Successive Refinement: A Methodology for Incremental Specification of Power Intent

Adnan Khan, Eamonn Quigley, John Biggs — [adnan.khan/eamonn.quigley/john.biggs]@arm.com

Erich Marschner — erich_marschner@mentor.com

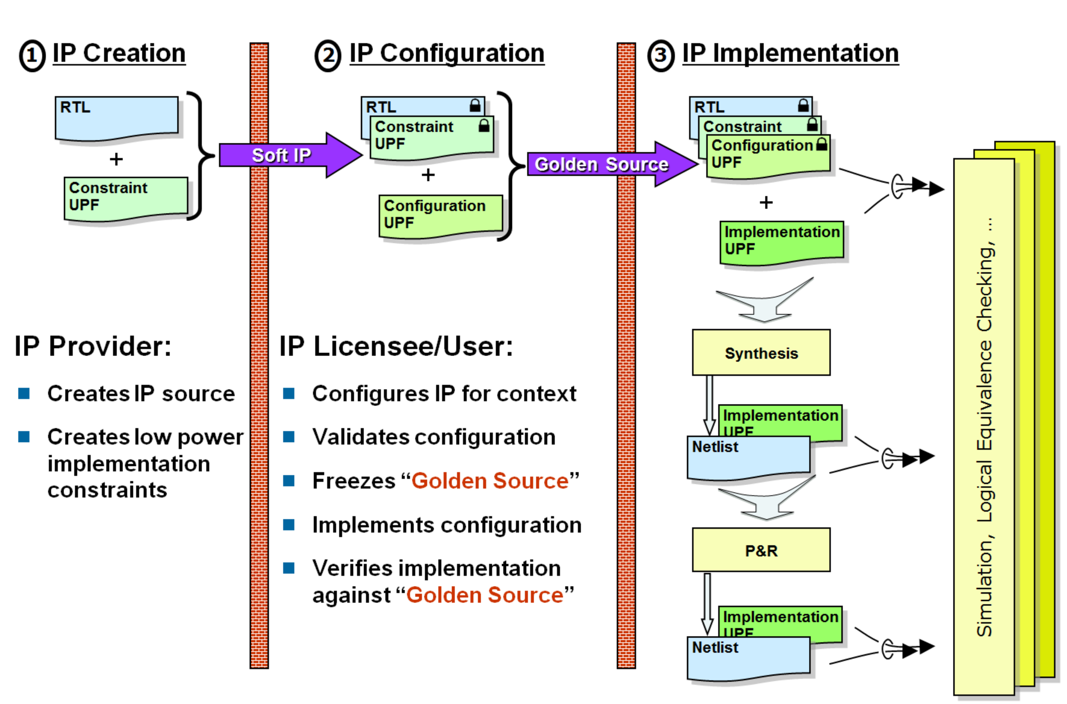
Introduction

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 flow.

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



Power Intent Partitioning

Constraint UPF:

- Describes the power intent inherent in the IP
  - power domains/states/isolation/retention etc.
- Constraints are part of source IP and travel with the RTL

