Low-power Design with the New IEEE 1801-2013 Standard
Workshop #4: presented by members of the IEEE P1801 WG

John Biggs
Jeffrey Lee
Erich Marschner
Sushma Honnavara-Prasad
Qi Wang
Agenda

- Introduction
  - Modeling Power Intent with IEEE 1801
  - New Features in IEEE 1801-2013
    ~ Break at approx. 2:50 ~
  - Modeling Power Management Cells and Hard Macros
  - Low Power Design Methodology for IP Providers
  - Low Power Design Methodology for SoC Designers

John Biggs, ARM Ltd. and P1801 WG chair

Jeffrey Lee, Synopsys, Inc.

Erich Marschner, Mentor Graphics Corp. and P1801 WG vice-chair

Qi Wang, Cadence Design Systems, Inc.

John Biggs, ARM Ltd. and P1801 WG chair

Sushma Honnavara-Prasad, Broadcom Corporation
Introduction

John Biggs
Senior Principal Engineer
ARM
P1801: IEEE-SA Entity Based Work Group

Low-power Design with the New IEEE 1801-2013 Standard

2 June 2013
The New IEEE1801-2013 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 New IEEE1801-2013 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

- “D14” approved by IEEE-SA March 6th 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

**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.

**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.

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

**STATUS:**
Active Standard

---

**RELATED MATERIALS**

- PAR

**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
P1801 Work Group Plans

- **1801-2015 PAR (Project Authorization Request)**
  - On the agenda for approval at June IEEE-SA NesCom meeting

- **Motivation**
  - Extend scope of “Power Intent” up to 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

- **Interested in working on UPF? Join the working group!**
  - Send an email to info@p1801.org for details
Agenda

- Introduction
- Modeling Power Intent with IEEE 1801
- New Features in IEEE 1801-2013
  ~ Break ~
- Modeling Power Management cells and Hard Macros
- Low Power Design Methodology for IP Providers
- Low Power Design Methodology for SoC Designers
Modeling Power Intent with IEEE 1801

Jeffrey Lee
Staff Corporate Application Engineer
Synopsys
### 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**

© 2013 Synopsys
UPF is

- An Evolving Standard
  - Accellera UPF 2007 (1.0)
  - IEEE 1801-2009 UPF (2.0)
  - IEEE 1801-2013 UPF (2.1)

- Used to describe Power Intent
  - Define power management
  - Used as a method to minimize power consumption

- Based upon Tcl
  - Tcl syntax and semantics
  - Can be mixed with non-UPF Tcl

- Used with HDLs
  - SystemVerilog, Verilog, VHDL

- For Verification and Implementation
  - Simulation or Emulation
  - Static/Formal Verification
  - Logic Synthesis
  - DFT
  - Place and Route
  - etc
Assumptions

- Knowledgeable about low power design

- Know IEEE 1801-2009 (UPF)

- This session is not intended to teach you UPF
  - Discuss what is new in 1801-2013
Power Domain

- Power domains are defined for a logical hierarchy

```plaintext
create_power_domain PD_Proc \\
   -include_scope

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

- All cells instantiated within the domain are considered part of the power domain unless explicitly placed within a child power domain
Power Domain

- Creation of a power domain also provides 3 pre-defined supply set handles to help with the abstraction of your supply network
  - primary
  - default_isolation
  - default_retention

- All cells instantiated within the domain will have their primary power and ground connected to the power_domain.primary unless the pg pin is explicitly connected to another supply
Supply sets and supply set handles are a bundle of supply nets

- Provides an abstraction and allows designers to define their power intent without having to create the actual supply nets
  - Supply nets will still be needed for physical implementation

- 6 supply set functions

<table>
<thead>
<tr>
<th>power</th>
<th>pwell</th>
<th>deeppwell</th>
</tr>
</thead>
<tbody>
<tr>
<td>ground</td>
<td>nwell</td>
<td>deepnwell</td>
</tr>
</tbody>
</table>

- Each supply set function represents a supply net
- Can use the functions of supply sets as if they were supply nets
Power Supply Network

- Defining an on-chip switched supply involves defining a UPF representation of the power switch

```
create_power_switch SW \ 
  -output_supply_port {vout VDD_SW} \ 
  -input_supply_port {vin VDD} \ 
  -control_port {ss_ctrl sw_ena} \ 
  -on_state {ss_on vin { ss_ctrl }} \ 
  -off_state {ss_off { !ss_ctrl }}
```

- Use the power, or ground, function of the primary supply set handle with the power switch to model a switched supply
  - Still requires you to define an OFF for the primary supply set
Power States of Supply Sets

- Power states are defined for supply sets (and supply set handles) to describe the characteristics of your power supply network

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

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

- Predefined Simstates
  - NORMAL
  - CORRUPT_STATE_ON_CHANGE
  - CORRUPT_STATE_ON_ACTIVITY
  - CORRUPT_ON_ACTIVITY
  - CORRUPT

- add_power_state -update

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

- Well defined interface specification between your design and the outside world
  - Provides connection points from the outside world to your design
    `create_supply_port VDD`

- Connect your supply network to supply ports via
  
  `connect_supply_net \ PD_Proc.primary.power \ -ports {VDD}`
Isolation Strategies

Isolation strategies for your power domains are defined using the set_isolation command

- If you want implementation tools to implement your isolation strategy using specific technology cells, use the use_interface_cell command

```
set_isolation myISO -domain PD_Mid \
  -applies_to_inputs \ 
  -source PD_Top.primary \ 
  -isolation_control_signal iso_en \ 
  -clamp_value high \ 
  -location self

use_interface_cell myISO \ 
  -domain PD_Mid \ 
  -lib_cells {TechISOX1}
```
Level Shifter Strategies

Level shifter strategies for your power domains are defined using the `set_level_shifter` command
- If you want implementation tools to implement your level shifter strategy using specific technology cells, use the `use_interface_cell` command

```
set_level_shifter myLS \
  -domain PD_Mid \
  -applies_to outputs

use_interface_cell myLS \
  -domain PD_Mid \
  -lib_cells {TechLSX2}
```
Retention Strategies

- Giving sequential elements the ability to retain their state is done via the `set_retention` command
  - Implementation tools need to be told explicitly which technology cells should be used to implement the retention strategy using the `map_retention_cell` command

