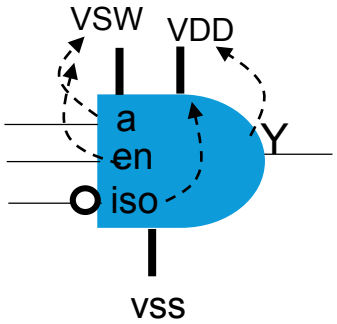


Retention flops with live slave latch

```
define_retention_cell -cells SR1 \
        -restore_check !Clk -save_check !Clk \
        -power_switchable VDD_SW -power VDD \
        -ground VSS
```

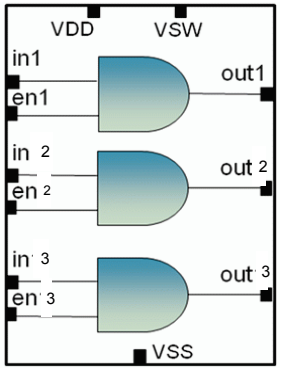
Relationship to Liberty

- **1801-2013 Annex H contains the current mapping**
  - Sushma covered with some examples earlier
- **1801 does not have to "wait" for Liberty if special needs arise, e.g.:**



Isolation cell with two control pins

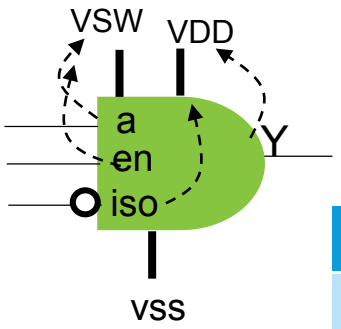
```
define_isolation_cell -cells myiso \
     -power_switchable VSW -power VDD -ground VSS \
     -enable iso -aux_enables en -valid_location source
```



Multi-bit isolation cell

```
define_isolation_cell -cells IsoLL \
       -power_switchable VSW -power VDD -ground VSS \
       -valid_location source \
       -pin_groups {{in1 out1 en1} {in2 out2 en2} \
        {in3 out3 en3}}
```

Work with Strategies, Example:



Cell definition

```
define_isolation_cell -cells isoandlow \
       -power_switchable VSW -power VDD -ground VSS \
       -enable iso -aux_enables en
```

Strategy specification

```
set_isolation iso1 -domain PD1 -location self \
      -isolation_signal { iso_drvr en_drvr} \
      -isolation_sense { high low } -clamp_value 0
```

Summary on 1801-2013 Library Cell Commands

- Model commonly used power management cells
- Compact form to specify common attributes of many cells in a single command
- An alternative modeling in addition to Liberty
- Direct mapping with corresponding strategies

Low Power Design Methodology for IP Providers

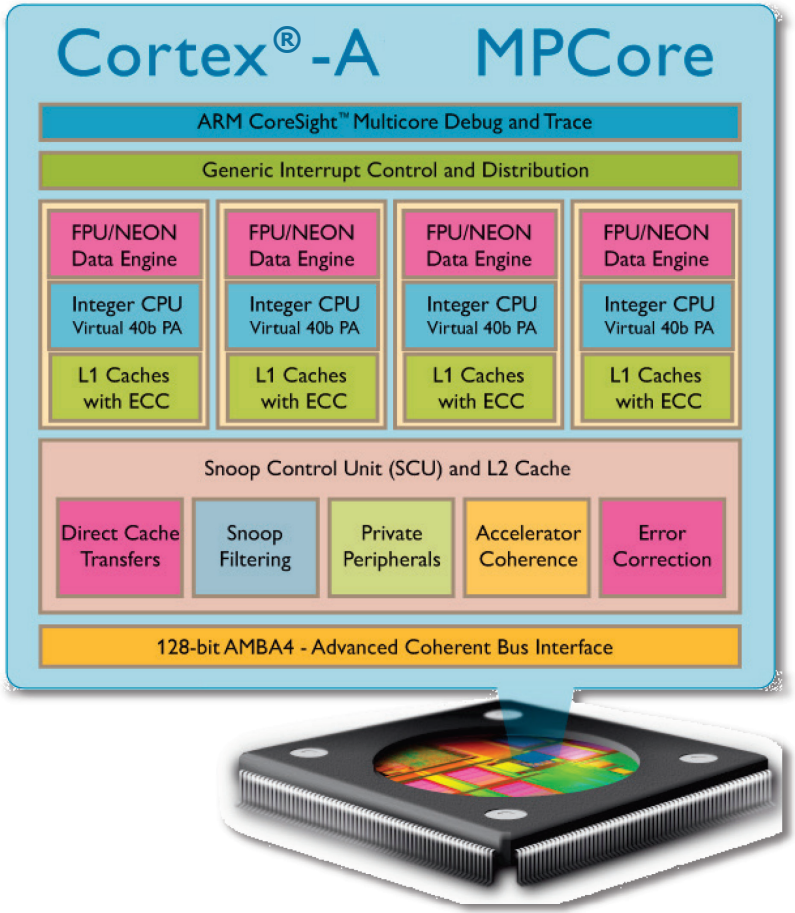
John Biggs

Senior Principal Engineer

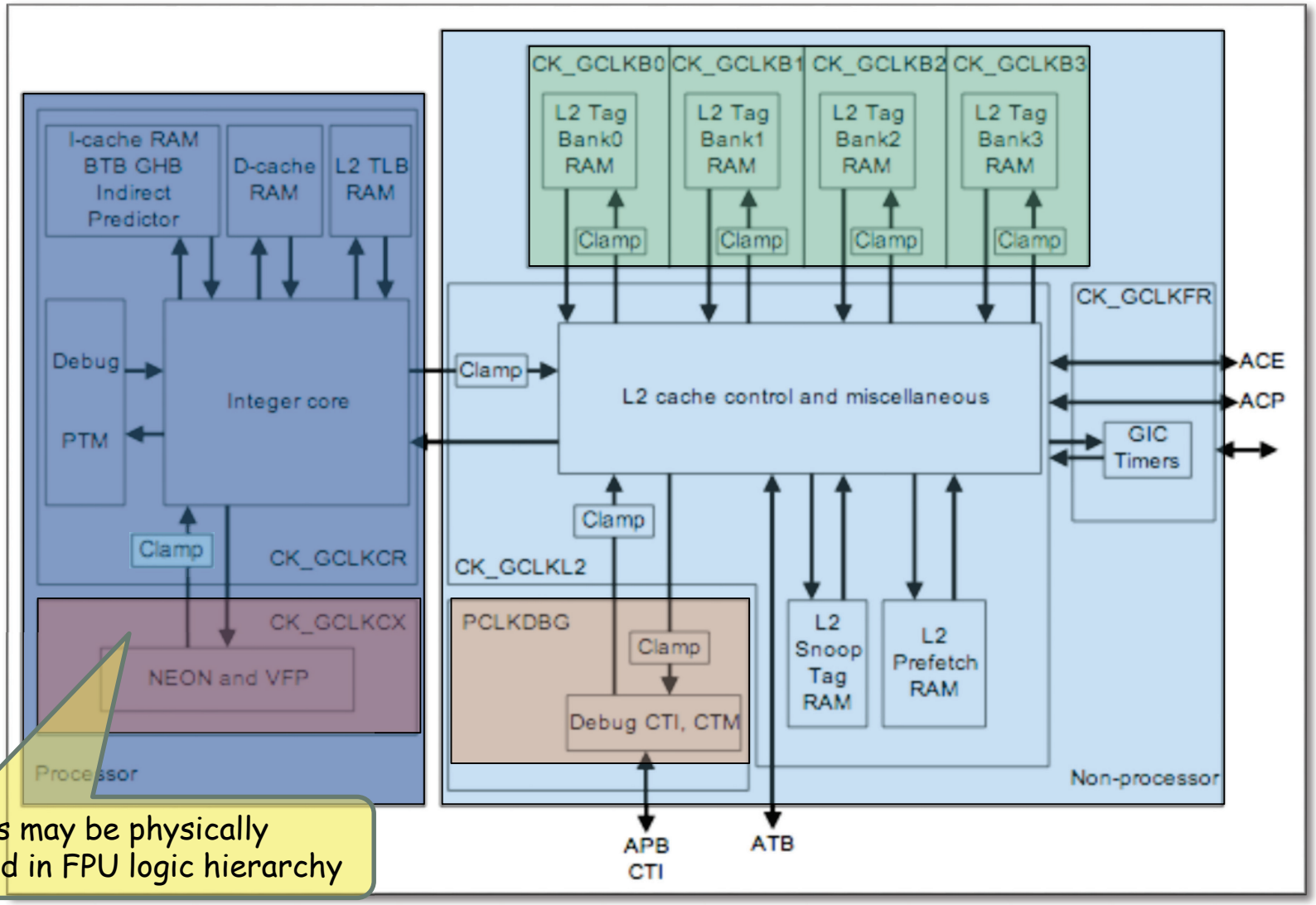
ARM

ARM® Cortex®-A MPCore Example

- **Generic simplified example**
- **Symmetric Multicore Cluster**
  - 1-4 CPUs with L2 cache
- **Each CPU has 2 power domains**
  - Integer CPU, L1 Cache, Debug and Trace
  - Floating point and SIMD engine
- **The MPCore has 3 power domains**
  - L2 cache RAMs.
  - L2 cache control
  - Debug and Trace



Cortex-A MPCore Power Domains

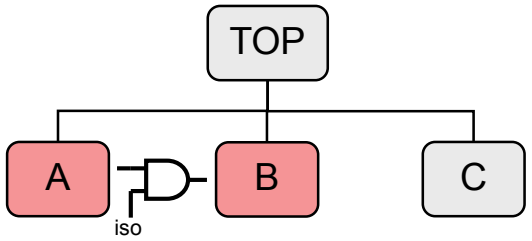


Clamps may be physically located in FPU logic hierarchy

Tip: Align Power Domains and Logic Hierarchy

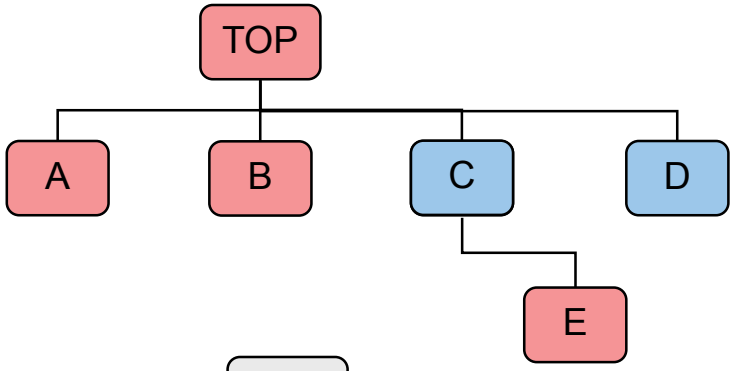
- Multi-element power domains can lead to unexpected “intra-domain” isolation

```
create_power_domain RED -elements {A B}
```



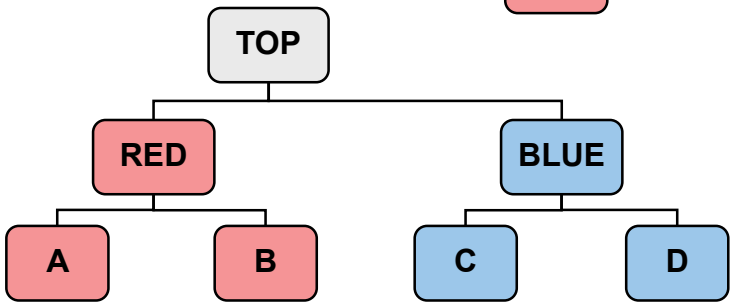
- These can often be avoided with a different approach

```
create_power_domain RED -elements {.}
create_power_domain BLUE -elements {C}
```