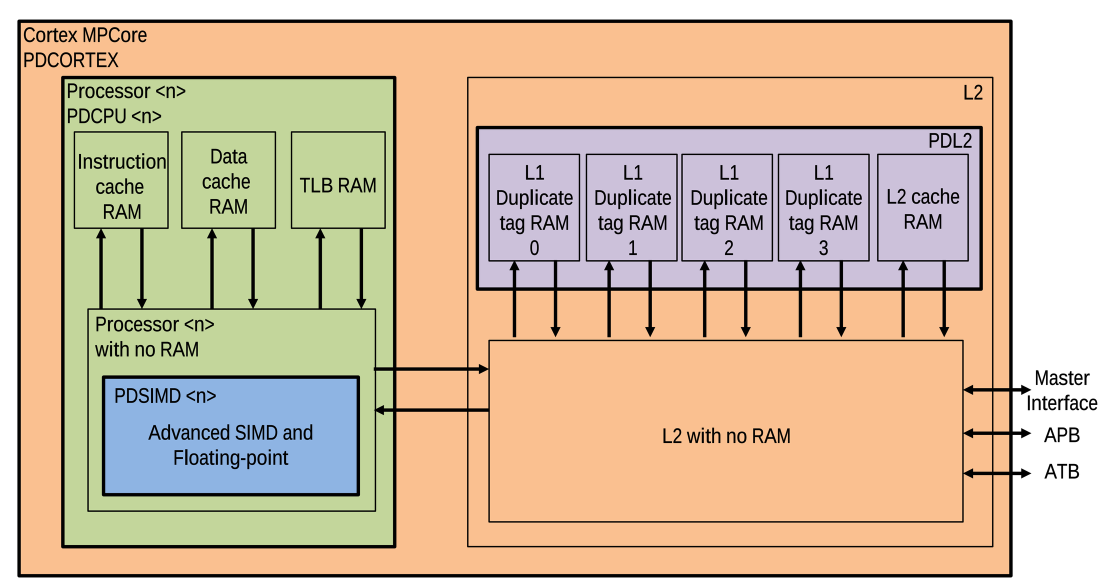
Configuration UPF:

- Describes application-specific configuration of instances
  - supply sets, power states, logic expressions, etc.
- Required for simulation - created by end user

Implementation UPF:

- Describes technology-specific implementation of system
  - supply nets/ports, switches, etc.
- Required for implementation - created by end user

ARM® Cortex®-MPCore Processor IP



Constraint UPF for an IP Block

1. Define Atomic Power Domains

```
# Create the cluster power domain
create_power_domain PDCORTEX \
  –elements {.} -atomic

# Create the CPU0 power domain
create_power_domain PDCPU0 \
  –elements "u_ca_cpu0" -atomic

# Create the SIMD power domain
create_power_domain PDSIMD0 \
  –elements "u_ca_advsimd0" -atomic

# Create the L2 Cache domain
create_power_domain PDL2 –elements \
  "u_ca_l2/u_ca_l2_datarams \
   u_ca_l2/u_ca_l2_tagrams \
   u_ca_l2/u_cascu_l1d_tagrams" \
  -atomic
```

2. Define Retention Constraints

```
set_retention_elements PDCPU0_RETN –elements "u_ca_cpu0"
```

3. Define Isolation Constraints

```
# default is isolate low
set_port_attributes –elements "u_ca_hierarchy" \
  -applies_to outputs \
  -clamp_value 0

# some ports need to be clamped high
set_port_attributes \
  -ports "u_ca_hierarchy/out1" \
  -clamp_value 1
```

4. Define Fundamental Power States

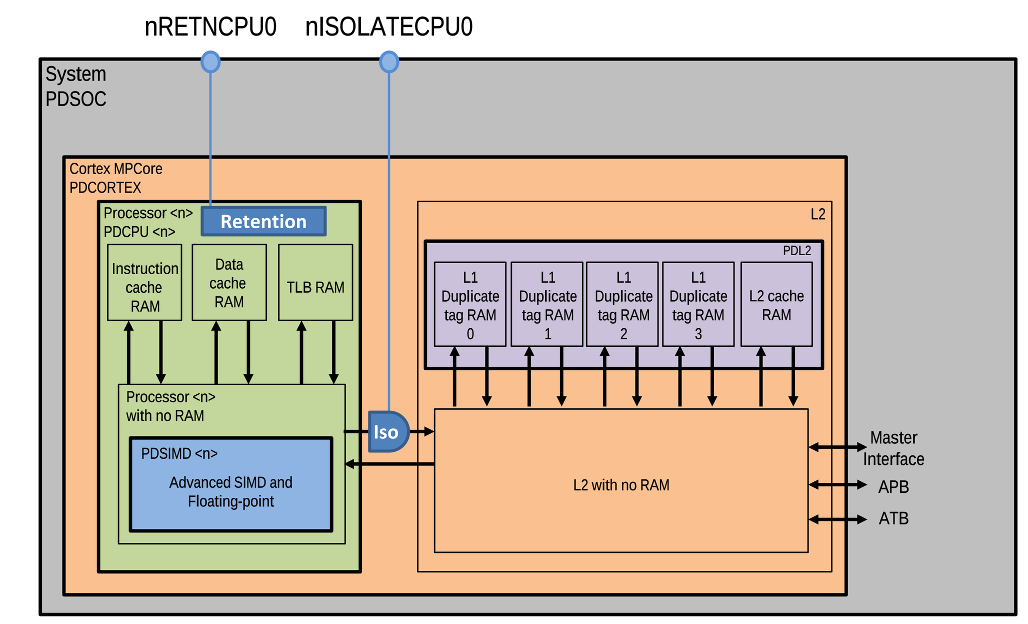
```
add_power_state PDSIMD0 –domain \
  -state {RUN -logic_expr {primary == ON}} \
  -state {SHD -logic_expr {primary == OFF}}

add_power_state PDSIMD0.primary –supply \
  -state {ON  -simstate NORMAL \
    -supply_expr {power == FULL_ON && ground == FULL_ON}} \
  -state {OFF -simstate CORRUPT \
    -supply_expr {power == OFF}}
```