```plaintext
set_retention myRet \
  -domain PD_Mid \ 
  -elements { \ 
    [find_objects -object_type cell -pattern bus* Mid1]} \ 
  -save_signal ret_save \ 
  -restore_signal ret_restore

map_retention_cell myRet \ 
  -domain PD_Mid \ 
  -lib_cells {TechRRX4}
```
Agenda

- Introduction
- Modeling Power Intent with IEEE 1801
- New Features in IEEE 1801-2013
  ~ Break ~
- Modeling Power Management cells and Hard Macros
- Low Power Design Methodology for IP Providers
- Low Power Design Methodology for SoC Designers
New Features in IEEE 1801-2013

Erich Marschner
Verification Architect
Mentor Graphics

Low-power Design with the New IEEE 1801-2013 Standard
IEEE 1801-2013 UPF Characteristics

- **Precision**
  - Attention to fine detail
  - More precise definitions
  - More fine-grained control

- **Fidelity**
  - Attention to low power design/verification semantics
  - More accurate representation of power aware behavior
  - More focus on library cell definition and usage

- **Incrementality**
  - Attention to process requirements
  - More complete specification of incremental refinement
  - More consistent application of incrementality concepts
Fidelity Enhancements

- **Power Domains**
  - Lower boundary definition
  - Location implications
  - Atomic power domains

- **Power States**
  - Supply/logic expression relationship
  - Supply set power state semantics
  - Additional simstate

- **Strategies**
  - Retention semantics
  - Repeater (buffer) strategy
  - Strategy interactions

- **Supplies**
  - Driver/receiver supply definition
  - Required functions definition
  - Supply equivalence
### Incremental UPF Design Flow

**IP Creation**
- RTL
- + Constraint UPF

**IP Configuration**
- RTL
- + Soft IP
- Constraint UPF
- Configuration UPF

**Golden Source**

**System Implementation**
- RTL
- + Implementation UPF

---

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

**IP Integrator**
- Configures IP for context
- Validates configuration
- Freezes "Golden Source"
- Adds implementation detail
- Implements configuration
- Verifies implementation against "Golden Source"

---

Simulation, Logical Equivalence Checking, …

---

Low-power Design with the New IEEE 1801-2013 Standard

2 June 2013
Incrementality Elaboration

- **Incremental Specification**
  - More specific commands override more generic commands
  - UPF power intent leverages HDL and library attributes
  - System power intent integrates component power intent

- **Incremental Implementation**
  - Synthesis (isolation, level shifting, retention)
  - Test Insertion (more isolation, level shifting)
  - Place & Route (supply network, buffering)

- **Incremental Verification**
  - Consistency checking within a specification
  - Early function verification at RTL/specification stage
  - Later functional verification during implementation stages
Topics

- Power Domain Definition
- Hard Macros
- Driver/Receiver Supplies
- Supply Set Definition
- Related Supply Attributes
- Power States and Simstates
- Power Management Strategies
- Supply Equivalence
- Supply Network Construction
- Power Aware Verification
Low-power Design with the New IEEE 1801-2013 Standard

© 2013 Mentor Graphics
Power Domain Definition

- **Partitioning**
  - `create_power_domain -elements .`
  - Includes all lower instances transitively by default
    - Can carve out subdomains with additional commands

- **Constraining**
  - For IP development, need to
    - Identify potential power domains
    - Constrain implementation to follow this partitioning
  - `create_power_domain -atomic`

**Convention:** *brighter blue signifies new in 1801-2013*

**Replaces** `-include_scope`
Precedence Rules for Power Domains

From lowest to highest:
- Commands that include an element transitively by default
- Commands that explicitly include elements
  - `create_power_domain PD5 -elements {.}`
  - `create_power_domain PD4 -elements {i1 i2 i3}`
  - `create_power_domain PD3 -elements [find_objects ...] ...`
- Commands that create atomic power domains
  - `create_power_domain PD1 -atomic ...`
Atomic Power Domains

- **File Sub.upf**
  ```
  create_power_domain PD_Sub -elements {.}
  load_upf Proc1.upf -scope P1
  load_upf Proc2.upf -scope P2
  ```

- **File Proc1.upf**
  ```
  set_design_top Proc1
  create_power_domain PD_Proc \-elements {.}
  create_power_domain PD_Mem \-elements {M1 M2}
  ```

- **File Proc2.upf**
  ```
  set_design_top Proc2
  create_power_domain PD_Proc \-elements {.} -atomic
  create_power_domain PD_Mem \-elements {M1 M2}
  ```
Using find_objects

New Object Type
- Now able to search for instances of a given module
  - `find_objects -object_type model -pattern ...`

Semantics
- Now a power intent command
  - No longer listed with query functions
  - Searches existing logic hierarchy only, not inferred power management
- Returns a list of explicit names
  - So `create_power_domain` using `find_objects`
    - is interpreted as “specific” rather than “generic”
    - takes precedence over more generic `create_power_domain` commands

Example
- `create_power_domain PD \
  -elements [find_objects . -object_type model -pattern "proc_*"] ...`
Power Domain Boundaries

**UPF 1.0**
- Boundary is defined by port declarations
  - “Upper boundary” only

**UPF 2.0**
- Boundary is defined by domain crossings
  - “Upper boundary”
    - Ports connected to nets in a ‘higher’ domain (lowconn of those ports)
  - “Lower boundary”
    - Nets connected to ports in a ‘lower’ domain (highconn of those ports)
Power Domain Lower Boundary

- **Domain Boundary**
  - Isolation/Level shifting are only inserted at power domain boundaries
  - Upper and lower boundaries are well defined for RTL
  - Lower boundary is more complex with hard macro instances
  - Hard macro instances may have multiple supplies
  - Each macro instance port may have a different supply

- **Lower Boundary**
  - Now includes *macro instance ports with different supplies*
Hard Macro Supplies

**Driver supply**
- Supply set powering outputs

**Receiver supply**
- Supply set powering inputs

**Behavioral models**
- Supplies may not be evident
- set_port_attributes -driver_supply ...
- set_port_attributes -receiver_supply ...

**Hard macros (e.g., Memory)**
- Attributes of cell pins:
  - PG type attributes
  - Related supply attributes
- Imply anonymous supply sets
- Can be defined in UPF or Liberty

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

- **UPF 1.0**
  - Upper boundary only
    - Defined by port declarations

- **UPF 2.0**
  - Upper and lower boundaries
    - Defined by domain crossings

- **UPF 2.1**
  - Lower boundary extended
    - Now includes ports with driving/receiving supplies different from the primary supply of the containing power domain
Supply Set Definition

**Functions**
- Restricted to (power, ground, nwell, pwell, deepnwell, deeppwell)
- User-defined functions removed

**Construction**
- Implicit driver/receiver supply set built from
  - Liberty attributes
    - related_power_pin, related_ground_pin, related_bias_pins
  - UPF attributes
    - UPF_related_power_port, UPF_related_ground_port, UPF_related_bias_ports

**Association**
- Connects corresponding functions
- Propagates required function list upwards
- Does not propagate power states in either direction
Related Supply Attributes

- **Liberty Attributes**
  - related_power_pin
  - related_ground_pin
  - related_bias_pin

- **UPF/HDL Attributes (supply net)**
  - UPF_related_power_port
  - UPF_related_ground_port
  - UPF_related_bias_ports

- **UPF Attributes (supply set)**
  - UPF_driver_supply
  - UPF_receiver_supply

- **Implicit Supply Set**
  - power = related power pin/port
  - ground = related ground pin/port
  - nwell = related bias pin/port P1
    where pg_type(P1) = nwell
  - pwell = related bias pin/port P2
    where pg_type(P2) = pwell
  - deepnwell = related bias pin/port P3
    where pg_type(P3) = deepnwell
  - deeppwell = related bias pin/port P4
    where pg_type(P4) = deeppwell
Power States

Syntax
- Added `-supply` and `-domain` to clarify usage

Expressions
- For supplies, `-supply_expr` can refer to functions directly
- For domains, `-logic_expr` can refer to supply sets directly

Semantics
- Supply expr (if any) must be True when logic expr is True
  - Indicates that supply required for state is being provided
    - Must turn on before, and turn off after, Logic expr is True
  - Does not represent a switch control
- Inverse is no longer required
Power States of Supply Sets

