Next Generation Design and Verification Today

New Developments in UPF 3.0

Erich Marschner, Vice-Chair, IEEE P1801 WG
P1801: IEEE-SA Entity Based Work Group
IEEE 1801 (UPF) timeline

- **2006**: UPF-1.0 Kick off Meeting
- **2007**: UPF-1.0 Donated to IEEE
- **2008**: New Project Kick off Meeting
- **2009**: UPF-1.0 Revision of 1801-2009
- **2010**: CPF-2.0 Donated to IEEE
- **2011**: New Project Revision of 1801-2013
- **2012**: New Project Amendment of 1801-2013
- **2013**: New Project
- **2014**: 1801-2013 (AKA UPF-2.1) Published
- **2015**: 1801a-2014 (AKA UPF-2.2) Published
- **2016**: 1801-2015 (AKA UPF-3.0) Planned

- Accellera UPF-1.0 Published
- IEEE1801-2009 (AKA UPF-2.0) Published
- 1801-2013 (AKA UPF-2.1) Published
- 1801a-2014 (AKA UPF-2.2) Published
- 1801-2015 (AKA UPF-3.0) Planned
Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept

- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition

- **Component Level Power Modeling**
  - Power States and Power Consumption Functions
### UPF 1.0 Design Flow

- **RTL is augmented with UPF**
  - To define power management architecture

- **RTL + UPF verification**
  - To ensure that power architecture completely supports planned power states of design
  - To ensure that design works correctly under power management

- **RTL + UPF implementation**
  - Synthesis, test insertion, place & route, etc.
  - UPF may be updated by user or tool

- **NL + UPF verification**
  - Power aware equivalence checking, static analysis, simulation, emulation, etc.
UPF 1.0 Flow Issues

- **Power Aware Verification requires complete supply distribution network**
  - Supplies determine when each power domain is on (normal) or off (corrupted)

- **Supply networks are not defined until system implementation**
  - Part of integrating the whole system together

- **So power aware verification cannot begin until implementation is specified**
  - Limits how much the schedule can be shortened by parallel development
  - Must be redone entirely if the design is retargetted to a different technology

- **And debugging power management issues becomes more difficult**
  - Is a failure due to
    - Incorrect implementation?
    - A power management architecture flaw?
    - Misuse of an IP block?
    - Some combination of the above?
UPF 1.0 Power Intent Specification

- Power Domain definitions
  - elements
  - supply connections
- Supply Ports and Supply Nets
  - and their connections
- Power Switches
  - supply connections
  - control inputs
- Isolation Strategies
  - clamp values
  - supply connections
  - control inputs
- Level Shifting Strategies
  - supply connections
- Retention Strategies
  - supply connections
  - control inputs
- Port States
  - states
  - voltages
- Power State Tables (PSTs)
  - combinations of port states

All of these are intermingled in a UPF 1.0 file

And IP Usage Requirements are minimal
Solution: Partition UPF into Layers

- **IP Usage Requirements**
  - For any given IP block,
    - How can this IP be used in a power-managed design?
    - What must the design ensure so the IP block can function correctly?

- **Power Management Architecture**
  - For each IP instance in the design,
    - What power states will it be in?
    - What state will be retained?
    - What ports will be isolated
    - What control logic will be involved?

- **System Implementation**
  - For the system as a whole,
    - What technology will be used?
    - What does this imply about voltages, level shifters, and isolation cell locations?
    - How will power be supplied to the system?
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”
UPF Command Layers

**Constraint UPF**
- Atomic power domains
- Clamp value requirements
- Retention requirements
- Fundamental power states
- Legal/illegal states/ transitions

**Configuration UPF**
- Actual power domains
- Additional domain supplies
- Additional power states
- Isolation and Retention strategies
- Control signals for power mgmt

**Implementation UPF**
- Voltage updates for power states
- Level Shifter strategies
- Mapping to Library power mgmt cells
- Location updates for isolation
- Supply ports, nets, switches, and sets
- Port states and Power state tables

**Constraint Commands**
- create_power_domain
- set_port_attributes
- set_design_attributes
- set_retention_elements
- add_power_state
- describe_state_transition

**Configuration Commands**
- create_composite_domain
- create_power_domain -update
- add_power_state -update
- set_isolation
- set_retention
- create_logic_port
- create_logic_net
- connect_logic_net

**Implementation Commands**
- add_power_state -update
- set_level_shifter
- map_retention
- use_interface_cell
- set_isolation -update
- create_supply_port
- create_supply_net
- create_power_switch
- create_supply_set
- associate_supply_set
- add_port_state
- create_pst, add_pst_state

UPF 1.0
UPF 2.0
Incremental Verification

Power Mgmt Architecture

State-Based (Logical) Power Aware Verification
Technology Independent

System

Implementation UPF
Power Aware Verification
Voltage-Based (Electrical) Technology Dependent

IP1
IP2
IP3
configuration UPF

IP1
IP2
IP3
Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept

- **Power State Definition and Refinement**
  - Power State Definition with `add_power_state`
  - Power State Composition

- **Component Level Power Modeling**
  - Power States and Power Consumption Functions
What is a “Power State”? 