5. Define Abstract Retention Power States

```
add_power_state PDSIMD0 –domain –update \
  -state {RET \
    -logic_expr {primary==OFF && default_retention==ON}}
```

6. Define Illegal Power State Combinations

A System with an Instance of the IP Block



Configuration UPF for an IP Instance

1. Load Constraint UPF Files onto Instances

```
load_upf -scope MPCoreInst CORTEX_constraints.upf
…
```

2. Define Control Logic

```
create_logic_port -direction in nRETNCPU0
create_logic_net nRETNCPU0
connect_logic_net nRETNCPU0 -ports nRETNCPU0
…
```

3. Define Retention Strategy

```
set_retention ret_cpu0 -domain PDCPU \
  -retention_supply_set PDCPU0.default_retention \
  -save_signal "nRETNCPU0 negedge" \
  -restore_signal "nRETNCPU0 posedge"
```

4. Define Isolation Strategies

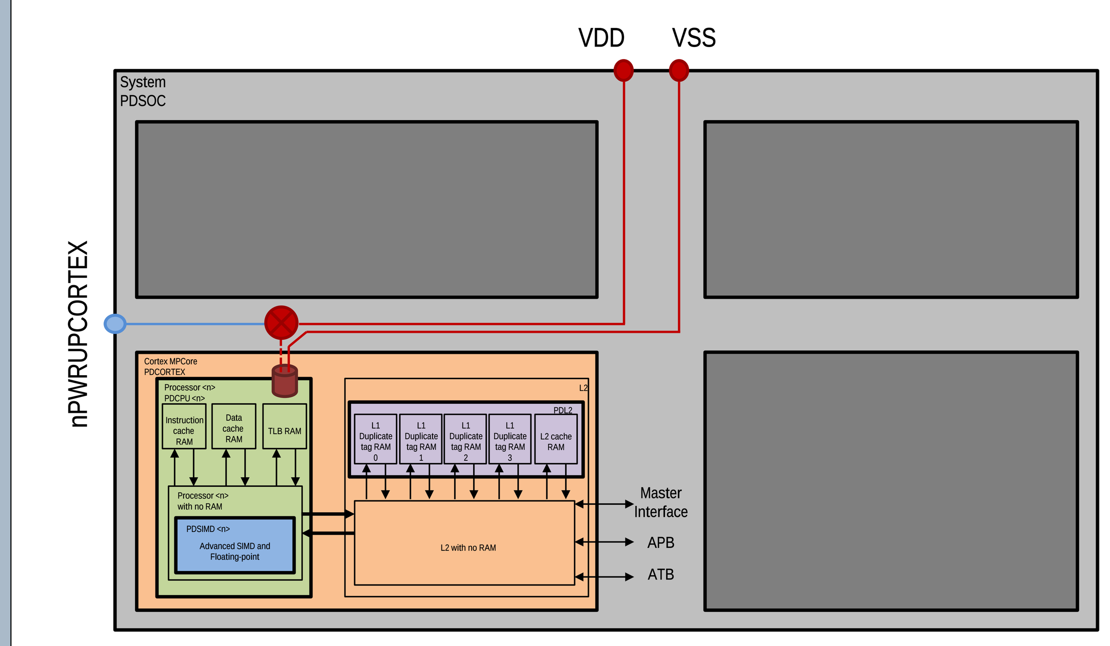
```
# -------- cpu clamp 0 ---------
set_isolation iso_cpu_0 -domain PDCPU0 \
  -isolation_supply_set PDCORTEX.primary \
  -clamp_value 0 \
  -applies_to outputs \
  -isolation_signal nISOLATECPU0 \
  -isolation_sense low

# -------- cpu clamp 1 ---------
set_isolation iso_cpu_1 -domain PDCPU0 \
  -isolation_supply_set PDCORTEX.primary \
  -clamp_value 1 \
  -elements "u_ca_hierarchy/out1" \
  -isolation_signal nISOLATECPU0 \
  -isolation_sense low
```