- Better to align power domains with logic hierarchy if at all possible

```
create_power_domain RED  -elements {RED}
create_power_domain BLUE -elements {BLUE}
```

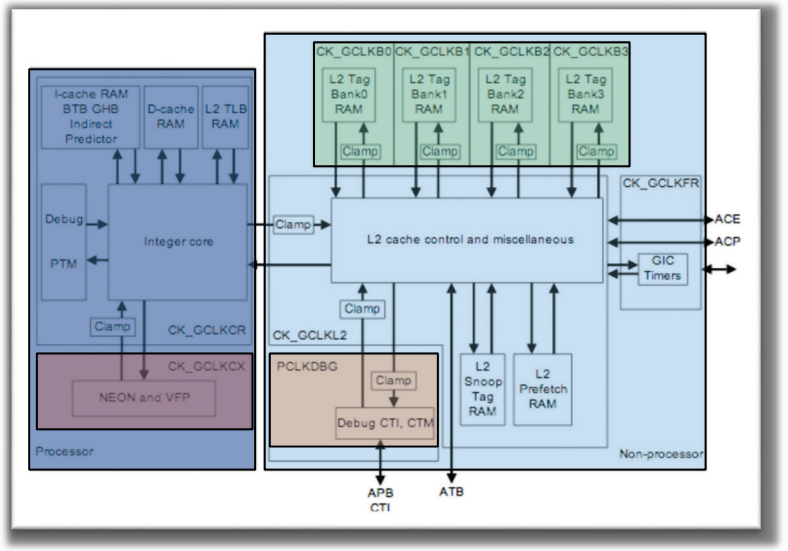


Failing that, use `-diff_supply_only` option or the `-source/-sink` filters

Cortex-A MPCore Power States

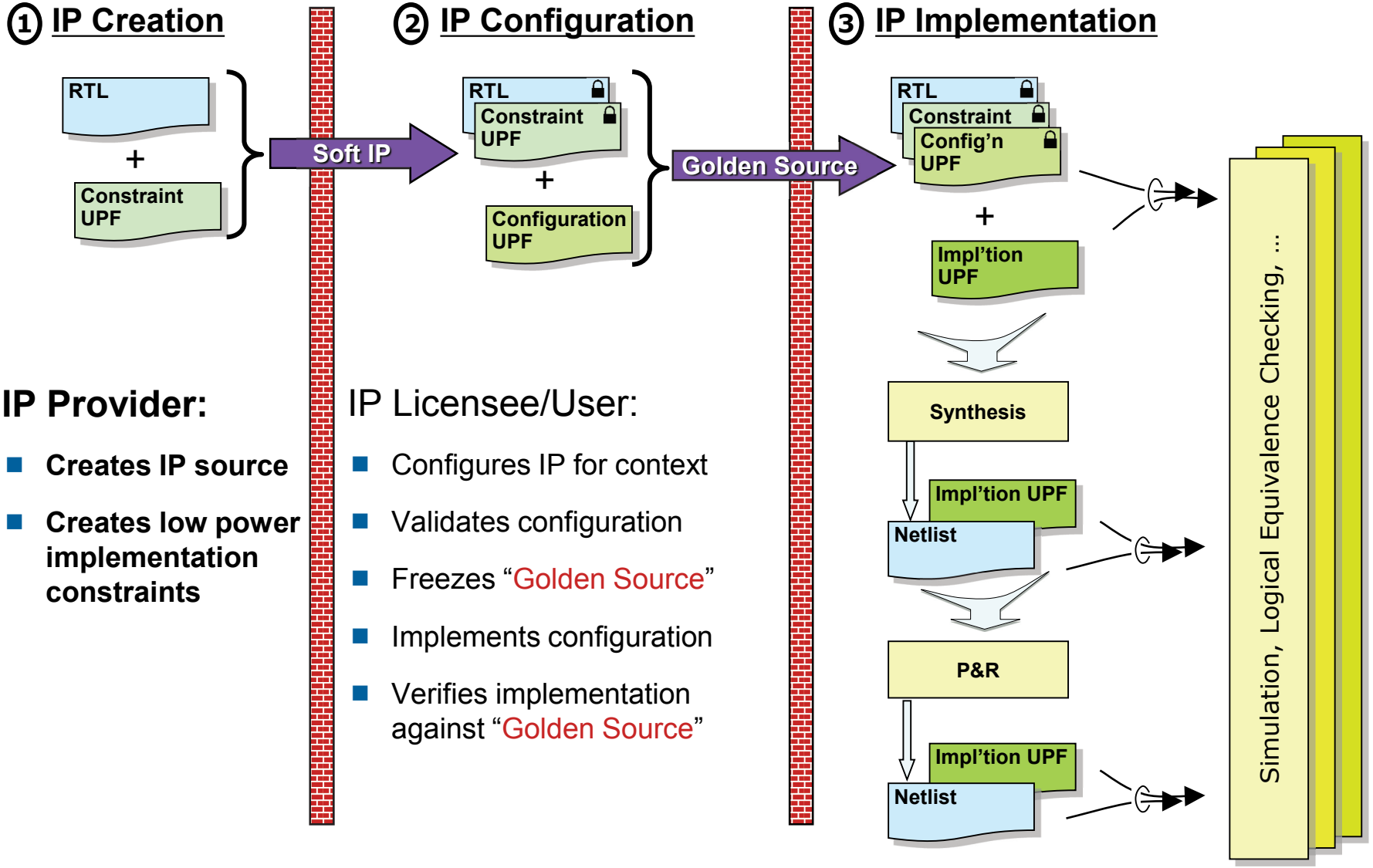
	PD_CPU	PD_FPU
RUN	RUN	*
RETENTION	RET	!RUN
OFF	OFF	OFF

!RUN = RET or OFF



	PD_CPU0	PD_CPU1	PD_CLSTR	PD_L2RAM	PD_DBG
RUN	RUN	RUN	RUN	RUN	*
RETENTION	!RUN	!RUN	!RUN	RET	*
DORMANT	OFF	OFF	OFF	ON	*
DEBUG	*	*	*	*	RUN
OFF	OFF	OFF	OFF	OFF	OFF

Successive Refinement of Power Intent



IP Provider:

- **Creates IP source**
- **Creates low power implementation constraints**

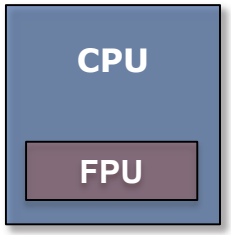
IP Licensee/User:

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

A Soft IP Provider Need Only Declare Four Things:

1. **The "atomic" power domains in the design**
   - These can be merged but not split during implementation
2. **The state that needs to be retained during shutdown**
   - Without prescribing how retention is controlled
3. **The signals that need isolating high/low**
   - Without prescribing how isolation is controlled
4. **The legal power states and sequencing between them**
   - Without prescribing absolute voltages

CPU Constraints



Put everything in PD_CPU execpt PD_FPU

1. Atomic power domains

```
create_power_domain PD_CPU -elements {.} \
  -exclude_elements "$FPU" -atomic
create_power_domain PD_FPU -elements "$FPU" -atomic
```

Retain "all or nothing"

2. Retention requirements

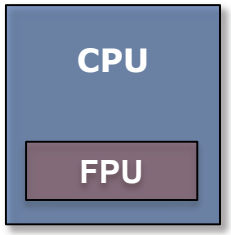
```
set_retention_elements RETN_LIST -elements {.}
```

3. Isolation requirements

```
set_port_attributes -model cortex_cpu -applies_to outputs \
                              -exclude_ports "$CPU_CLAMP1" -clamp_value 0
set_port_attributes -model cortex_cpu -ports "$CPU_CLAMP1" -clamp_value 1
set_port_attributes -elements "$FPU" -applies_to outputs   -clamp_value 0
```

Clamp everything low by default
Then call out the exceptions

CPU Constraints (cont...)



4. Power State

Define PD_FPU in terms of its supply sets

```
add_power_state PD_FPU -domain \
    -state {RUN -logic_expr {primary == ON     \
                && default_retention == ON }} \
    -state {RET -logic_expr {primary == OFF    \
                && default_retention == ON }} \
    -state {OFF -logic_expr {primary == OFF    \
                && default_retention == OFF }}
```

PD_FPU	primary	retention
RUN	ON	ON
RET	OFF	ON
OFF	OFF	OFF

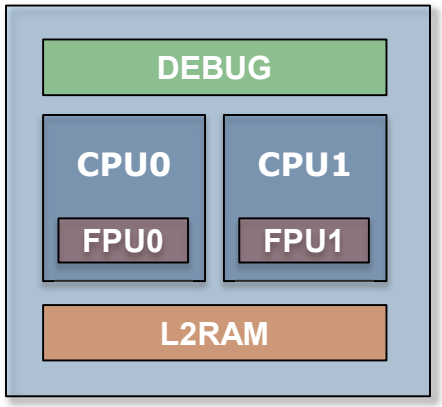
```
add_power_state PD_CPU -domain \
    -state {RUN -logic_expr {primary == ON     \
                && default_retention == ON}    \
    -state {RET -logic_expr {primary == OFF    \
                && default_retention == ON     \
                        && PD_FPU != RUN}} \
    -state {OFF -logic_expr {primary == OFF    \
                && default_retention == OFF    \
                        && PD_FPU == OFF}}
```

PD_CPU	primary	retention	PD_FPU
RUN	ON	ON	*
RET	OFF	ON	!RUN
OFF	OFF	OFF	OFF

Define PD_CPU in terms of its supply sets and the state of PD_FPU

In RET the FPU state can be anything but RUN

Cluster Constraints



1. "Atomic" power domains

Put everything in PD_CLSTR except PD_L2RAM and PD_DEBUG

```
create_power_domain PD_CLSTR -elements {.} \
   -exclude_elements {"$L2RAM" "$DEBUG"} -atomic
create_power_domain PD_L2RAM -elements "$L2RAM" -atomic
create_power_domain PD_DEBUG -elements "$DEBUG" -atomic
```

2. Retention requirements

```
set_retention_elements RETN_LIST -elements {.}
```

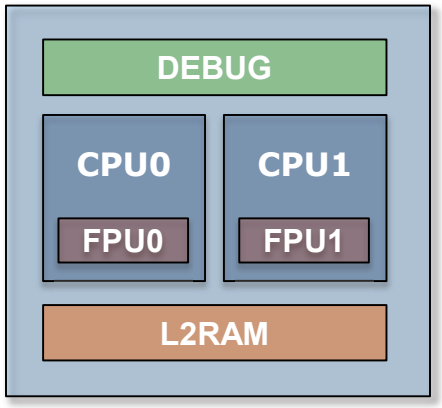
Retain "all or nothing"

3. Isolation requirements

```
set_port_attributes -model cortex_cluster -applies_to outputs \
                                  -exclude_ports "$CLSTR_CLAMP1" -clamp_value 0
set_port_attributes -model cortex_cluster -ports "$CLSTR_CLAMP1" -clamp_value 1
set_port_attributes -elements $L2RAM -applies_to outputs \
                               -exclude_ports "$L2RAM_CLAMP1" -clamp_value 0
set_port_attributes -elements $L2RAM -ports "$L2RAM_CLAMP1" -clamp_value 1
set_port_attributes -elements $DEBUG -applies_to outputs    -clamp_value 0
```

Clamp everything low by default
Then call out the exceptions

Cluster Constraints (cont...)