```plaintext
add_power_state <supply set>

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

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

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

<table>
<thead>
<tr>
<th>PD_Proc</th>
<th>PD_Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary</td>
<td>memory</td>
</tr>
<tr>
<td>ON_10</td>
<td>ON_08</td>
</tr>
<tr>
<td>or OFF</td>
<td>or OFF</td>
</tr>
</tbody>
</table>

© 2013 Mentor Graphics
Power States of Supply Sets

- **add_power_state <supply set>**

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

```
add_power_state PD_Mem.primary -supply -update\  
  -state INACTIVE { \  
    -logic_expr {sleep == 1'b1} } \  
  -state ACTIVE { \  
    -logic_expr {sleep == 1'b0 } }
```

```
add_power_state PD_Mem.primary -supply -update\  
  -state IN_ACTIVE { \  
    -logic_expr {sleep == 1'b1 || \  
      PD_Mem.primary == OFF } \  
  -state ACTIVE { \  
    -logic_expr {sleep == 1'b0 && \  
      PD_Mem.primary == ON_08 }
```

**UPF 2.1**

- A -supply_expr for a supply can refer to
  - supply ports/nets at or below the current scope
  - functions of the supply set to which the add_power_state command applies

- But cannot refer to
  - functions of another supply set

- A -logic_expr for a supply can refer to
  - logic ports/nets at or below the current scope
  - interval functions
  - power states of the supply set to which the add_power_state command applies

- But cannot refer to
  - functions of a supply set
  - power states of another supply set
  - power states of a domain
Power States of Power Domains

- **add_power_state <domain>**

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

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

- **Table:**

<table>
<thead>
<tr>
<th>PD_PROC</th>
<th>primary</th>
<th>memory</th>
<th>PD_MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>ON_10</td>
<td>ON_08</td>
<td>UP</td>
</tr>
<tr>
<td>Sleep</td>
<td>OFF</td>
<td>ON_08</td>
<td>UP</td>
</tr>
<tr>
<td>Hibernate</td>
<td>OFF</td>
<td>OFF</td>
<td>DOWN</td>
</tr>
</tbody>
</table>
Power States of Power Domains

- **add_power_state <domain>**

  add_power_state PD_Mem -domain \ 
  -state UP { \ 
    -logic_expr {primary == ACTIVE} } \ 
  -state DOWN { \ 
    -logic_expr {primary == INACTIVE} } 

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

- **UPF 2.1**

  - A `-supply_expr` cannot be used to define power states of a domain
  - A `-supply_expr` for a domain can refer to
    - logic ports/nets at or below the current scope
    - interval functions
    - power states of supply sets
    - power states of other power domains
  - But cannot refer to
    - supply ports or nets
    - supply set functions
Hierarchical Power State Definition

- Restrictions are intended to
  - Avoid circular definitions
  - Encourage methodical usage
Simstates

- **CORRUPT**
  - Combinational outputs corrupted
  - Sequential state/outputs corrupted

- **CORRUPT_ON_ACTIVITY**
  - Combinational outputs maintained as long as inputs are stable
  - Sequential state/outputs corrupted

- **CORRUPT_ON_CHANGE**
  - Combinational outputs maintained as long as outputs are stable
  - Sequential state/outputs corrupted

- **NORMAL**
  - Combinational logic functions normally
  - Sequential logic functions normally
  - Both operate with characterized timing

- **CORRUPT_STATE_ON_ACTIVITY**
  - Combinational logic functions normally
  - Sequential state/outputs maintained as long as inputs are stable

- **CORRUPT_STATE_ON_CHANGE**
  - Combinational logic functions normally
  - Sequential state/outputs maintained as long as outputs are stable
Precedence for Simstate Application

- **Principle**
  - For simstates that apply to a given object at any given time, a more conservative (i.e., more corrupting) simstate takes precedence over a less conservative (less corrupting) simstate.

- **Precedence (lowest to highest)**
  - NORMAL
  - CORRUPT_STATE_ON_CHANGE
  - CORRUPT_ON_CHANGE
  - CORRUPT_STATE_ON_ACTIVITY
  - CORRUPT_ON_ACTIVITY
  - CORRUPT
Power Management Strategies

**Strategies**
- set_retention
- set_repeater
- set_isolation
- set_level_shifter

**Refinements**
- \(-\text{source/}\)sink filters generalized
  - Can take either supply set or domain name (=> domain.primary)
- Different strategies allowed for different sinks of the same port
  - Sink filter makes each strategy “more specific”
- Locations \text{fanin, faninout, sibling} deprecated
  - Led to unpredictable iso/ls ordering
- Old power/ground net options deprecated
  - Use supply set options instead
Retention Strategies

- Much more detailed explanation of retention states
- New support for live-slave style retention latch
  - Single control pin or no control pin

![Retention state transition diagram for balloon-style retention](image1)

![Retention state transition diagram for master/slave-alive style retention](image2)
Repeater (Buffer) Strategies

- **New command:** `set_repeater`
  - Replaces `set_port_attributes ... -repeater_supply`
  - Can apply to either inputs or outputs

- **Similar to isolation/level shifter strategies:**
  - `-domain -elements`
  - `-source -sink -applies_to`
  - `-use_equivalence`
  - `-repeater_supply_set`
  - `-instance -update`
Repeater Insertion

- **set_repeater**

  set_repeater BUF1 \ 
  -domain PD_Proc \ 
  -elements {.} \ 
  -repeater_supply_set SS_AON
Repeater Insertion

- **set_repeater**
  
  set_repeater BUF1 \\
  -domain PD_Proc \\
  -elements {.} \\
  -repeater_supply_set SS_AON

set_repeater BUF2 \\
-domain PD_Proc \\
-applies_to_inputs \\
-sink PD_Mem \\
-repeater_supply_set PD1.primary
Repeater Insertion

