Successive Refinement – An approach to decouple Front-End and Back-end Power Intent

Sinha Rohit Kumar
Intel Technology Private Ltd
Bangalore, India -560103
rohit.kumar.sinha@intel.com

Kotha kavya
Intel Technology Private Ltd
Bangalore, India -560103
kotha.kavya@intel.com

***Abstract*- IEEE 1801 UPF [1] format comes with a limitation that it doesn't entirely support decoupling of front and backend power intent files and as many SoC projects in Intel are marching towards ASIC products on different process technologies, it becomes all the more important for designers to code power intent with the process agnostic approach. Therefore, IEEE 1801-2015 UPF (UPF3.1) [2]has come up with a methodology called Successive refinement that supports Incremental specification. This methodology enables incremental design and verification of the power management architecture, and it is specifically designed to support specification of power management requirements for IP components used in a low power design. This incremental flow accelerates design and verification of the power management architecture using partition methodology wherein the power intent is partitioned into constraints, configuration, and implementation. In this paper, we will present the new methodology Successive refinement implemented for IOTG-SOC in which power intent is specified in a technology independent manner and verified abstractly before implementation.**

I. INTRODUCTION

Use of IP in SoC's is essential in order to meet time-to-market requirements and leveraging existing technologies efficiently. So, designing the power management mechanism should involve both IP requirements and SoC concerns. UPF(IEEE 1801 UPF) format which we are using now follows an implementation-methodology that provides power-management structures and behavior for a design which drives both verification and implementations steps. There are some problems with this methodology.

In this paper, we will present the new methodology Successive refinement in which power intent is specified in a technology independent manner and verified abstractly before implementation

Following are the challenges of using implementation oriented UPF

- After investing a lot of verification effort to prove that the strategy and UPF file are correct, if the UPF file has to be modified or re-created after selection of the target process technology, the verification equity built up is lost and the verification process has to be repeated with associated delays and extra resource costs. So, power aware verification is often postponed until late in the flow
- IPs are re-used either in different SoC's, different generation of same SoC's or in different target technologies. So, if IP UPF contains implementation details, it should be re-generated based on the customer's usage.

Successive Refinement addresses both of these issues. This methodology defines

- How an IP provider can provide UPF that specify constraints on the use of an IP component within a system, without knowledge of the characteristics of the system.
- How UPF can be used by the system integrator to specify the logical configuration of power management for the individual IP components used in the system and for the system as a whole. This enables early verification of the power management architecture before any implementation decisions are made.
- How UPF can be used by the system implementer to realize the power intent in the context of a given technology and implementation approach

The concept and methodology come with an idea of adding detailed description of power management in different files by separating the logical functionality of power management for a system from the technology-specific implementation of the system.

II. Successive Refinement And Its Challenges

Successive refinement in UPF allows an IP provider to capture the low power constraints inherent in an IP block without predicting a particular configuration. Then any customer who uses that IP can configure the IP, within these constraints, for their particular application without predicating a particular technology specific implementation. The result is a simulatable but technology-independent golden source against which all technology specific implementations can be verified. In this way the verification of strategies and UPF file need not be repeated even if implementation details change.

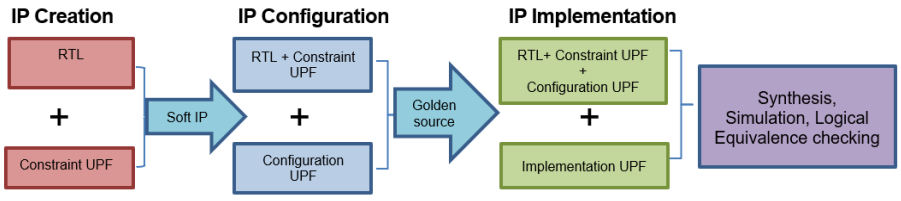

Figure 1. Successive Refinement Flow

Successive refinement essentially partitions the upf into three categories

A. *Constraint UPF*

The constraint UPF file is the most abstract view of power intent. This file is used to describe constraints on the power intent of the design, as it applies to a particular design component. This UPF describes the power intent inherent in the IP like power domains/states/isolation/retention etc. The IP developer creates Constraint UPF. The Constraint UPF file should not be replaced or changed by the IP consumer. It is completely verified by the IP owner and IP consumer can use it for any power management implementation Approach. The Constraint UPF contains following :

1) Defining Atomic Power Domains

The Constraint UPF file defines each power domain that is identified in the specification for the IP component. The option –atomic indicates that this power domain cannot be further partitioned by the IP consumer. If the IP Customer doesn't want the power domain specified to be modified in the SoC, it can be specified with a regular power domain definition.