4. Power Domain State

Define PD_L2RAM in terms of its supply sets

```
add_power_state PD_L2RAM -domain \
    -state {RUN -logic_expr {primary == ON    \
                && default_retention == ON }}\
    -state {RET -logic_expr {primary == OFF   \
                && default_retention == ON }}\
    -state {OFF -logic_expr {primary == OFF   \
                && default_retention == OFF}}
```

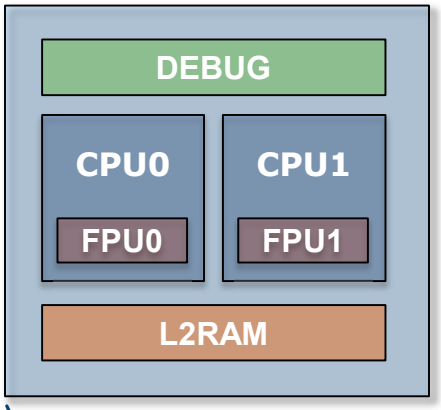
PD_L2RAM	primary	retention
RUN	ON	ON
RET	OFF	ON
OFF	OFF	OFF

```
add_power_state PD_DEBUG -domain \
    -state {RUN -logic_expr {primary == ON   }}\
    -state {OFF -logic_expr {primary == OFF }}
```

PD_DEBUG	primary
RUN	ON
OFF	OFF

Define PD_DEBUG in terms of its supply sets

Cluster Constraints (cont...)



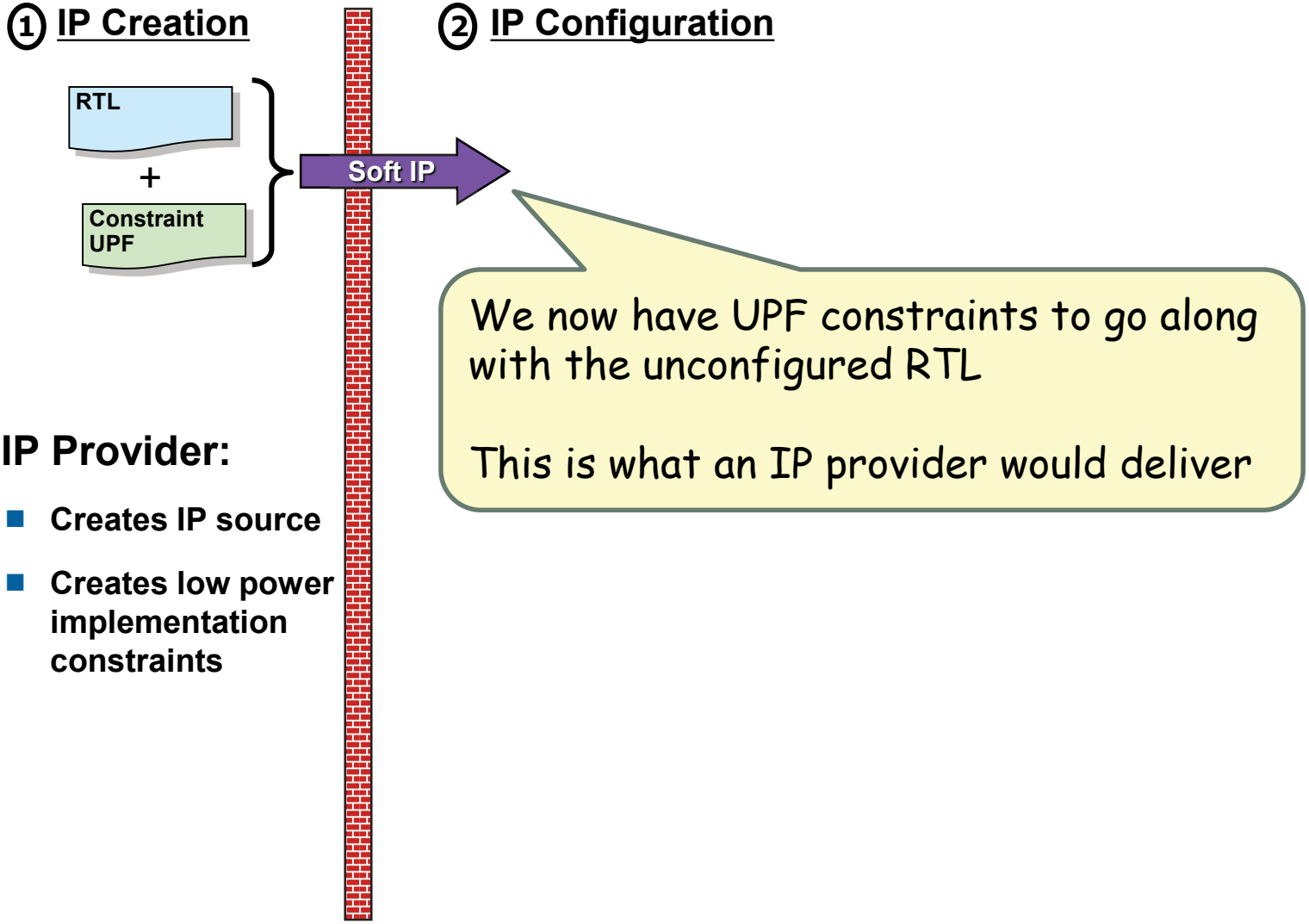
4. Power Domain State (Cont.)

```
add_power_state PD_CLSTR -domain \
    -state {RUN -logic_expr {primary == ON  && PD_L2RAM     == RUN   \
                  &&  uCPU0/PD_CPU == RUN || uCPU1/PD_CPU == RUN}} \
    -state {RET -logic_expr {primary == OFF && PD_L2RAM     == RET   \
                  &&  uCPU0/PD_CPU != RUN && uCPU0/PD_CPU != RUN}} \
    -state {DMT -logic_expr {primary == OFF && PD_L2RAM     == RUN   \
                  &&  uCPU0/PD_CPU == OFF && uCPU0/PD_CPU == OFF}} \
    -state {DBG -logic_expr {PD_DEBUG  == RUN}}
    -state {OFF -logic_expr {primary == OFF && PD_L2RAM     == OFF   \
                  &&  uCPU0/PD_CPU == OFF && uCPU0/PD_CP   == OFF   \
                  &&       PD_DEBUG == OFF}}
```

Define PD_CLSTR in terms of its supply sets and also the state of PD_L2RAM and PD_DEBUG

PD_CLSTR	primary	PD_L2RAM	PD_CPU0	PD_CPU1	PD_DEBUG
RUN	ON	RUN	RUN	RUN	*
RET	OFF	RET	!RUN	!RUN	*
DMT	OFF	RUN	OFF	OFF	*
DBG	*	*	*	*	RUN
OFF	OFF	OFF	OFF	OFF	OFF

Successive Refinement of Power Intent



IP Provider:

- Creates IP source
- Creates low power implementation constraints

Flat vs Hierarchical

- **Configure instances in context**

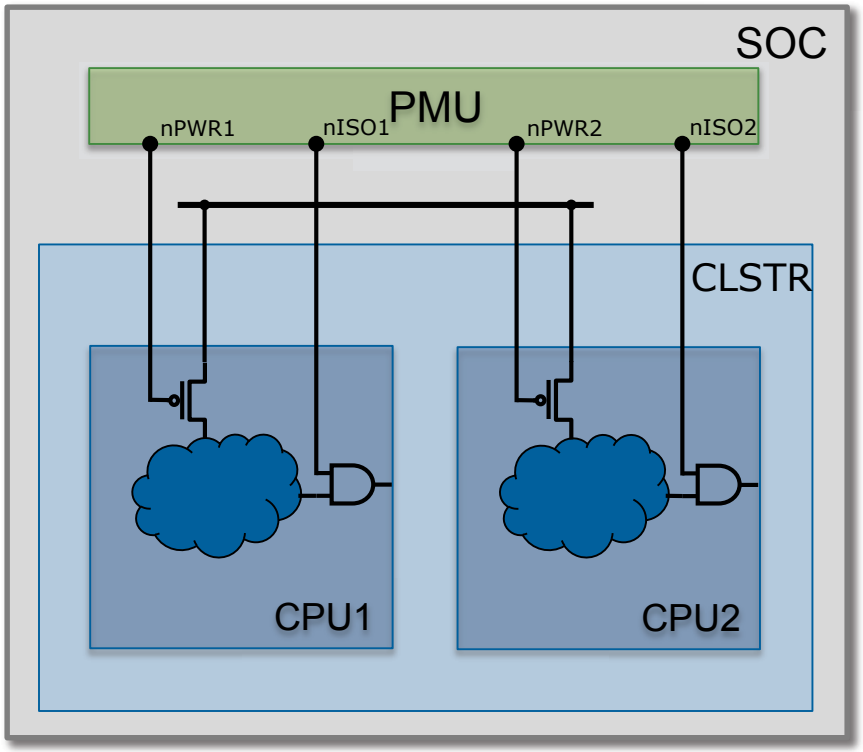
```
set_scope /SOC
load_upf cpu_cnstr.upf -scope CLSTR/CPU1
load_upf cpu_cnstr.upf -scope CLSTR/CPU2
load_upf clstr_cnstr.upf -scope CLSTR
```

- **Isolation**

```
set_isolation ISO -domain PD_CPU1
    -isolation_signal PMU/nISO1
    -location self
```

- **Power switches**

```
create_power_switch SW -domain PD_CPU1
    -input_supply_port    {sw_in    VDDSOC}
    -output_supply_port   {sw_out   VDDCPU1}
    -control_port         {sw_ctl   PMU/nPWR1}
    -on_state  {on_state   sw_in {!PMU/nPWR1}}
    -off_state {off_state        { PMU/nPWR1}}
```



Flat vs Hierarchical

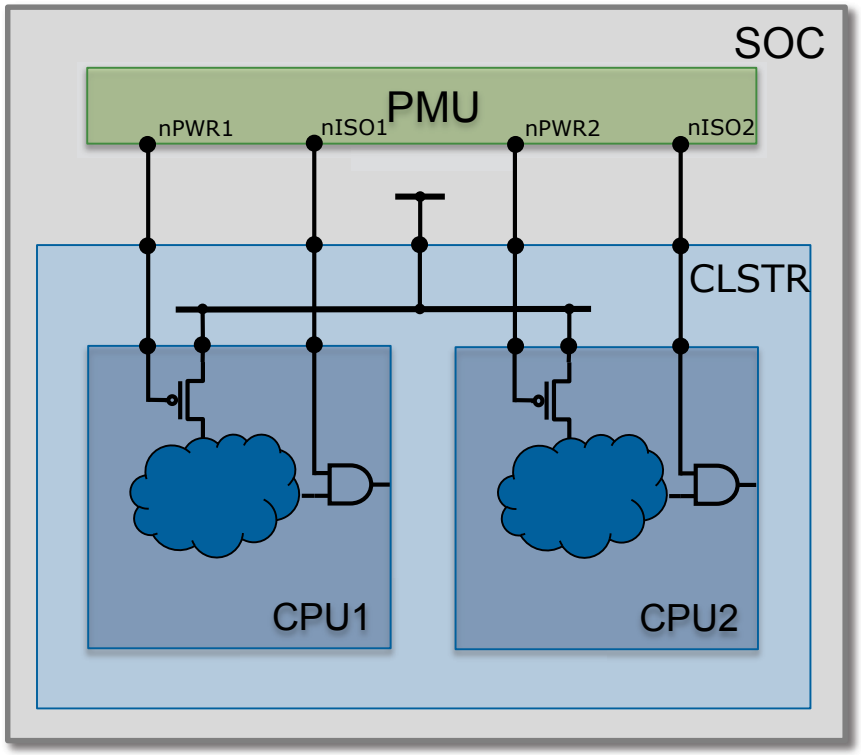
- **Configure IP out of context**

```
create_logic_port nISO -direction in
create_logic_port nPWR -direction in
set_isolation ISO -domain PD1
    -isolation_signal nISO
create_power_switch SW -domain PD
    -control_port {sw_ctl nPWR}
```