- **set_repeater**

```python
set_repeater BUF1 \
- domain PD_Proc \
- elements {.} \
- repeater_supply_set SS_AON

set_repeater BUF2 \
- domain PD_Proc \
- applies_to inputs \
- sink PD_Mem \
- repeater_supply_set PD1.primary

set_repeater BUF3 \
- domain PD_Proc \
- applies_to outputs \
- source PD_Proc \
- repeater_supply_set PD2.backup
```
Isolation and Level Shifting Strategies

- **Isolation**
  - Clarified `-force_isolation` and `-no_isolation` options
    - If neither specified, tools may optimize
  - Deprecated `-source_off_clamp/-sink_off_clamp`
    - These are constraints - specify with `set_port_attributes`

- **Level Shifting**
  - Clarified `-force_shift` and `-no_shift` options
    - If neither specified, tools may optimize
  - Deprecated `-threshold list` form
    - Too complex and not used at all

- **Both**
  - Deprecated `-transitive` option
    - Strategies apply at domain boundaries, not at each hierarchy level
Isolation Locations

- location **self** is a generalization of **parent** (refers to child instances also)
- location **fanout** is also supported; essentially a possibly remote **other**
Isolation Defaults

- **Isolation Strategy**
  - No default strategy

- **Isolation Supply**
  - Uses `default_isolation` supply of the domain it is inserted into

Not the default
Requires explicit setting

Not the default
Requires explicit setting
Level Shifter Locations

Level shifters can be inserted before, after, or between isolation cells.
Each domain can contribute a level shifter.
Level Shifter Defaults

Level Shifter Strategy
- Default strategy defined for LowConn side of a port:
  - `set_level_shifter -domain <domain name> \ -elements <port name> \ -rule both -threshold 0`

Level Shifter Supplies
- Input supply = driving supply at insertion point
- Output supply = receiving supply at insertion point
- But only if
  - Exactly one level-shifter strategy applies to the port and
  - All driving supplies are equivalent, all receiving supplies are equivalent.
Strategy Execution Order

Retention, then Repeater, then Isolation, then Level Shifter
<table>
<thead>
<tr>
<th></th>
<th>Retention</th>
<th>Repeater</th>
<th>Isolation</th>
<th>Level Shifter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retention</td>
<td>--</td>
<td>affects</td>
<td>affects</td>
<td>affects</td>
</tr>
<tr>
<td>Repeater</td>
<td>affected</td>
<td>--</td>
<td>affects</td>
<td>affects</td>
</tr>
<tr>
<td></td>
<td>by</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Isolation</td>
<td>affected</td>
<td>affected</td>
<td>--</td>
<td>affects</td>
</tr>
<tr>
<td></td>
<td>by</td>
<td>by</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level Shifter</td>
<td>affected</td>
<td>affected</td>
<td>affected</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>by</td>
<td>by</td>
<td>by</td>
<td></td>
</tr>
</tbody>
</table>

Strategies may change driver and/or receiver supplies of a port
This may affect *source/-sink* filters of subsequently executed strategies
Precedence for Strategies

From lowest to highest:

- **Strategy for all ports of a specified power domain**

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

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

- **Strategy for a whole port specified explicitly by name**

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

- **Syntax**

  ```
  set_equivalent
  [-function_only]
  [-nets supply_net_name_list]
  [-sets supply_set_name_list]
  ```

  - Default is electrical and functional equivalence
  - Use for nets or sets, but not both

- **Examples**

  ```
  set_equivalent \
  -nets {vddaon backup}
  ```

  ```
  set_equivalent \
  -sets { \
    PD1.primary \
    PD2.retention }
  ```

  ```
  set_equivalent \
  -function_only \
  -nets {vss1 vss2 gnd}
  ```
Using Equivalence

- **New Option**
  - `use_equivalence [TRUE | FALSE]`
  -Defaults to TRUE; can be turned off (FALSE)

- **For Strategies**
  - `set_isolation -source -sink -diff_supply_only -use_equivalence`
  - `set_level_shifter -source -sink -use_equivalence`
  - `set_repeater -source -sink -use_equivalence`

- **Used Implicitly Elsewhere**
  - Implicitly (TRUE) in
  - `create_composite_domain`
IEEE 1801-2013 UPF - Summary

- Many small changes
  - Focused on clarity and precision
  - Improving modeling fidelity
  - Reflecting flow requirements

- Some deprecations/restrictions
  - Supply set functions
  - Supply nets for strategies

- Some additions/extensions
  - Atomic power domains
  - Supply equivalence
  - Precedence rules

- A more powerful and finely tuned standard
Agenda

- Introduction
- Modeling Power Intent with IEEE 1801
- New Features in IEEE 1801-2013

~ Break ~

- Modeling Power Management cells and Hard Macros
- Low Power Design Methodology for IP Providers
- Low Power Design Methodology for SoC Designers
Modeling Power Management Cells and Hard Macros

Dr. Qi Wang
Solutions Group Director
Cadence
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

- Additional documentation on how to use these commands
  - Appendix I – Power management cell modeling examples
Use Model

- Annotation attributes on instances of specified cells under the context of UPF
- An alternative to modeling low power management cells using Liberty

Usage scenarios
- Internal library development
- Single representation (using UPF) of all power intents
- Incomplete libraries
- Legacy libraries

Additional information
- Appendix H – Power management commands semantics and Liberty mappings
State Retention Cells

Retention flops with both save and restore

```bash
define_retention_cell -cells SR1 -clock_pin Clk \   
   -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

```bash
define_retention_cell -cells SR1 -clock_pin Clk \   
   -restore_check !Clk -save_check !Clk \   
   -power_switchable VDD_SW -power VDD \   
   -ground VSS
```
Always-on Cells

Power switched always-on buffer

```c
define_always_on_cell -cells LP_buf_pow \
  -power VDD -power_switchable VSW -ground VSS
```

Power and ground switched always-on buffer

```c
define_always_on_cell -cells LP_buf \
  -power VDD -power_switchable VSW \ 
  -ground VSS -ground_switchable GSW
```

Always-on flop with isolated input pins

```c
define_always_on_cell -cells LP_ff\ 
  -power VDD -power_switchable VSW -ground VSS \ 
  -isolated_pins {{SE SI}} -enable {!ISO}
```
Isolation Cells

Isolation cell can be put in switched-off domain

```
define_isolation_cell -cells IsoLL \ 
  -power_switchable VSW -power VDD -ground VSS \ 
  -enable E -valid_location source
```

Isolation cell with two control pins

```
define_isolation_cell -cells {myiso} \ 
  -power_switchable VDD -power VSW -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}}
```
Level Shifter Cells

Simple power level shifter

