Successive Refinement: A Methodology for Incremental Specification of Power Intent

Adnan Khan, Eamonn Quigley, John Biggs
Erich Marschner

[adnan.khan@eamonn.quigley@john.biggs@erich.marschner@mentor.com]

**Introduction**

IEEE 1801 UPF enables early specification of “power intent”, or the power management architecture of a design, so that power management can be taken into account during design verification and verified along with design functionality. The verified power intent then serves as a golden reference for the implementation team.

To fully realize the advantages of this capability, a methodology called Successive Refinement was conceived during development of IEEE 1801-2009 UPF. However, this methodology is still not well understood in the industry.

In this paper, we present the UPF Successive Refinement methodology in detail. We explain how power management constraints can be specified for IP blocks to ensure correct usage in a power-managed system. We explain how a system’s power management architecture can be specified in a technology-independent manner and verified abstractly, before implementation. We also explain how implementation information can be added later. Finally, we explain the benefits of Successive Refinement.

**Successive Refinement**

**Constraint PF for an IP Block**

1. Define Atomic Power Domains

   - Create the cluster power domain `create_power_domain CORTEX`
   - Create the VDD power domain `create_power_domain VDDCPU`
   - Create the SMD power domain `create_power_domain VDDCPU`

2. Define Retention Constraints

   - Set retention elements `set_retention_elements VDDCPU -elements {u_ca_cpu0}`

3. Define Isolation Constraints

   - Define `isolation_sense low`
   - Define `isolation_signal nISOLATECPU0`
   - Define `nISOLATECPU0`

4. Define Fundamental Power States

   - Add power state `add_power_state`
     - state `RUN` {`logic_expr (primary == ON)`}
     - state `OFF` {`logic_expr (primary == OFF)`}

5. Define Abstract Retention Power States

   - Add power state `add_power_state`
     - state `RUN` {`logic_expr (primary == OFF)`
     - state `OFF` {`logic_expr (primary == OFF)`}

6. Define Illegal Power State Combinations

**Configuration PF for an IP Instance**

1. Load Constraint UPF Files onto Instances

   - `load_upf`_runtime

2. Define Control Logic

   - Create logic port `create_logic_port`
     - direction `IN`
     - name `nWTRFORCPU`
     - create logic set `create_logic_set`
       - name `WTRFORCPU`

3. Define Retention Strategy

   - Set retention `set_retention`
     - set `set_power_state`
     - state `OFF`

4. Define Isolation Strategies

   - Add isolation `add_isolation`
     - name `u_ca_cpu0`

5. Define Power States with Control Conditions

   - Add power state `add_power_state`
     - state `RUN`
     - state `OFF`

6. Define Illegal Power States as Required

**Power Distribution for the System**

1. Load Configuration UPF File

2. Create Supply Network Elements

   - Create supply port `create_supply_port`

3. Create Power Switches

   - Create power switch `create_power_switch`

4. Update Supply Sets with Supply Nets

   - Create supply set `create_supply_set`

5. Update Power States with Voltages

   - Add power state `add_power_state`

6. Specify Other Technology Info As Required

**Benefits of Successive Refinement**

- Clear communication between IP provider and IP consumer
- Decreases risk and facilitates successful usage
- Separation of logical design from implementation
- Verification can start earlier, before technology is known
- Easier re-targeting to different technologies
- Easier debugging at each stage
- Preservation of verification equity
- No need to re-verify logical configuration for new technology