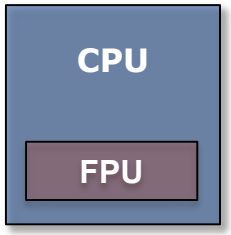
- **Load configured IP in to context**

```
set_scope /CLSTR
load_upf cpu_config.upf -scope CPU1
load_upf cpu_config.upf -scope CPU2
create_logic_port nISO1
create_logic_port nPWR1
connect_logic_net CPU1/ISO -port nPWR1
connect_logic_net CPU1/ISO -port nISO1
```

- **Connect up to PMU**



CPU Configuration



- **Compose PD_CPU and PD_FPU in to single domain**

```
create_composite_domain PD_myCPU -subdomains {PD_CPU PD_FPU}
```

PD_FPU not required

- **Create power control ports**

```
create_logic_port nPWRUP_CPU    -direction in
create_logic_port nISOLATE_CPU  -direction in
```

- **Create isolation strategies to fulfill isolation requirements**

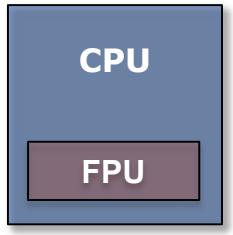
```
set_isolation ISO_LO -domain PD_myCPU \
    -applies_to outputs -clamp_value 0 \
    -isolation_signal nISOLATE_CPU -isolation_sense low \
    -location self
```

Clamp all outputs low by default

```
set_isolation ISO_HI -domain PD_myCPU \
    -elements "$CPU_CLAMP1" -clamp_value 1 \
    -isolation_signal nISOLATE_CPU -isolation_sense low \
    -location self
```

Clamp the exceptions high (more specific overrides more generic)

CPU Configuration



- **Update power supply state with supply expressions**

```
add_power_state PD_myCPU.primary -supply -update\
    -state {ON  -supply_expr {power == FULL_ON  && ground == FULL_ON }}\
    -state {OFF -supply_expr {power == OFF      || ground == OFF     }}
```

- **Update power domain state with logic expressions**

```
add_power_state PD_CPU -domain -update \
    -state {RET -illegal} \
    -state {RUN -logic_expr {!nPWRUP_CPU}} \
    -state {OFF -logic_expr { nPWRUP_CPU}}
```

CPU state retention not required

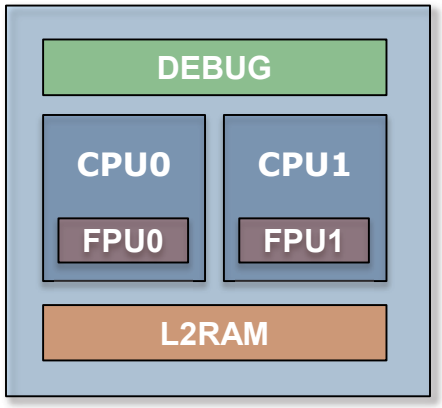
```
add_power_state PD_FPU -domain -update \
    -state {RUN -logic_expr {!nPWRUP_CPU}}
    -state {OFF -logic_expr { nPWRUP_CPU}}
```

PD_FPU is switched by nPWRUP_CPU

```
add_power_state PD_myCPU -domain -update \
    -state {RUN -logic_expr {PD_CPU = RUN && PD_FPU == RUN}
    -state {OFF -logic_expr {PD_CPU = OFF && PD_FPU == OFF}
```

Express PD_myCPU state in terms of PD_CPU & PD_FPU

Cluster Configuration



Create Cluster power control ports

```
create_logic_port nPWRUP_CLSTR     -direction in
create_logic_port nISOLATE_CLSTR   -direction in

create_logic_port nPWRUP_L2RAM     -direction in
create_logic_port nISOLATE_L2RAM   -direction in
create_logic_port nRETAIN_L2RAM    -direction in

create_logic_port nPWRUP_DEBUG     -direction in
create_logic_port nISOLATE_DEBUG   -direction in
```

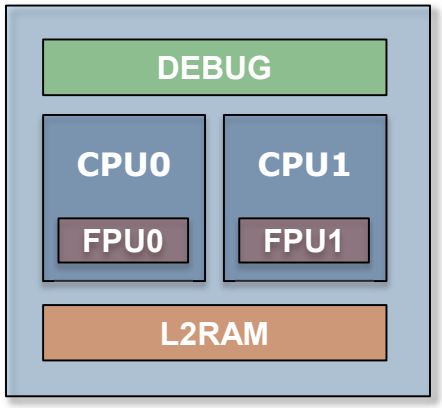
Create CPU power control ports

```
create_logic_port nPWRUP_CPU0      -direction in
create_logic_port nISOLATE_CPU0    -direction in

create_logic_port nPWRUP_CPU1      -direction in
create_logic_port nISOLATE_CPU1    -direction in
```

Connect CPU power control ports

```
connect_logic_net nPWRUP_CPU0   -port uCPU0/nPWRUP_CPU0
connect_logic_net nISOLATE_CPU0 -port uCPU0/nISOLATE_CPU0

connect_logic_net nPWRUP_CPU1   -port uCPU1/nPWRUP_CPU1
connect_logic_net nISOLATE_CPU1 -port uCPU1/nISOLATE_CPU1
```

Cluster Configuration



- **Create retention strategy for L2RAM**

```
set_retention PD_L2RAM \
    -save_signal         {nRETAIN_L2RAM posedge} \
    -restore_signal      {nRETAIN_L2RAM negedge} \
    -retention_condition {nRETAIN_L2RAM} \
```

Single pin retention control

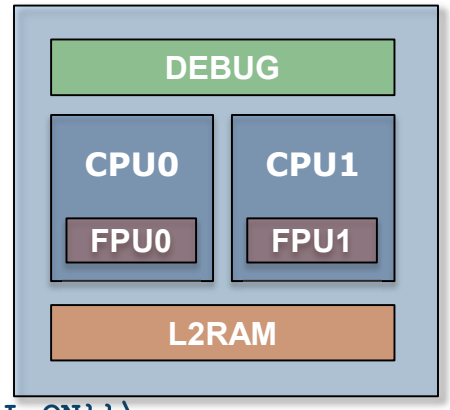
- **Create isolation strategies to fulfill isolation requirements**

```
set_isolation ISO_LO -domain PD_CLSTR -clamp_value 0 \
    -isolation_signal nISOLATE_CPU -isolation_sense low \
    -location self
set_isolation ISO_HI -domain PD_CLSTR -elements "$CLSTR_CLAMP1" -clamp_value 1
    -isolation_signal nISOLATE_CPU -isolation_sense low \
    -location self
set_isolation ISO_LO -domain PD_L2RAM -clamp_value 0 \
    -isolation_signal nISOLATE_L2RAM -isolation_sense low \
    -location self
set_isolation ISO_LO -domain PD_DEBUG -clamp_value 0 \
    -isolation_signal nISOLATE_DEBUG -isolation_sense low \
    -location self
```

Clamp all outputs low by default

Clamp the exceptions high (more specific overrides more generic)

Cluster Configuration



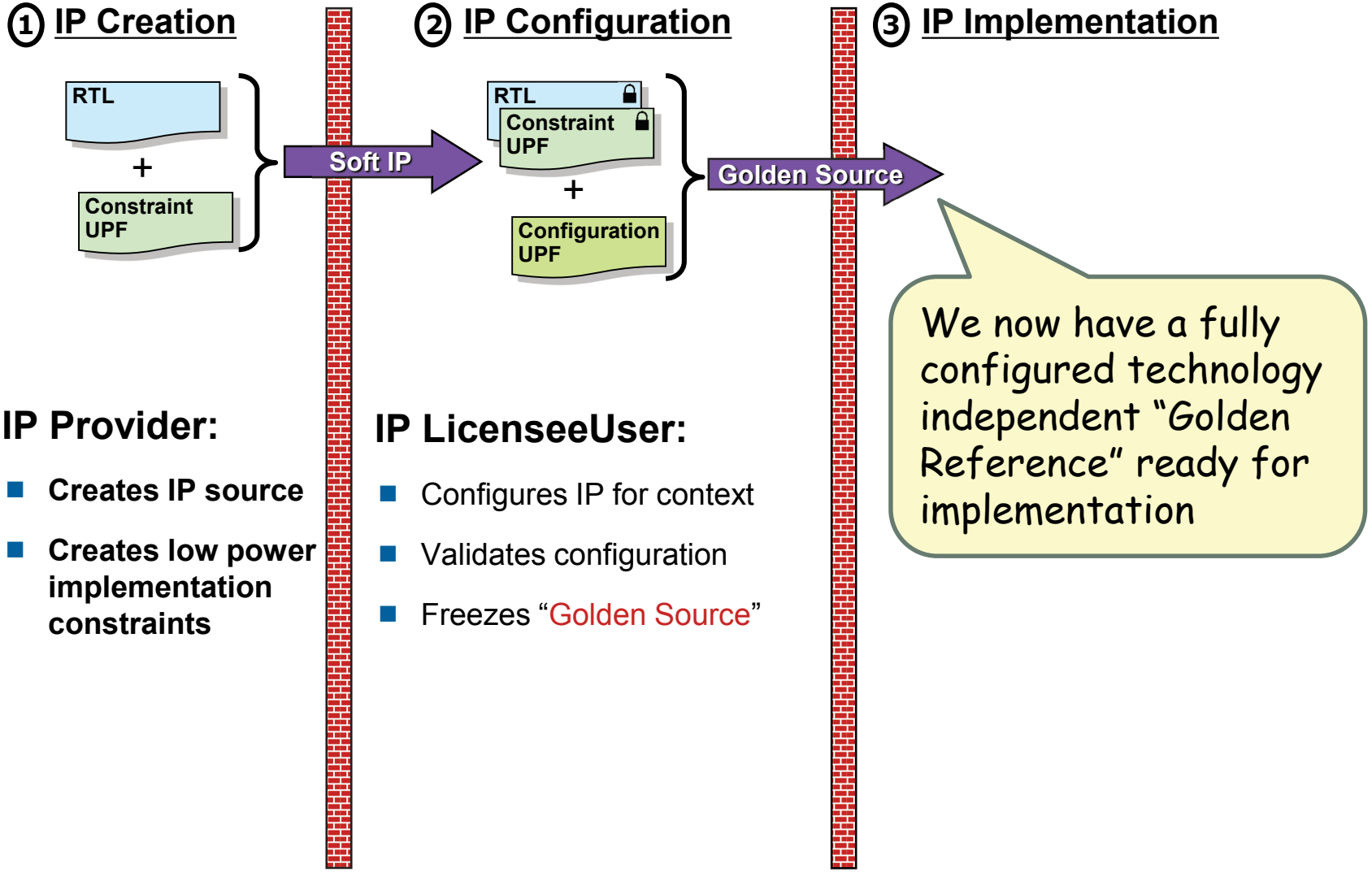
- Update power supply state with supply expressions

```
add_power_state PD_CLSTR.primary -supply -update \
    -state {ON  -supply_expr {power == FULL_ON  && ground == FULL_ON}}\
    -state {OFF -supply_expr {power == OFF      || ground == OFF    }}
add_power_state PD_L2RAM.primary -supply -update ...
add_power_state PD_L2RAM.default_retention -supply -update ...
add_power_state PD_DEBUG.primary -supply -update ...
```

- Update power domain state with logic expressions

```
add_power_state PD_L2RAM -domain -update \
    -state {RUN -logic_expr {!nPWRUP_L2RAM && !nRETAIN_L2RAM}} \
    -state {RET -logic_expr { nPWRUP_L2RAM &&  nRETAIN_L2RAM}} \
    -state {OFF -logic_expr { nPWRUP_L2RAM }}
add_power_state PD_DEBUG -domain \
    -state {RUN  -logic_expr {!nPWRUP_DEBUG}} \
    -state {OFF  -logic_expr { nPWRUP_DEBUG}}
add_power_state PD_CLSTR -domain \
    -state {RUN  -logic_expr {!nPWRUP_CLSTR}} \
    -state {RET  -logic_expr {!nPWRUP_CLSTR}} \
    -state {DMT  -logic_expr { nPWRUP_CLSTR}} \
    -state {OFF  -logic_expr { nPWRUP_CLSTR}}
```