A named set of object states

- Each state has a “defining expression”
- It refers to values of the object’s “characteristic elements”
- Some characteristic elements may be don’t cares for a given state
- Multiple object states may satisfy the defining expression

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- A == 1'b0 && B == 1'b0
- (A xor B) == 1'b1
- A == 1'b1 && B == 1'b1
Power States as Sets

- Largest set = all possible object states
- Some of these states are legal states
- Subsets represent “more specific” (or more refined) power states
  - Refinement creates subsets by adding more conditions to satisfy
  - The innermost subset containing a given object state represents the most specific power state of that object
- Supersets represent “more general” (or more abstract) power states
- Non-overlapping subsets represent mutually exclusive power states
- Subset containment implies non-mutex power states (subset => superset)
Power State Definition Rules

You can:

- Define (legal) states
- Define explicitly illegal states
- Specify -complete to make undefined states illegal
- Define **Definite** subset states (existing state AND new condition)
- Define **Indefinite** superstates ([X]OR of existing states)
- Mark existing legal states illegal

You cannot:

- Create legal states in illegal state space
- Define superstates that are the AND of two or more existing states
Applying These Concepts

- Same level states must be mutually exclusive
- Superstates contain (overlap) substates - non-mutex
- These principles allow state partitioning, hierarchical refinement

**Fundamental Power States**

- G0: Working
  - S0: Awaymode
- G1: Sleeping
  - S1: Power on Suspend
  - S2: CPU off
  - S3: Standby
  - S4: Hibernation
- G2: Soft Off
- G3: Mech. Off

**Refined Power States (Substates)**

- {All States} represents the set of all possible states; the fundamental states are subsets of {All States}
Defining Hierarchical Power States

add_power_state -model CPU
- state {UP -logic_expr {...} } \n- state {UP.ACTIVE -logic_expr {...} } \n- state {UP.ACTIVE.P0 -logic_expr {...} } \n- state {UP.ACTIVE.P1 -logic_expr {...} } \n- state {UP.ACTIVE.P2 -logic_expr {...} } \n- state {UP.IDLE -logic_expr {...} } \n- state {UP.CLKGATED -logic_expr {...} } \n- state {DOWN -logic_expr {...} } \n- state {DOWN.RET -logic_expr {...} }
Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept

- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition

- **Component Level Power Modeling**
  - Power States and Power Consumption Functions
Power State Dependencies

- **Instance**
  - Functional modes as power states
  - Based on module states

- **Module**
  - Functional modes as power states
  - Based on component states, control inputs

- **Composite Domain**
  - Functional modes as power states
  - Based on subdomain states, control inputs

- **Power Domain**
  - Operational modes as power states
  - Based on supply set states, control inputs

- **Supply Set**
  - Supply function combinations as power states
  - Based on individual supply function electrical states (and voltages), clock frequency, control inputs

- **Supply Function**
  - Electrical states/voltages as power states
  - Based upon supply net/port states/voltages
  - Determined also by supply_on/off calls from testbench (for unassociated supply sets)

- **Supply Net**
  - Electrical states/voltages as power states
  - Based upon supply net/port states/voltages
  - Determined also by supply net resolution (for resolved supply nets)

- **Supply Port**
  - Electrical states/voltages as power states
  - Determined by supply_on/off calls from testbench (for primary supply inputs)
  - Determined also by power switches (for switch output ports)
  - NOT based on port state definitions
    - no way to refer to them today

Named power states (add_power_state)
Supply states (supply_net_type values)
Power State References

- **Supply Set** power states
  - can refer to SS function supply states

- **Power Domain** power states
  - can refer to supply set power states

- **Composite Domain** power states
  - can refer to subdomain power states and/or supply set power states

- **Group** power states
  - can refer to power states of any object at or below the same scope

- **Module** power states
  - can refer to power states of any object at or below the module scope

- **Instance** power states
  - inherit (upwards) power states of the instantiated module
  - can override legality of a power state for a given instance (make a legal state illegal)

* not showing supply refs to ports/nets or control conditions

**Architecture oriented** (useful for SLP)

**Implementation oriented** (useful for power intent)
Power State Composition

- Fundamental power states of a given object are mutually exclusive.

- Power states of two different objects are by default independent.
  - All combinations of the legal states of each are legal.