```bash
define_level_shifter_cell -cells low_to_high_power \ 
  -input_voltage_range {{0.8 1.0}} \ 
  -output_voltage_range {{1.0 1.2}} \ 
  -input_power_pin VDD_IN -output_power_pin VDD_OUT \ 
  -ground VSS_IN \ 
  -direction low_to_high -valid_location source
```

Enabled power level shifter

```bash
define_level_shifter_cell -cells low_to_high_power_enable \ 
  -input_voltage_range {{0.8 1.0}} \ 
  -output_voltage_range {{1.0 1.2}} \ 
  -input_power_pin VDD_IN -output_power_pin VDD_OUT \ 
  -ground VSS_IN \ 
  -direction low_to_high -valid_location source \ 
  -enable En
```
Power Switch Cells

Single stage power switch

```plaintext
define_power_switch_cell -cells sw1 \ 
    -stage_1_enable Ei -stage_1_output Eo \ 
    -type header -power_switchable VSW -power VIN -ground VSS
```

Single stage ground switch

```plaintext
define_power_switch_cell -cells sw1 \ 
    -stage_1_enable Ei -stage_1_output Eo \ 
    -type footer -ground_switchable GSW -power VDD -ground VSS
```

Single stage ground switch

```plaintext
define_power_switch_cell -cells sw1 \ 
    -stage_1_enable Ri -stage_1_output Ro \ 
    -stage_2_enable Ei -stage_2_output Eo \ 
    -type header -power_switchable VSW -power VIN -ground VSS
```
Work with Strategies

Cell definition

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

Strategy specification

```
set_isolation isol1 -domain PD1 -source PD1 -location self \  
  -isolation_signal { iso_drvr en_drvr} \  
  -isolation_sense { high low } -clamp_value 0
```

Retention flops with live slave latch

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

Strategy specification

```
set_retention srl -domain PD1 \  
  -retention_condition {!clock && nreset} \  
  -use_retention_as_primary \  
  ...
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
Hard Macros Modeling Requirements

- Power models for hard macros with power management features
  - Multiple supply sets (VDD1, VDD2, VSS)
  - Power states
  - Domain with retention
  - Domain without retention
  - Inputs isolated (D3)
  - Inputs not isolated (D2)
  - Outputs isolated (D4)
  - Outputs not isolated (D5)
  - Feed-through (F1, F2)
  - Floating port (NC)

- Consistent power spec used for both implementation and verification
Use Model

- Modeling hard macros with complex power management features

- Liberty cell modeling provides basic modeling requirements

- New 1801-2013 power model provides additional modeling capabilities:
  - Power states
  - Feedthrough and floating ports
  - Internal isolations
New Commands for Power Model

**Commands to define a power model containing other UPF commands**

```plaintext
begin_power_model  power_model_name  [-for model_list]
end_power_model
apply_power_model  power_model_name  [-elements instance_list]
[-supply_map  {{lower_scope_handle upper_scope_supply_set}*}]
```

- 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
Additional Modeling for Power Model
Attributing feedthrough ports

- **Option** `-feedthrough` in `set_port_attributes`
  - Specify the ports of a model that are all connected by the same metal wire electrically.

- **For the example**
  ```
  set_port_attributes -ports {I2 O1 O2} \\
  -model cellX -feedthrough
  set_port_attributes -ports {I3 O3} \\
  -model cellX -feedthrough
  ```

- **Transitive effect**
  ```
  set_port_attributes -ports {I2 O1} \\
  -model cellX -feedthrough
  set_port_attributes -ports {I2 O2} \\
  -model cellX -feedthrough
  ```

  is equivalent to
  ```
  set_port_attributes -ports {I2 O1 O2} \\
  -model cellX -feedthrough
  ```
Additional Modeling for Power Model
Attributing unconnected ports

- Option **-unconnected** in set_port_attributes
  - If such a port is an input port, it means there is no logic within the model driven by the port
  - If such a port is an output port, it means there is no logic within the model driving the port
  - These ports shall not be associated with any other port attributes
  - This attribute also overwrites any default supply net or supply set association with respect to the specified ports, i.e., the specified ports are not associated with any supply net or supply set in UPF.

- For the example
  ```
  set_port_attributes -ports {I4} -unconnected
  ```
Example of Hard Macro Modeling using Power Model

```
begin_power_model pml -for myCell
#define power domains and interface supply sets
create_power_domain PD -elements {.}:
  -supply { SSAH } -supply { SSBH }
...
#associate the interface supplies to boundary supply #ports or internally generated supplies
create_supply_set PD.SHAH :
  -function { power vddA } :
  -function { ground vssA } -update
...
#define data port and interface supply set handle #associations
set_port_attributes -ports {W X} :
  -receiver_supply PD.SSAH
set_port_attributes -ports {Z2} :
  -driver_supply PD.SSBH

#define power states for supply sets and system
add_power_state PD.SSAH ...
add_power_state PD ...
end_power_model
```
Integration of Power Model

- Use "load_upf <file> -scope <inst_name>" to load in macro UPF
- Use apply_power_model to connect the SoC level supply sets with the interface supply sets of the soft IP

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

define the connection for IP level
#interface supply
apply_power_model pml -elements {I2} \ 
-supply_map {{PD.SSAH PDTop.SSH1} \ 
{PD.SSBH PDTop.SSH2}}
```
Agenda

- Introduction
- Modeling Power Intent with IEEE 1801
- New Features in IEEE 1801-2013
  ~ Break ~
- Modeling Power Management cells and Hard Macros
- Low Power Design Methodology for IP Providers
- Low Power Design Methodology for SoC Designers
Low Power Design Methodology for IP Providers

John Biggs
Senior Principal Engineer
ARM

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
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 }
  ```
Cortex-A MPCore Power States

<table>
<thead>
<tr>
<th>Power State</th>
<th>PD_CPU</th>
<th>PD_FPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>RUN</td>
<td>*</td>
</tr>
<tr>
<td>RETENTION</td>
<td>RET</td>
<td>!RUN</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

!RUN = RET or OFF

<table>
<thead>
<tr>
<th>Power State</th>
<th>PD_CPU0</th>
<th>PD_CPU1</th>
<th>PD_CLSTR</th>
<th>PD_L2RAM</th>
<th>PD_DBG</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>RUN</td>
<td>RUN</td>
<td>RUN</td>
<td>RUN</td>
<td>*</td>
</tr>
<tr>
<td>RETENTION</td>
<td>!RUN</td>
<td>!RUN</td>
<td>!RUN</td>
<td>RET</td>
<td>*</td>
</tr>
<tr>
<td>DORMANT</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>*</td>
</tr>
<tr>
<td>DEBUG</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>RUN</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

© 2013 ARM Ltd
2 June 2013

Low-power Design with the New IEEE 1801-2013 Standard
Successive Refinement of Power Intent

1. **IP Creation**
   - RTL
   - +
   - Soft IP
   - Constraint UPF

2. **IP Configuration**
   - RTL
   - Constraint UPF
   - +
   - Configuration UPF
   - Golden Source

3. **IP Implementation**
   - RTL
   - Constraint
   - Config’n UPF
   - +
   - Imp’tion UPF
   - Impl’tion UPF
   - +
   - Synthesis
   - Impl’tion UPF
   - Netlist
   - P&R
   - Impl’tion UPF
   - Netlist

**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

1. Atomic power domains
   create_power_domain PD_CPU -elements {.
   -exclude_elements "$FPU" -atomic
   create_power_domain PD_FPU -elements "$FPU" -atomic

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

Put everything in PD_CPU
Then call out PD_FPU

Retain “all or nothing”

Clamp everything low by default
Then call out the exceptions
CPU Constraints (cont...)  