Successive Refinement of Power Intent



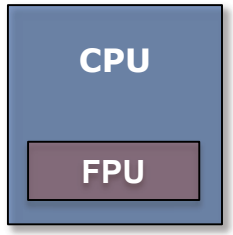
IP Provider:

- **Creates IP source**
- **Creates low power implementation constraints**

IP LicenseeUser:

- Configures IP for context
- Validates configuration
- Freezes "Golden Source"

CPU Implementation



- Create supply nets and update supply set functions

```
create_supply_net VDD
create_supply_net VDD_CPU
create_supply_net VSS

create_supply_set PD_myCPU.primary -update\
    -function {power VDD_CPU} -function {ground VSS}

create_supply_set PD_myCPU.default_isolation -update \
    -function {power VDD}     -function {ground VSS}
```

Use `VDD_CPU` for primary power

Use `VDD` for isolation power

- Map the isolation strategies on to specific library cells

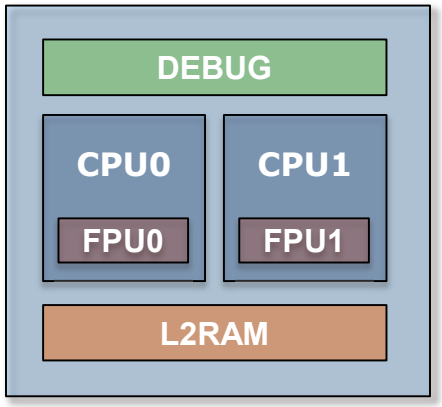
```
use_interface_cell CPU_LO -strategy ISO_LO -domain PD_CPU -lib_cells "$ISO_LO"
use_interface_cell CPU_HI -strategy ISO_HI -domain PD_CPU -lib_cells "$ISO_HI"
```

- Create a switch to fulfill the power state

```
create_power_switch SW_CPU -domain PD_myCPU \
    -input_supply_port   {sw_in     VDD} \
    -output_supply_port  {sw_out    VDD_CPU} \
    -control_port        {sw_ctl    nPWRUP_CPU} \
    -on_state  {on_state  sw_in  {!sw_ctl}} \
    -off_state {off_state        {sw_ctl}}
```

Switch drives `VDD_CPU` with `VDD` when `nPWRUP_CPU` is low

Cluster Implementation



- Create supply nets and update supply set functions

```
create_supply_net VDD
create_supply_net VDD_CLSTR
create_supply_net VDD_L2RAM
create_supply_net VDD_DEBUG
create_supply_net VSS

create_supply_set PD_CLSTR.primary -update \
    -function {power VDD_CLSTR} -function {ground VSS}

create_supply_set PD_CLSTR.default_isolation -update \
    -function {power VDD}      -function {ground VSS}

create_supply_set PD_L2RAM.primary -update \
    -function {power VDD_L2RAM} -function {ground VSS}

create_supply_set PD_L2RAM.default_isolation -update \
    -function {power VDD} -function {ground VSS}

create_supply_set PD_L2RAM.default_retention -update \
    -function {power VDD} -function {ground VSS}

create_supply_set PD_DEBUG.primary -update \
    -function {power VDD_CLSTR} -function {ground VSS}

create_supply_set PD_DEBUG.default_isolation -update \
    -function {power VDD}      -function {ground VSS}
```

Use VDD_CLSTR for primary power

Use VDD for isolation power

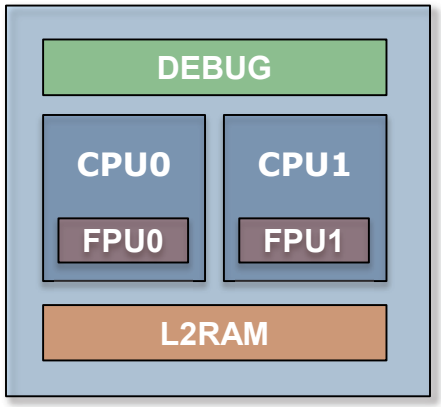
Use VDD for retention power

Cluster Implementation

- Map the isolation strategies on to specific library cells

```
use_interface_cell CLSTR_LO -strategy ISO_LO -domain PD_CLSTR -lib_cells "$ISO_LO"
use_interface_cell CLSTR_HI -strategy ISO_HI -domain PD_CLSTR -lib_cells "$ISO_HI"
use_interface_cell L2RAM_LO -strategy ISO_LO -domain PD_L2RAM -lib_cells "$ISO_LO"
use_interface_cell DEBUG_LO -strategy ISO_LO -domain PD_DEBUG -lib_cells "$ISO_LO"
```

Cluster Implementation

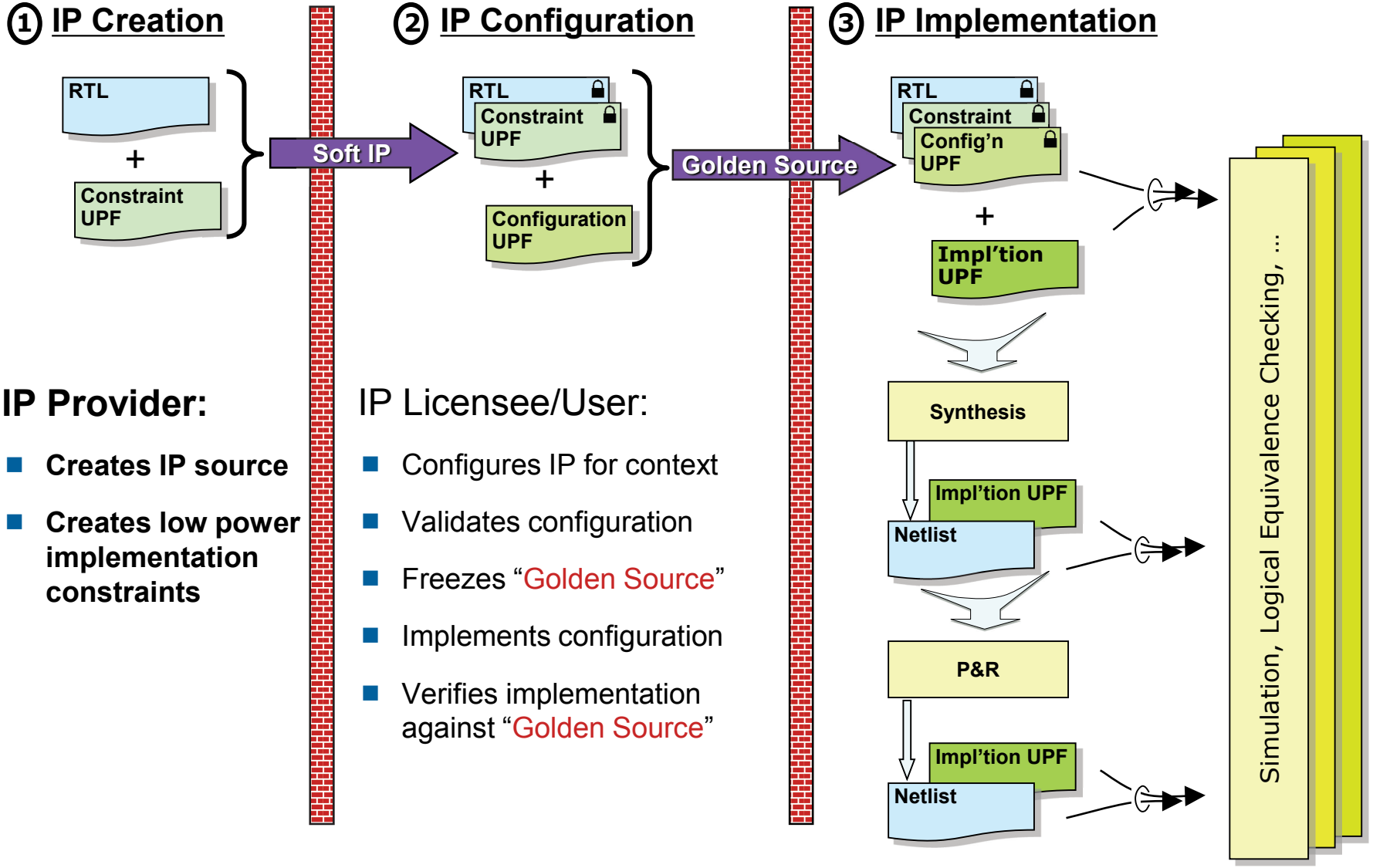


- Create switches to fulfill the power state

```
create_power_switch SW_CLSTR -domain PD_CLSTR \
    -input_supply_port   {sw_in     VDD} \
    -output_supply_port  {sw_out    VDD_CLSTR} \
    -control_port        {sw_ctl    nPWRUP_CLSTR} \
    -on_state  {on_state  sw_in    !sw_ctl}} \
    -off_state {off_state          {sw_ctl}}

create_power_switch SW_L2RAM -domain PD_L2RAM \
    -input_supply_port   {sw_in     VDD} \
    -output_supply_port  {sw_out    VDD_L2RAM} \
    -control_port        {sw_ctl    nPWRUP_L2RAM} \
    -on_state  {on_state  sw_in   {!sw_ctl}} \
    -off_state {off_state          {sw_ctl}}

create_power_switch SW_DEBUG -domain PD_DEBUG \
    -input_supply_port   {sw_in     VDD} \
    -output_supply_port  {sw_out    VDD_DEBUG} \
    -control_port        {sw_ctl    nPWRUP_DEBUG} \
    -on_state  {on_state  sw_in   {!sw_ctl}} \
    -off_state {off_state          {sw_ctl}}
```

Switch drives `VDD_CLSTR` with `VDD` when `nPWRUP_CLSTR` is low

Successive Refinement of Power Intent



IP Provider:

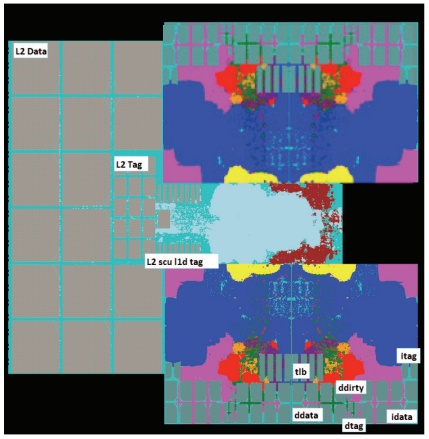
- **Creates IP source**
- **Creates low power implementation constraints**

IP Licensee/User:

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

Hand Off as Hard Macro

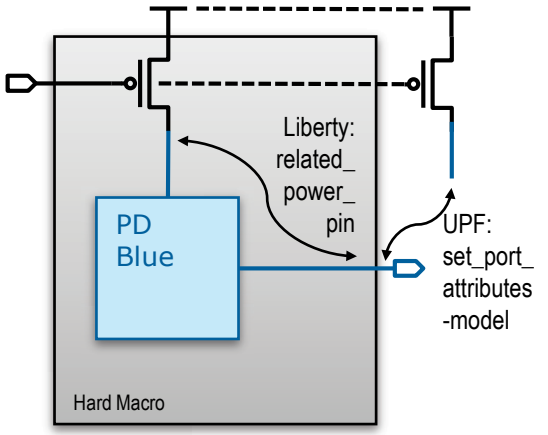
- **No need to re-verify the low power implementation**
  - Just need to verify its low power integration in to the SoC



1. **Power aware simulation model**
   - Corruption and retention behaviours during shutdown
   - Assertions to check correct sequencing of power controls

2. **Liberty model with power/ground pin syntax**
   - related_power_pin, power_down_function etc.