5. Update Power States with Control Conditions

```
add_power_state PDSIMD0 –domain -update \
  -state {RUN \
    -logic_expr {nPWRUP_SIMD0==0 && nPWRUPRetn_SIMD0==0}} \
  -state {RET \
    -logic_expr {nPWRUP_SIMD0==1 && nPWRUPRetn_SIMD0==0 && \
                 nRETN_SIMD0 ==0 && nISOLATE_SIMD0  ==0}} \
  -state {SHD \
    -logic_expr {nPWRUP_SIMD0==1 && nPWRUPRetn_SIMD0== 1}}

add_power_state PDSIMD0.primary –supply -update \
  -state {ON  -logic_expr {nPWRUP_SIMD0 == 0}} \
  -state {OFF -logic_expr {nPWRUP_SIMD0 == 1}}

add_power_state PDSIMD0.default_retention –supply -update \
  -state {ON  -logic_expr {nPWRUPRetn_SIMD0 == 0}} \
  -state {OFF -logic_expr {nPWRUPRetn_SIMD0 == 1}}
```

6. Refine Power States as Required

Power Distribution for the System



Implementation UPF for the System

1. Load Configuration UPF File

```
load_upf SoC_configuration.upf
```

2. Create Supply Network Elements

```
create_supply_port VDD
create_supply_port VSS

create_supply_net  VDDCORTEX -domain PDCORTEX
create_supply_net  VDDCPU0 -domain PDCPU0
create_supply_net  VDDRCPU0 -domain PDCPU0
```

3. Create Power Switches

```
create_power_switch ps_CORTEX_primary -domain PDCORTEX \
  -input_supply_port { VDD VDD } \
  -output_supply_port { VDDCORTEX VDDCORTEX } \
  -control_port { nPWRUPCORTEX nPWRUPCORTEX } \
  -on_state { on_state VDD {!nPWRUPCORTEX} } \
  -off_state { off_state {nPWRUPCORTEX} }
…
```

4. Update Supply Sets with Supply Nets

```
create_supply_set PDCPU0.primary -update \
  -function {power VDDCPU0} -function {ground VSS}

create_supply_set PDCPU0.default_retention -update \
  -function {power VDDRCPU0} -function {ground VSS}
…
```

5. Update Power States with Voltages

```
add_power_state PDCPU0.primary –supply -update \
  -state {ON  -supply_expr {power  == {FULL_ON 0.81} && \
                            ground == {FULL_ON 0.00}}} \
  -state {OFF -supply_expr {power == OFF || \
                            ground == OFF}}
…
```

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 retargeting to different technologies
  - Easier debugging at each stage
- Preservation of Verification Equity
  - No need to re-verify logical configuration for new technology