4. Power State

```plaintext
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 }}

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}}
```

<table>
<thead>
<tr>
<th>PD_FPU</th>
<th>primary</th>
<th>retention</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>RET</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PD_CPU</th>
<th>primary</th>
<th>retention</th>
<th>PD_FPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>ON</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>OFF</td>
<td>ON</td>
<td>!RUN</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Define **PD_FPU** in terms of its supply sets

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
   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 \\ 

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

Put everything in PD_CLSTR
Then call out PD_L2RAM and PD_DEBUG
Retain "all or nothing"
Clamp everything low by default
Then call out the exceptions
Cluster Constraints (cont...)

4. Power Domain State

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}}

Define PD_L2RAM in terms of its supply sets

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

Define PD_DEBUG in terms of its supply sets

<table>
<thead>
<tr>
<th></th>
<th>primary</th>
<th>retention</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD_L2RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>RET</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD_DEBUG</td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td>ON</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>
Cluster Constraints (cont...) 

4. Power Domain State (Cont.)

```c
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_CPU == OFF}}
```

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

<table>
<thead>
<tr>
<th>PD_CLSTR</th>
<th>primary</th>
<th>PD_L2RAM</th>
<th>PD_CPU0</th>
<th>PD_CPU1</th>
<th>PD_DEBUG</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>ON</td>
<td>RUN</td>
<td>RUN</td>
<td>RUN</td>
<td>*</td>
</tr>
<tr>
<td>RET</td>
<td>OFF</td>
<td>RET</td>
<td>!RUN</td>
<td>!RUN</td>
<td>*</td>
</tr>
<tr>
<td>DMT</td>
<td>OFF</td>
<td>RUN</td>
<td>OFF</td>
<td>OFF</td>
<td>*</td>
</tr>
<tr>
<td>DBG</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>RUN</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>
Successive Refinement of Power Intent

1. IP Creation
   - RTL
   - +
   - Constraint UPF
   - Soft IP

2. IP Configuration

We now have UPF constraints to go along with the unconfigured RTL

This is what an IP provider would deliver

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}
  ```

- **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
  ```

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

  PD_FPU not required

Clamp all outputs low by default

Clamp the exceptions high (more specific overrides more generic)
CPU Configuration

- Update power supply state with supply expressions

```bash
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

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

CPU state retention not required

PD_FPU is in run when nPWRUP_CPU is low

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} \
  ```

- 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
  ```
Cluster Configuration

- Update power supply state with supply expressions

```plaintext
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

```plaintext
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

1. **IP Creation**
   - **RTL**
   - +
   - **Constraint UPF**

2. **IP Configuration**
   - **Soft IP**
   - +
   - **Constraint UPF**
   - **Configuration UPF**

3. **IP Implementation**
   - **Golden Source**

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

**IP LicenseeUser:**
- Configures IP for context
- Validates configuration
- Freezes “Golden Source”

We now have a fully configured technology independent “Golden Reference” ready for implementation.
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}
  ```

- 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 {!nPWRUP_CPU}}
  -off_state {off_state nPWRUP_CPU}
  ```

Use **VDD_CPU** for primary power

Use **VDD** for isolation power

Switch drives **VDD_CPU** with **VDD** when **nPWRUP_CPU** is low
Cluster Implementation

- Create supply nets and update supply set functions

```plaintext
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

```c
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

```plaintext
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 !nPWRUP_CLSTR}
  -off_state {off_state nPWRUP_CLSTR}

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 {!nPWRUP_L2RAM}
  -off_state {off_state nPWRUP_L2RAM}

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 {!nPWRUP_DEBUG}
  -off_state {off_state nPWRUP_DEBUG}
```

Switch drives VDD_CLSTR with VDD when nPWRUP_CLSTR is low
Successive Refinement of Power Intent

1. **IP Creation**
   - RTL
   - Soft IP
   - +
   - Constraint UPF

2. **IP Configuration**
   - RTL
   - Constraint
   - UPF
   - +
   - Configuration
   - UPF

3. **IP Implementation**
   - RTL
   - Constraint
   - Config'n
   - UPF
   - +
   - Impl'tion
   - UPF
   - Synthesis
   - Netlist
   - P&R
   - Netlist

**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 of macro
  - Just need to verify its low power integration into 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

```plaintext
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
Agenda

- Introduction
- Modeling Power Intent with IEEE 1801
- New Features in IEEE 1801-2013
  ~ Break ~
- Modeling Power Management cells and Hard Macros
- Low Power Design Methodology for IP Providers
- Low Power Design Methodology for SoC Designers
Low Power Design Methodology for SoC Designers

Sushma Honnavara-Prasad
Principal Engineer
Broadcom
Section Agenda

- Introduction
- Soc Low Power Integration Tips
- Hard IP Modeling
- SoC Integration
- SoC Verification
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 Power Intent - prerequisites

- Power management architecture
  - Power domains, retention etc
- List of top level supplies
- List of multi-rail macros and their power connectivity
- List of IO pads and their power connectivity
- Power states
- Simulation behavior
SoC Low Power Integration Tips
Design/UPF Partitioning

- **Partition design UPF into sub-module UPF**
  - Place and route block boundary
  - Power domain boundary
    ```
    load_upf $env(UPF_PATH)/module1/upf/module1.upf \
    -scope core_inst/module1_inst
    load_upf $env(UPF_PATH)/module2/upf/module2.upf \
    -scope core_inst/module2_inst
    load_upf $env(UPF_PATH)/module3/upf/module3.upf \
    -scope core_inst/module3_inst
    ```

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

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

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

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

```plaintext
set pad_inst_list [find_objects -pattern *PAD_SEG2_inst* \n                     -object_type inst -leaf_only -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