3. **Macro level UPF (descriptive not directive)**
   - “Virtual” switches to “expose” internal supply sets
   - Power states, related power pins, isolation etc.



Hand Off as Hard Macro

- **Improved support for Macro Cell modelling in IEEE1801-2013**

```
begin_power_model CLSTR
  create_power_domain PD_CLSTR -elements {.}
  create_supply_set PD_CLSTR.primary -update -function {power VDD} -function {ground VSS}
  add_power_state PD_CLSTR -domain
    -state {RUN -logic_expr {primary == DEFAULT_NORMAL &&
                (!nPWRUP_CPU0 || !nPWRUP_CPU1) && !nPWRUP_L2RAM &&  nRETAIN_L2RAM}}
    -state {RET -logic_expr {primary == DEFAULT_NORMAL &&
                ( nPWRUP_CPU0 &&  nPWRUP_CPU1) && !nPWRUP_L2RAM && !nRETAIN_L2RAM}}
    -state {DMT -logic_expr {primary == DEFAULT_NORMAL &&
                ( nPWRUP_CPU0 &&  nPWRUP_CPU1) && !nPWRUP_L2RAM &&  nRETAIN_L2RAM}}
    -state {DBG -logic_expr {primary == DEFAULT_NORMAL && !nPWRUP_DGB}
    -state {OFF -logic_expr {primary == DEFAULT_CORRUPT}
end_power_model

apply_power_model CLSTR -elements uCLSTR -supply_map {PD_CLSTR.primary SS1}
```

- **Alternatively just use the original RTL+UPF to model Hard Macro**

SoC-Level Design and Verification

Sushma Honnavara-Prasad

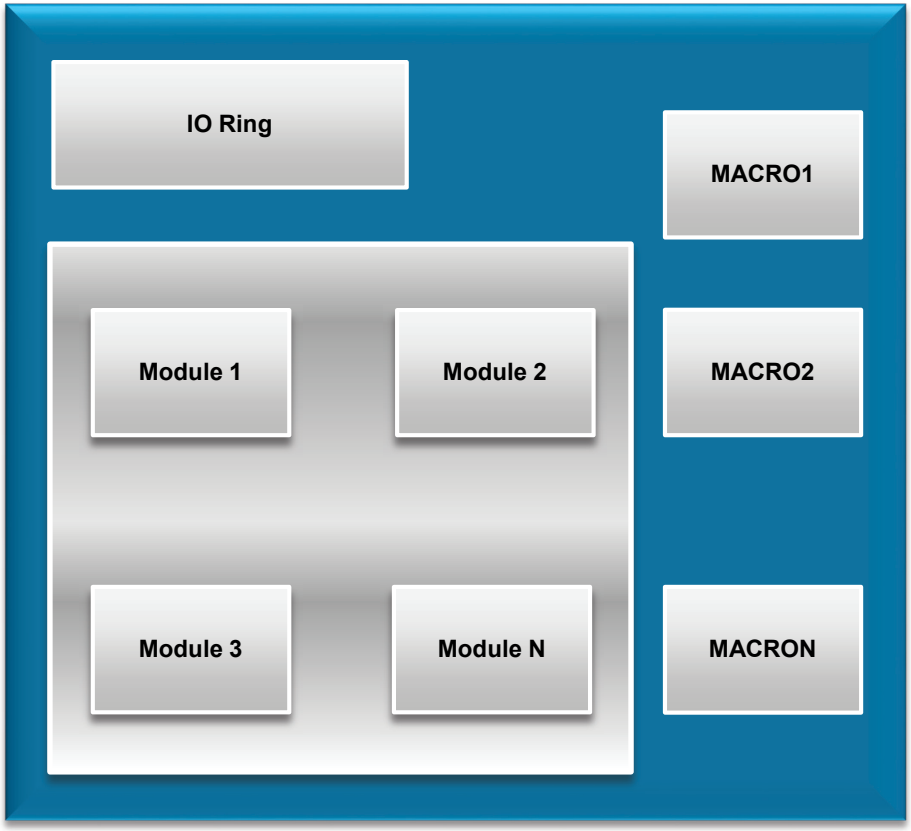
Principal Engineer

Broadcom

SoC Power Intent Specification

Introduction

- A typical SoC contains:
  - Hard IP (fully implemented macros)
  - Soft IP (HDL integrated into top level)
  - Analog/mixed signal 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



SoC UPF 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 -elements { . }
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
.....						

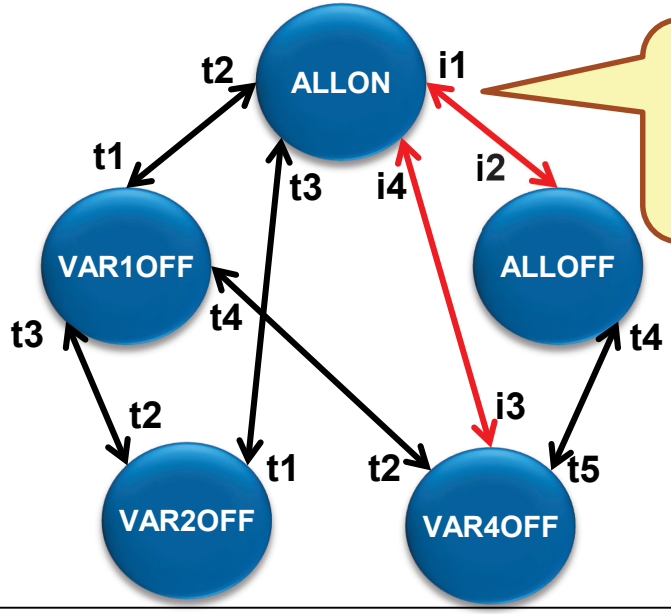
State	ioss	aonss	var1ss	var2ss	var3ss	var4ss
on	nom	nom	!off	!off	nom | turbo	!off
var1off	nom	nom	off	!off | off	nom | turbo	!off
var2off	nom	nom	!off	off	nom | turbo	!off
var4off	nom	nom	off	!off | off	nom | turbo	off
alloff	nom	nom	off	off	nom | turbo	off

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: i1 is a transition to ALLON state from ALLOFF state

```
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}
```

SoC UPF Integration Tips

Summary

- Design/UPF Partitioning
- Modularization
- Mixed language handling
- IO modeling
- Macro handling

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
```

Mixed Language Handling

- **Be aware of HDL case sensitivity**
  - Verilog/VHDL (and UPF) – case sensitive
  - VHDL – case insensitive
  - Issue: This particular affects how we write a UPF for a piece of HDL code
    - find_objects – by default returns case sensitive match patterns
- **Bus notation differences**
  - Verilog/System Verilog – square bracket
  - VHDL – parenthesis
- **Tip**
  - Keeping the domain boundary in one HDL type helps keep the UPF simple.

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_inst_list [find_objects . -pattern "PADRING_*" \
            -object_type model -transitive]
foreach pad_inst $pad_inst_list {
      connect_supply_net pad_ring_VSS -ports "$pad_inst/VSSP"
}
```

- **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 }
```

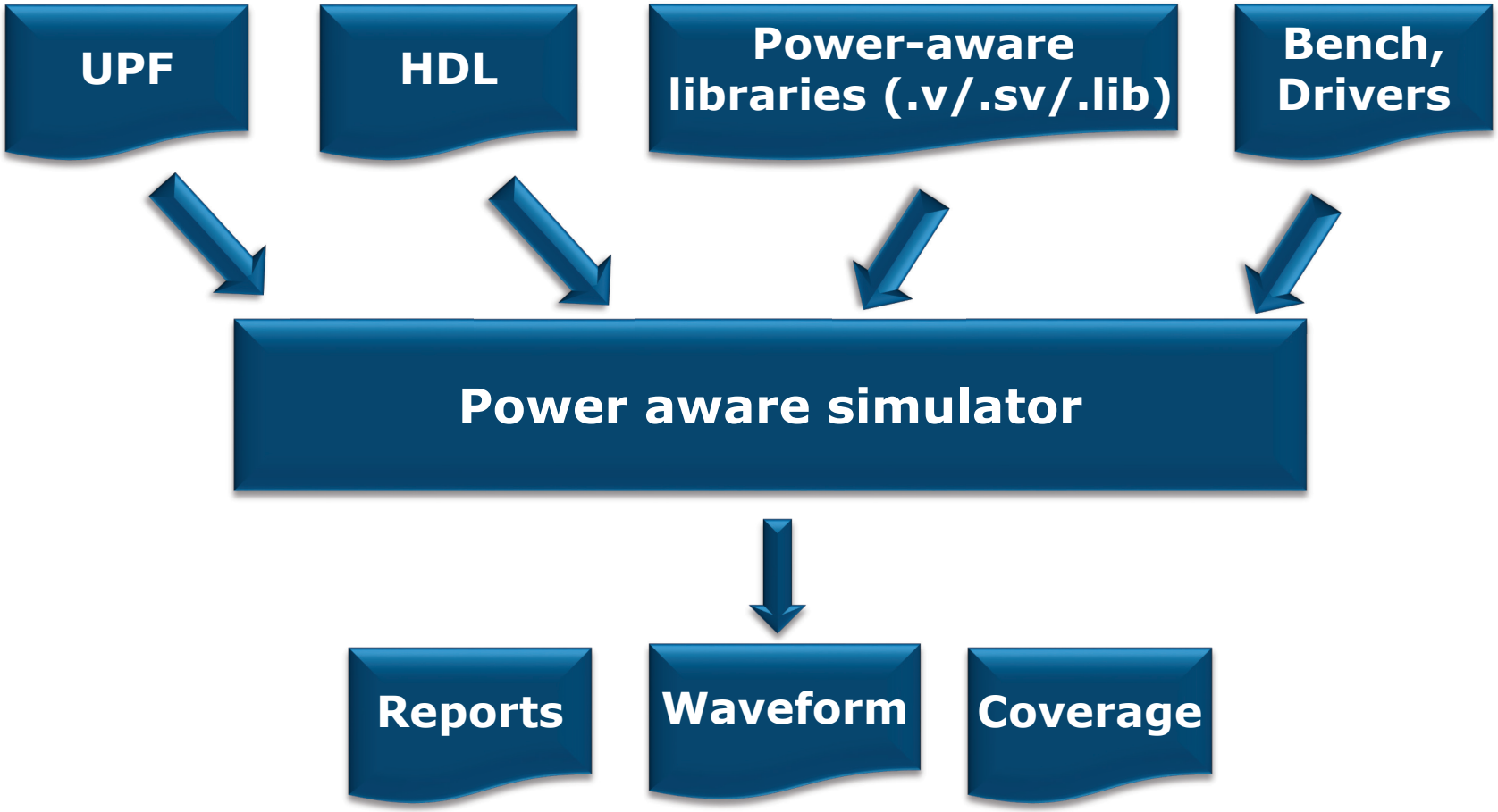
Handling Macros

- **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_list [find_objects . -pattern *u_pll* -object_type inst \
                   -leaf_only -transitive]
foreach inst $pll_inst_list {
       connect_supply_net 1p8ss.power  -ports "$inst/AVDD1P8"
       connect_supply_net 1p8ss.ground -ports "$inst/AVSS"
}
```