- An object that consists of other objects can:
  - Define named combinations of the states of its component objects.
    - Some of these are fundamental power states and therefore must be mutex.
  - Mark a named combination of component objects states as illegal.
  - Mark the set of named combinations as complete - which makes all others illegal.
  - In particular:
    - supply set states define named combinations of supply set function (supply) states.
    - domain states define named combinations of the domain’s supply set states.
    - composite domain states define named combinations of the subdomain states.

- An object that contains other objects can do the same (UPF 3.0):
  - In particular:
    - group power states name combinations of states of objects at/below the group scope.
    - module power states name combinations of states of objects in/below the module scope.
    - module states become instance states when the module is instantiated.

- A legal module state can be marked illegal for a given instance.
Example
Agenda

- **Successive Refinement**
  - Elaborating the UPF 2.0 Concept

- **Power State Definition and Refinement**
  - Power State Definition with add_power_state
  - Power State Composition

- **Component Level Power Modeling**
  - Power States and Power Consumption Functions
Energy Consumption Varies w/ Usage

Highly dynamic operation of multiple interacting hardware and software components

Usage Scenarios
- Idle
- Walking
- Driving
- Navigating
- Talking
- Browsing
- VideoPlay
- AudioPlay

Combined Scenarios:
- Navigating/Talking
- Browsing/3G
- VideoPlay/3G

User Activity (Keys, Touchscreen)

Sensors (CCD, Accelerometer)

Radios (GPS, WiFi, GSM)

Java Application

Android Java Power Manager

Sensors Control

Audio/Video Control

Suspend Control

GPS Control

Lights On/Off/Dim

Mode Control

Idle/Run

Suspend & Resume

Application Processor
### Need to Model Energy Usage

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Current</th>
<th>Diff = 3 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CursorA</td>
<td>CursorB</td>
<td>10 s</td>
</tr>
<tr>
<td>0x2</td>
<td></td>
<td>13 s</td>
</tr>
</tbody>
</table>

#### Power State Trace
- **PowerAnalysis**
  - $\$ Slice By Metric
  - $\$ Sliced

#### Frequency Trace
- **PowerAnalysis**
  - Frequency
  - 200.0

#### Voltage Trace
- **PowerAnalysis**
  - Voltage
  - 1.2

#### Dynamic Power Stats
- **PowerAnalysis**
  - $\$ Sum
  - $\$ No Slicing

#### Leakage Power Stats
- **PowerAnalysis**
  - $\$ Sum

#### Total Power Stats
- **PowerAnalysis**
  - $\$ Sum

---

Dynamic energy

Total energy

---

[Accellera Systems Initiative Logo]
Each State Has Different Power Reqs.

Power State Machine

- Off uW/State
- Standby uW/State
- Transmit uW/State
- Receive uW/State

mW/State

Power Analysis
Power Model Components

- Power state enumeration
  - Steady states
  - Transient states (transitions)
  - Power dissipation function per state
    - With relevant parameters
      - Voltage, frequency, event rates, ...
    - Returns Static + Dynamic power
  - PVT independent

- Power consumption data
  - PVT specific parameters
  - Characterized or estimated

- Power state activation
  - Scenario-based or functional simulation based
  - Resolution limits overall accuracy of power model

Addressed in UPF 3.0
add_power_state -model CPU -update
  -state {UP -power_expr {fU ...} } \ 
  -state {UP.ACTIVE -power_expr {fA ...} } \ 
  -state {UP.ACTIVE.P0 -power_expr {f0 ...} } \ 
  -state {UP.ACTIVE.P1 -power_expr {f1 ...} } \ 
  -state {UP.ACTIVE.P2 -power_expr {f2 ...} } \ 
  -state {UP.IDLE -power_expr {fI ...} } \ 
  -state {UP.CLKGATED -power_expr {fC ...} } \ 
  -state {DOWN -power_expr {fD ...} } \ 
  -state {DOWN.RET -power_expr {fR ...} } 

Power expression of the “current” power state would be the natural one to use for power computations

More refined power states would have more detailed power functions
For More Information On …

- **Successive Refinement of UPF Power Intent**
  - See paper/presentation/poster
  - *Successive Refinement: A Methodology for Incremental Specification of Power Intent*
    - by A. Khan, E. Quiggley, J. Biggs (ARM); E. Marschner (Mentor Graphics)
    - Session 8: Low Power Verification (Weds 10:00-11:30am; Oak)

- **Power State Definition and Refinement**
  - See paper/presentation
  - *Unleashing the Full Power of UPF Power States*
    - by E. Marschner (Mentor Graphics), J. Biggs (ARM)
    - Session 3: Design (Tues 9:00-10:30am; Monterey/Carmel)

- **Component Power Modeling**
  - Join the P1801 Working Group and the System Level Power (SLP) subgroup
    - Visit the web page at [http://standards.ieee.org/develop/project/1801.html](http://standards.ieee.org/develop/project/1801.html)
    - Or send a request for information to admin@p1801.org
Thank you!