```plaintext
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

```plaintext
set pll_inst_list [find_objects . -pattern *u_pll* -object_type inst \ 
    -leaf_only -transitive]
foreach inst $pll_inst_list {
    connect_supply_net lp8ss.power -ports "$inst/AVDD1P8"
    connect_supply_net lp8ss.ground -ports "$inst/AVSS"
}
```
Hard IP Modeling
Domain Definitions and supply ports

create_power_domain PD_SW -elements {}
create_power_domain PD_AON -elements \u_control_inst/u_aon_control_inst\}

create_supply_port VDD_1P8
create_supply_port VDD_AON
create_supply_port VDD_VAR
create_supply_port VSS_1P8
create_supply_port VSS
create_supply_net VDD_SW \ 
- domain PD_SW

create_supply_net VDD_AON \ 
- domain PD_AON \ 
- reuse

create_supply_net VSS \ 
- domain PD_AON \ 
- reuse

create_supply_set aonss -function {power VDD_AON} -function {ground VSS} -function {nwell VDD_AON}
create_supply_set varss -function {power VDD_VAR} -function {ground VSS} -function {nwell VDD_VAR}
create_supply_set swss -function {power VDD_SW} -function {ground VSS} -function {nwell VDD_SW}
create_supply_set 1p8ss -function {power VDD_1P8} -function {ground VSS_1P8 }
Set Port Attributes – internal constraints

```
set aon_ports  [find_objects . -pattern *_aon* -object_type port]
set var_in_port [find_objects . -pattern *pwr* -object_type port]

set_port_attributes -elements {.} -applies_to outputs -driver_supply varss
set_port_attributes -elements {.} -ports "$var_in_port" -receiver_supply varss
set_port_attributes -elements {.} -ports "$aon_ports" -applies_to inputs \ -receiver_supply aonss
set_port_attributes -elements {.} -ports "$aon_ports" -applies_to outputs \ -driver_supply aonss
```

IP Centric view
Does not specify external integration requirement
Set Port Attributes – external constraints

```
set aon_ports [find_objects -pattern *_aon* -object_type port]

set_port_attributes -elements {.} -applies_to inputs -driver_supply varss
set_port_attributes -elements {.} -applies_to outputs -receiver_supply varss
set_port_attributes -elements {.} -ports "$aon_ports" -applies_to inputs \  
  -receiver_supply aonss
set_port_attributes -elements {.} -ports "$aon_ports" -applies_to outputs \  
  -driver_supply aonss
```
Power Switch Creation

create_power_switch PSW_SW -domain PD_SW \\  -output_supply_port { VDD swss.power } \\
-input_supply_port { VDDB VDD_VAR } \\
-control_port { pwronin u_control_inst/u_aon_control_inst/pwronin } \\
-control_port { pwrokin u_control_inst/u_aon_control_inst/pwrokin } \\
-ack_port { pwronout u_control_inst/u_aon_control_inst/pwronout {pwronin}} \\
-ack_port { pwrokout u_control_inst/u_aon_control_inst/pwrokout {pwrokin}} \\
-on_state { full_on VDDB {pwronin & pwrokin}} \\
-off_state { full_off {!pwronin & !pwrokin}}
Supply net connections

```plaintext
associate_supply_set swss \ -handle PD_SW.primary
associate_supply_set aonss \ -handle PD_AON.primary

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"
}
```
Isolation strategy

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

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

use_interface_cell source_isolow -strategy sw_iso_c0 -domain PD_SW -lib_cells {CELL_ISOLOWX4}
```
Isolation/Level shifter strategies

set_level_shifter no_ls_sw_to_aon
-domin PD_SW -sink aonss \\ -threshold 0.2

set_level_shifter no_ls_aon_to_sw \\
-domin PD_AON -sink swss \\
-threshold 0.2

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

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

use_interface_cell source_isolo1 -strategy sw_to_aon_0 -domain PD_AON -lib_cells {CELL_ISOLOWSX4}
Supply states

<table>
<thead>
<tr>
<th>Supply</th>
<th>nom</th>
<th>turbo</th>
<th>offmode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VDD</td>
<td>VSS</td>
<td>Nwell</td>
</tr>
<tr>
<td>aonss</td>
<td>0.8</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>varss</td>
<td>0.8</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>swss</td>
<td>0.8</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>1p8ss</td>
<td>1.8</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th>State</th>
<th>aonss</th>
<th>varss</th>
<th>swss</th>
<th>1p8ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
</tr>
<tr>
<td>turbo</td>
<td>nom</td>
<td>turbo</td>
<td>turbo</td>
<td>nom</td>
</tr>
<tr>
<td>sw1off</td>
<td>nom</td>
<td>nom</td>
<td>offmode</td>
<td>nom</td>
</tr>
<tr>
<td>sw2off</td>
<td>nom</td>
<td>turbo</td>
<td>offmode</td>
<td>nom</td>
</tr>
<tr>
<td>sw3off</td>
<td>nom</td>
<td>offmode</td>
<td>offmode</td>
<td>nom</td>
</tr>
</tbody>
</table>

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

Power model for the Hard IP

begin_power_model upf_block
  -for {block}
  create_supply_port VDD_1P8
  create_supply_port VDD_AON
  create_supply_port VDD_VAR
  create_supply_port VSS_1P8
  create_supply_port VSS
  create_supply_net VDD_SW
  create_supply_set swss
    -function {power VDD_SW} \ 
    -function {ground VSS}
  create_supply_set varss
    -function {power VDD_VAR} \ 
    -function {ground VSS}
  create_supply_set aonss
    -function {power VDD_AON} \ 
    -function {ground VSS}
  create_supply_set lp8ss
    -function {power VDD_1P8} \ 
    -function {ground VSS_1P8}
  create_power_domain PD_SW
    -supply { primary swss } \ 
    -supply { backup varss } \ 
    -supply { top aonss } \ 
    -supply { analog lp8ss }
Power model for the Hard IP

```bash
set_port_attributes \
-elements 
-apply_to outputs 
-receiver_supply vars

set_port_attributes \
-elements 
-apply_to inputs 
-driver_supply vars

set_port_attributes \
-driver_supply aonss \
-ports { iso \
    [find_objects . \
        -pattern *_aon* \
        -object_type port] }