SoC Verification

Power Aware Simulation

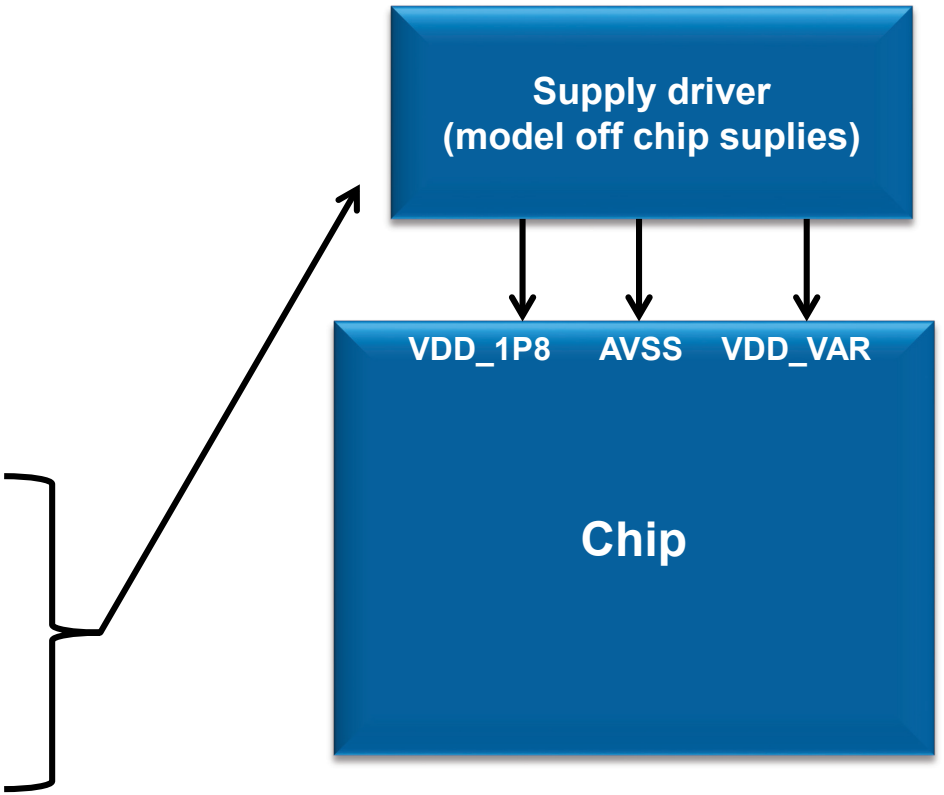


SoC Test-bench UPF

```
# Set the scope to the test bench
set_design_top top/chip_tb_inst

# Load chip upf
load_upf $env(UPF_PATH)/chip/upf/chip.upf -scope u_chip

# Global settings -treat all partial on as OFF
set_partial_on_translation OFF


# Simstate settings - simulation behavior for a model/library
 load_simstate_behavior MACRO_LIB -file macro_lib_simsate.upf
 set_simstate_behavior ENABLE -lib ANA_PLL_LIB -model ANA_PLL
 ....
```

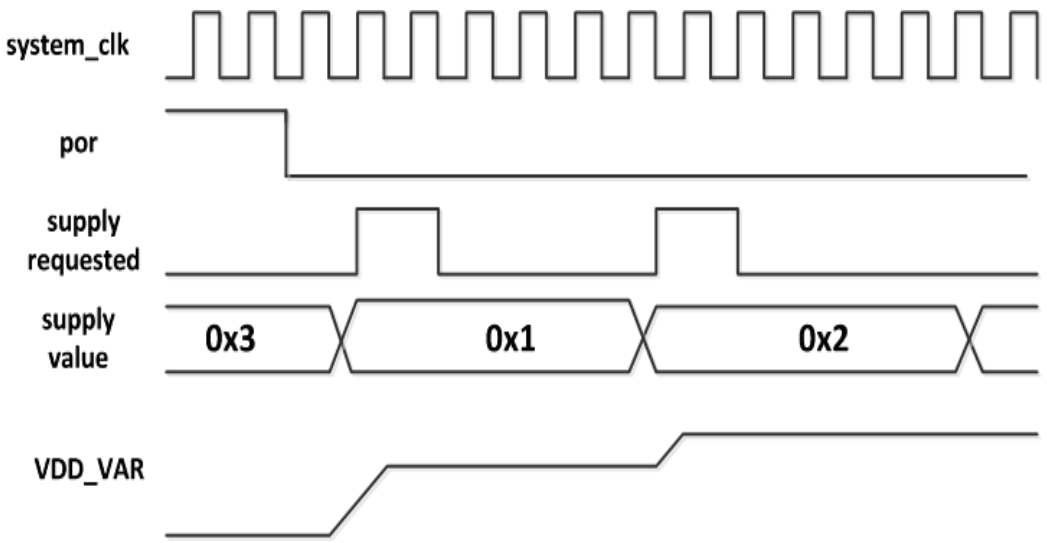
SoC Test-bench

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



SoC Test-bench

```
// Dynamic supplies
always @ (posedge system_clk, negedge por)
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
```



Note: system_clk, por and supply_value are signals in the test bench, not UPF objects

Adopting UPF

Sushma Honnavara-Prasad

Principal Engineer

Broadcom

Recommendations for Adopting UPF

Draw a power diagram

- A color coded domain hierarchy illustrating picture
- Visually describes the domains, relationships, isolation locations, power connectivity etc

Enable incremental UPF

- Initial UPF could leave out implementation details
- UPF can be refined during the flow

Know your libraries

- Understanding what is implementable with a given set of libraries is useful
- A rich library set could allow various implementations, the user can choose the most optimal combination

Recommendations for Adopting UPF

Unleash the power of Tcl

- Tcl variables, foreach loops, procs could help making the UPF compact and readable
- For example: find_objects can return a list which a foreach loop could consume to do a certain operation

Enable UPF reuse

- Try to keep the UPF technology agnostic if possible, this way, it can be paired with technology agnostic RTL and reused.

Avoid vendor specific constructs

- Use of tool specific commands in a UPF file is not allowed per standard. It is possible vendors might allow this, but it would make the UPF non-portable.
- Use of a pure UPF based solution is recommended

Recommendations for Adopting UPF

Avoid hardcoding paths

- If loading other UPF files or helper tcl files, avoid hard-coding the path. Try using a configurable env variable instead.
- Avoid hardcoding HDL hierarchical paths if possible by using find_objects (allows use of same UPF in spite of hierarchy changes)

Keep it readable

- Add comments wherever possible. UPF is like HDL, the author's explanation in the form of comments might be useful to the end user
- Indent command options appropriately to make them readable.

Modularize

- Try to break up the UPF into functional boundaries if possible instead of having a long file. Again, think how we would write RTL. It is never easy to debug a big-flat design with no hierarchies.

Adopting UPF*

Shreedhar Ramachandra

Staff Engineer

Synopsys

* Slides contributed by Jon Worthington

Recommendations for Adopting UPF

- **1) Determine the domains needed by identifying areas of the design that will have different supply needs.**
  - Sometimes cells that have the same supply requirements may actually benefit to being in a unique domain if there are other reasons to treat them differently. Such as if they reside in a very different physical regions of the chip, or require a different set of strategies or power states to apply to them.
  - Consider the hierarchy connections between cells of the same domain. Should they really cross a different domain just because they need to traverse logic hierarchy? An extra level of hierarchy my help here.

Recommendations for Adopting UPF

- **Make sure you build a UPF that is common to both your verification and implementation flow**
  - you want to verify what you are implementing.

- **Build the complete power state table –**
  - In verification you check for the correctness and completeness of your power state table and in implementation, static checking you take the power state table as golden for all the optimization. So start with a complete PST.

- **Analyze your protection requirements early**
  - so that you can build generic isolation policies that still give you optimum results without redundant isolation cells.

- **Check your libraries.**
  - You should have libraries characterized at all corners based on all the combinations of voltages on the supply nets to make sure that the power intent is implementable and that you have protection cells to meet the requirements of your power intent.

Recommendations for Adopting UPF

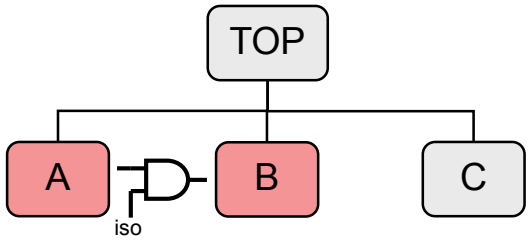
- **Make sure the power architecture your implement will actually save power.**
  - Ensure the saved power outweighs the additional power consumed by iso cells, level shifters, and power switches

- **Make sure the power intent you create aligns with the physical implementation of your design.**
  - As an example, at the RTL level ISO cells and LS can be specified in a wide range of locations. But in the physical implementation, these cells will require special power connections that can have real impact on the physical design.

Tip: Align Power Domains and Logic Hierarchy

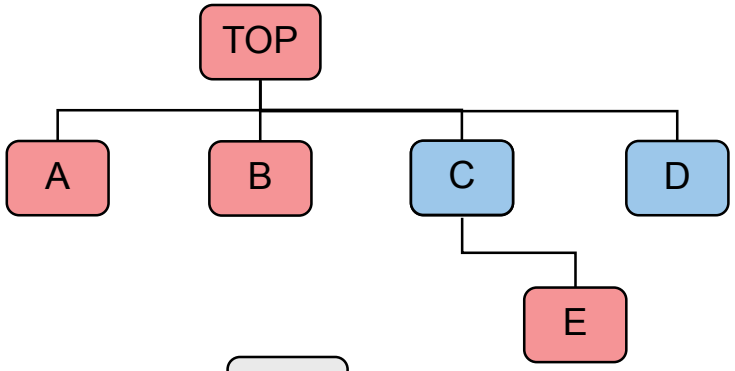
- Multi-element power domains can lead to unexpected “intra-domain” isolation

```
create_power_domain RED -elements {A B}
```



- These can often be avoided with a different approach

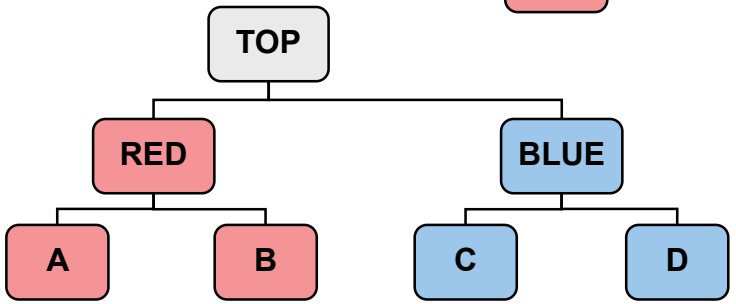
```
create_power_domain RED -elements {.}
create_power_domain BLUE -elements {C}
```



- Better to align power domains with logic hierarchy if at all possible

```
create_power_domain RED  -elements {RED}
create_power_domain BLUE -elements {BLUE}
```