set_port_attributes \
-elements 
-apply_to outputs 
-clamp_value 0

set_port_attributes \
-ports $clamp1_ports \
-clamp_value 1
```
Power model for the Hard IP

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

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

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

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

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

end_power_model

Low-power Design with the New IEEE 1801-2013 Standard © 2013 Broadcom 2 June 2013 138
AON Buffering

- Aka supply aware buffering
- Used to specify repeater insertion on nets using a supply other than the domain primary supply.
- Implemented based on driver supply/receiver supply and domain primary supply sets.
- For primary input or output ports of a domain, the driver or receiver supply sets can be specified using `set_port_attributes`.
- An explicit repeater strategy can also be specified for ports of a domain using the `set_repeater` command.

<table>
<thead>
<tr>
<th>set_port_attributes</th>
<th>Set_repeater</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not imply a repeater</td>
<td>Implies a repeater</td>
</tr>
<tr>
<td>Source/sink is specified in the command</td>
<td>Source/sink of the port is altered by <code>repeater_supply_set</code></td>
</tr>
</tbody>
</table>
AON Buffering

- set_repeater aonbuf1 -domain PD_SW -elements {inp1} \
  -repeater_supply_set vdd1ss

- set_repeater aonbuf2 -domain PD_SW -elements {inp2} \
  -repeater_supply_set vdd2ss

- set_repeater aonbuf3 -domain PD_SW -elements {out2} \
  -repeater_supply_set vdd3ss
IO Pads
Supply Ports

1. `create_supply_port VDD_1P8`
2. `create_supply_port VSS_1P8`
3. `create_supply_port VDD_VAR`
4. `create_supply_port VSS`

5. `create_supply_net VDD_1P8` 
   - domain PD_TOP
6. `create_supply_net VSS_1P8` 
   - domain PD_TOP
7. `create_supply_net VDD_VAR` 
   - domain PD_TOP
8. `create_supply_net VSS` 
   - domain PD_TOP

9. `connect_supply_net VDD_1P8` 
   - ports u_pads/VDDP_pad_inst/pad
10. `connect_supply_net VSS_1P8` 
    - ports u_pads/VSSP_pad_inst/pad
11. `connect_supply_net VDD_VAR` 
    - ports u_pads/VDDC_pad_inst/pad
12. `connect_supply_net VSS` 
    - ports u_pads/VSSC_pad_inst/pad
Supply Connections

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

connect_supply_net VDDC_1P8 \  
- ports \  
u_pads/VDDP_pad_inst/VDDPOUT  
connect_supply_net VSSC_1P8 \  
- ports \  
connect_supply_net VDDC_1P8 \  
- ports u_pads/SIG_PAD1/VDDP  
connect_supply_net VSSC_1P8 \  
- ports u_pads/SIG_PAD1/VSSP  
connect_supply_net VDDC_VAR \  
- ports u_pads/SIG_PAD1/VDDC  
connect_supply_net VSSC \  
- ports u_pads/SIG_PAD1/VSSC
Supply States

<table>
<thead>
<tr>
<th>Supply</th>
<th>nom</th>
<th>turbo</th>
<th>offmode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VDD</td>
<td>VDD</td>
<td>VDD</td>
</tr>
<tr>
<td></td>
<td>VSS</td>
<td>VSS</td>
<td>VSS</td>
</tr>
<tr>
<td></td>
<td>Nwell</td>
<td>Nwell</td>
<td>Nwell</td>
</tr>
<tr>
<td>varss</td>
<td>0.8</td>
<td>0.9</td>
<td>off</td>
</tr>
<tr>
<td>1p8ss</td>
<td>1.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>State</th>
<th>varss</th>
<th>1p8ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>nom</td>
<td>nom</td>
<td>nom</td>
</tr>
<tr>
<td>turbo</td>
<td>turbo</td>
<td>nom</td>
</tr>
<tr>
<td>offmode</td>
<td>offmode</td>
<td>nom</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Supplies</th>
<th>Type</th>
<th>nom</th>
<th>turbo</th>
<th>offmode</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO supplies</td>
<td>Constant</td>
<td>1.8V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AON supply</td>
<td>Constant</td>
<td>0.8V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR1 supply</td>
<td>Variable/switchable</td>
<td>0.8V</td>
<td>0.9V</td>
<td>Off</td>
</tr>
<tr>
<td>VAR2 supply</td>
<td>Variable/switchable</td>
<td>0.8V</td>
<td>0.9V</td>
<td>Off</td>
</tr>
<tr>
<td>VAR3 supply</td>
<td>Variable</td>
<td>0.8V</td>
<td>0.9V</td>
<td></td>
</tr>
<tr>
<td>VAR4 supply</td>
<td>Switchable</td>
<td>0.9V</td>
<td></td>
<td>Off</td>
</tr>
</tbody>
</table>

Note: Some states are left out for brevity.

```c
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}
```

```c
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

<table>
<thead>
<tr>
<th>State</th>
<th>ioss</th>
<th>aonss</th>
<th>var1ss</th>
<th>var2ss</th>
<th>var3ss</th>
<th>var4ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
</tr>
<tr>
<td>state1</td>
<td>nom</td>
<td>nom</td>
<td>turbo</td>
<td>turbo</td>
<td>turbo</td>
<td>nom</td>
</tr>
<tr>
<td>state2</td>
<td>nom</td>
<td>nom</td>
<td>turbo</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
</tr>
<tr>
<td>state3</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>turbo</td>
<td>nom</td>
<td>nom</td>
</tr>
<tr>
<td>state4</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
<td>turbo</td>
<td>nom</td>
</tr>
<tr>
<td>state5</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>nom</td>
<td>nom</td>
<td>nom</td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>State</th>
<th>ioss</th>
<th>aonss</th>
<th>var1ss</th>
<th>var2ss</th>
<th>var3ss</th>
<th>var4ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>nom</td>
<td>nom</td>
<td>!off</td>
<td>!off</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>var1off</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>!off</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>var2off</td>
<td>nom</td>
<td>nom</td>
<td>!off</td>
<td>off</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>var4off</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>!off</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>alloff</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>off</td>
<td>nom</td>
<td>turbo</td>
</tr>
</tbody>
</table>
## Top-Level Power States

<table>
<thead>
<tr>
<th>State</th>
<th>ioss</th>
<th>aonss</th>
<th>var1ss</th>
<th>var2ss</th>
<th>var3ss</th>
<th>var4ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>nom</td>
<td>nom</td>
<td>!off</td>
<td>!off</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>var1off</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>- (any)</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>var2off</td>
<td>nom</td>
<td>nom</td>
<td>!off</td>
<td>off</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>var4off</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>- (any)</td>
<td>nom</td>
<td>turbo</td>
</tr>
<tr>
<td>alloff</td>
<td>nom</td>
<td>nom</td>
<td>off</td>
<td>off</td>
<td>nom</td>
<td>turbo</td>
</tr>
</tbody>
</table>

```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 Verification
SoC Test-bench and Test-bench UPF

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

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

- Test-bench (contd....)

```vhdl
// 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
```
Questions?
Announcements

- Please fill out the survey/feedback form

- Need a copy of the IEEE 1801 UPF standard?
  - Available at no charge via the IEEE Get™ Program

- Interested in working on UPF? Join the working group ...
  - send an email to [info@p1801.org](mailto:info@p1801.org) for information about joining
  - or visit [http://standards.ieee.org/develop/wg/UPF.html](http://standards.ieee.org/develop/wg/UPF.html)
Thank You!