Failing that, use `-diff_supply_only` option or the `-source/-sink` filters



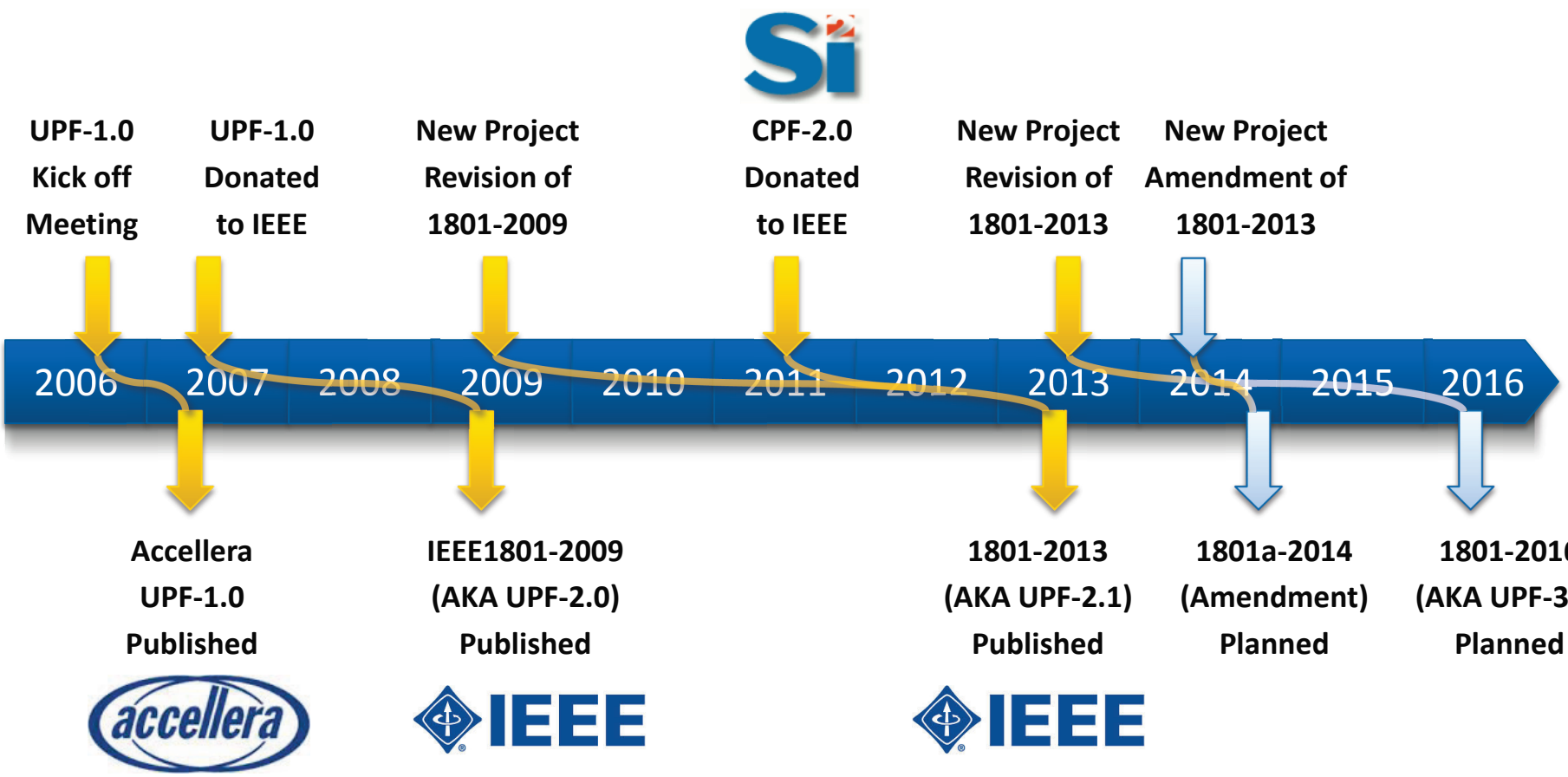
Where We Go From Here

John Biggs

Senior Principal Engineer

ARM

IEEE 1801 (UPF) timeline



P1801 Work Group Plans

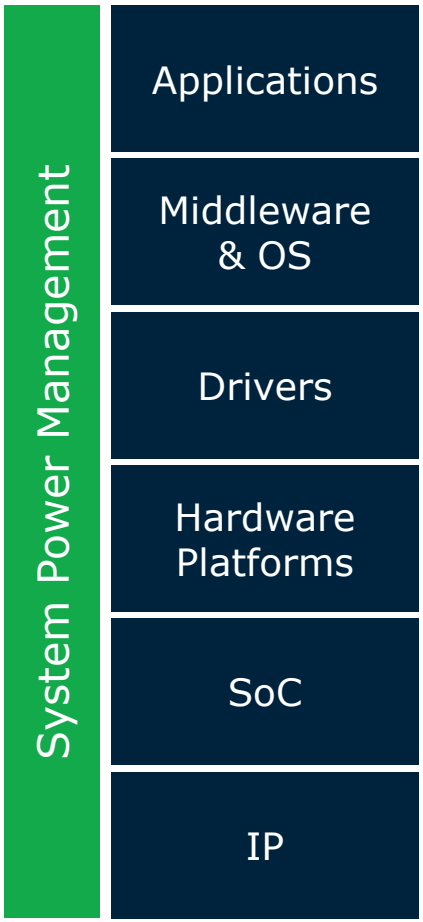
- **1801-2013 Amendment PAR (2014)**
  - Project on the agenda for approval at the March 2014 IEEE-SA board meeting
  - Correct technical/editorial errors and inconsistencies
  - Address a small number of critical enhancements

- **1801 Full Revision PAR (2015/16)**
  - Project approved at the June 2013 IEEE-SA board meeting
  - Extend scope of "Power Intent" up towards System Level
  - Add power modeling and estimation capabilities
    - SAIF integration and extension
  - Consider further UPF/CPF methodology convergence
  - Enhance and extend Low Power Methodology Annex

System Level Power Intent

- Depends on perspective
  - **SW Centric:** abstract, task, transaction/event based
  - **HW Centric:** detailed, component, state/level based
- Bridge the gap
  - **Top down:** Add detail to the abstract SW centric world of system performance modelling.
  - **Bottom up:** Add abstraction to detailed HW centric world of RTL+UPF implementation
- Extend UPF as far as is appropriate
  - Raise abstraction level of "power intent"
  - Need a better understanding of the system level requirements.
- Working closely with the IEEE Low Power Study Group
  - Helping to coordinate various standard activities in this area
  - Si2/LPC, Liberty, IP-XACT IEEE1666 SystemC

System Energy Analysis & Optimization



Software Centric Analysis and Optimization:

- Software impacts on energy consumption – trend based analysis
- Detecting and correcting energy bugs in software
- System power management (RFTS/DVFS) optimization
- OS level system and runtime power management bring-up

System Centric Analysis and Optimization:

- Energy aware heterogeneous multi-core architecture exploration
- Energy aware hardware-software partitioning
- Energy aware SoC hardware architecture and exploration
- Memory system topology and interconnect optimization
- Scenario driven power state space optimization
- Power Management Unit (and PMIC) architecture and design
- Dedicated hardware (ASIP) for energy efficiency

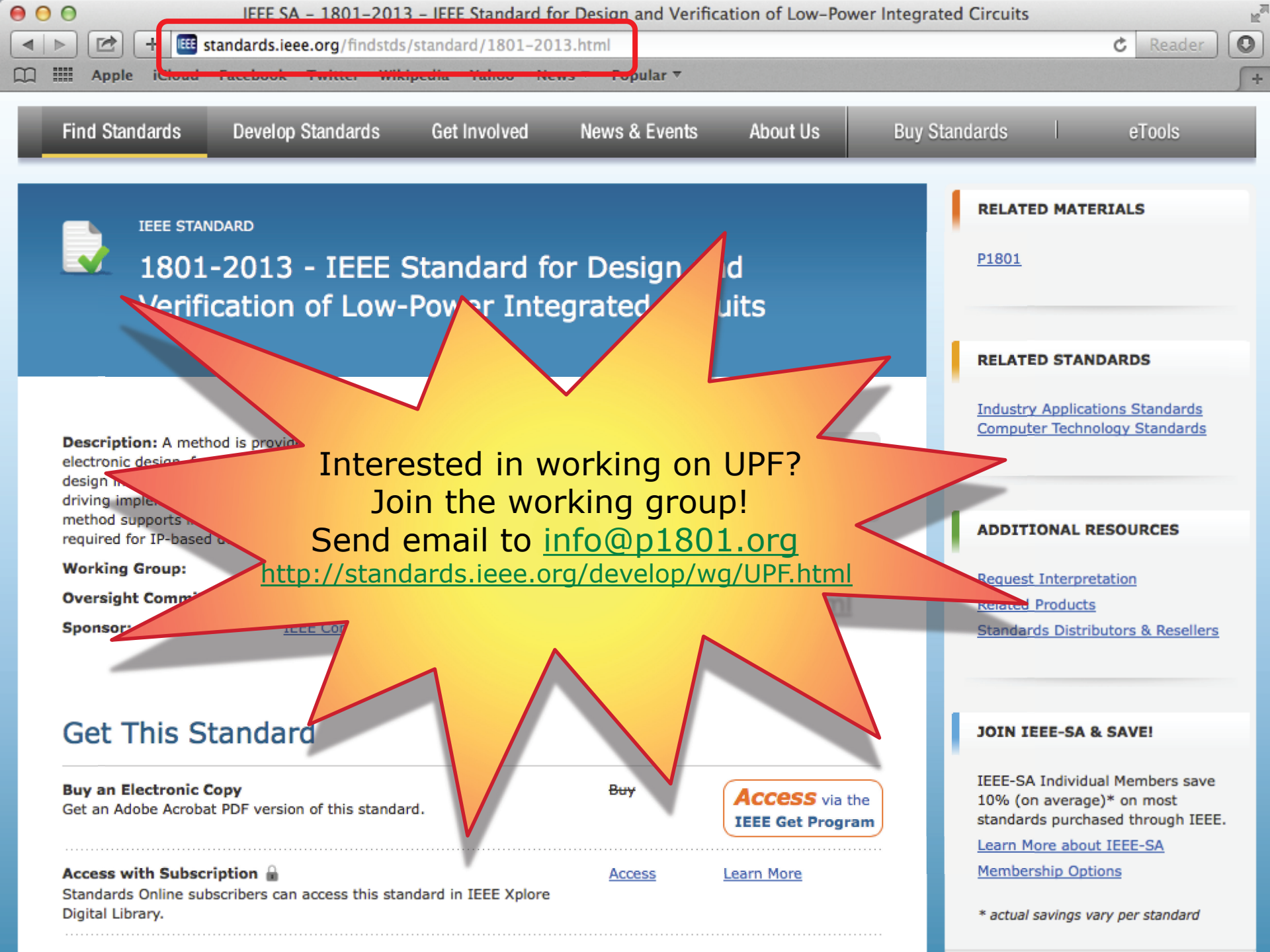
Hardware Centric Optimization:

- Low power design and verification
- Component level power optimization
- Power aware SoC component integration and optimization
- Macro model enhancements for low power design

Source: Alan Gibbons, Synopsys Inc.

System Level Power Subcommittee

- **Formed Feb 2014**
  - 18 people from 11 Entities, Chair: Alan Gibbons, Synopsys
  - Requirements gathering phase
    - Focus on Virtual Prototyping
  - Face2Face meeting April 8/9/10th in UK
- **Near term objectives:**
  - Identify practical use models
  - Determine accuracy and granularity requirements
  - Scope and extent of standardisation?
    - Power model structure, consumption data, activation, interfaces...
  - Evaluate feasibility of extending 1801 to meet requirements
    - Deliver proposal on specific extensions to 1801 Work Group
  - Develop and prototype specific examples


IEEE SA – 1801-2013 – IEEE Standard for Design and Verification of Low-Power Integrated Circuits
standards.ieee.org/findstds/standard/1801-2013.html
Reader
Apple
Find Standards
Develop Standards
Get Involved
News & Events
About Us
Buy Standards
eTools
IEEE STANDARD
1801-2013 - IEEE Standard for Design
Verification of Low-Power Integrated
uits
Description: A method is provid
electronic design
design
driving imple
method supports
required for IP-based
Working Group:
Oversight Comm
Sponsor:
Interested in working on UPF?
Join the working group!
Send email to info@p1801.org
http://standards.ieee.org/develop/wg/UPF.html
Get This Standard
Buy an Electronic Copy
Get an Adobe Acrobat PDF version of this standard.
Buy
Access via the IEEE Get Program
Access with Subscription
Standards Online subscribers can access this standard in IEEE Xplore Digital Library.
Access
Learn More
RELATED MATERIALS
P1801
RELATED STANDARDS
Industry Applications Standards
Computer Technology Standards
ADDITIONAL RESOURCES
Request Interpretation
Related Products
Standards Distributors & Resellers
JOIN IEEE-SA & SAVE!
IEEE-SA Individual Members save 10% (on average)* on most standards purchased through IEEE.
Learn More about IEEE-SA
Membership Options
* actual savings vary per standard