Power domain can be defined as follows

```
#####################################################################
# Power Domains
#####################################################################
create_power_domain PD_CLUSTER_AON   -include_scope \
                                     -supply "primary          SS_${SOC_VDD_CPU}" \
                                     -supply "extra_supplies_0 SS_$SOC_VDD_1V1_TS_CPU" \
                                     -supply "extra_supplies_1 SS_$SOC_VDD_1V8_PLL_A55" \
                                               -supply "extra_supplies_2 SS_$SOC_VDD_1V8_PLL_A76" \
                                               -supply "extra_supplies_3 SS_$SOC_VDD_1V8_PLL_DSU" -atomic


create_power_domain PDSYS    -elements "$PD_SYS_ELE" \
                                       -supply "primary SS_${SOC_VDD_CPU}_CLUSTER_GATED" \
                                     -supply "extra_supplies_0 SS_${SOC_VDD_CPU}" -atomic
```

Figure 2. Atomic power domains in constraint upf

Note: Tcl variable $ is used to represent list of elements , ports or exceptions which has same supply, power state or a clamp value

2) Define Isolation Requirements

Isolation is not actually specified in the constraint UPF file for an IP component. Implementation of retention, isolation is an implementation choice and is usually left for IP licensees to decide whether they would like to include

it in their design. However, it is necessary to specify which state elements must be retained, isolated if the user decides to make use of retention, Isolation in his power management scheme.The Constraint UPF file should specify the isolation clamp values that must be used if the user decides to shutdown portions of the system as part of this power management scheme.

The command 'set_port_attributes' is used to define the clamp value requirements:

```
#####################################################################
## specify the isolation clamp values that must be used
#####################################################################
set_port_attributes -ports $DSU_CLAMP1 \ -clamp_value 1
set_port_attributes -ports $DSU_CLAMP0 \ -clamp_value 0
set_port_attributes -ports "big_little_cluster/CLUSTER/u_corinth_cluster/dsu_mbist_shell/big_little_cluster_rtl
_tessent_mbist_SRAM_c1_shared_bus_bisr_mux_inst/in0 big_little_cluster/CLUSTER/u_corinth_cluster/dsu_mbist_shel
l/big_little_cluster_rtl_tessent_mbist_SRAM_c2_shared_bus_bisr_mux_inst/in0" \ -clamp_value 0
```

Figure 3. Isolation Clamp Values

3) Define retention elements for strategy

Specify the retention on the state elements if the user decides to make use of retention in his power management scheme. set_retention_elements retn_list_<name> -elements [list <Retention elements>]

4) Define power states without voltage value

For constraint UPF, add_power_state should be used to define the fundamental power states of an IP block and its component domains in a technology-independent manner. This implies that power states should be defined without reference to voltage levels. Similarly, constraint UPF should not impose any particular power management approach on the IP consumer, so it should define power states without dictating how power will be controlled.

```
#########################################################################
# Power States
#########################################################################
add_power_state SS_$SOC_VDD_CPU \
                -state "SS_${SOC_VDD_CPU}_VM"    " -simstate NORMAL" \
                -state "SS_${SOC_VDD_CPU}_GND" " -simstate NORMAL" \
                -state "SS_${SOC_VDD_CPU}_OFF"  "-simstate CORRUPT"

add_power_state SS_${SOC_VDD_CPU}_CLUSTER_GATED \
                -state "SS_${SOC_VDD_CPU}_CLUSTER_GATED_VM"  "-simstate NORMAL" \
                -state "SS_${SOC_VDD_CPU}_CLUSTER_GATED_OFF"  "-simstate CORRUPT"
```

Figure 4. Fundamental power states

B. Configuration UPF

The IP consumer adds Configuration UPF describing his system design including how all the IP blocks in the system are configured and power managed. The Configuration UPF consists of

1) Define design ports

Add design ports that a design may use to control power management logic are defined using create_logic_port, create_logic_net, connect_logic_net and used for validation

```
#########################################################################
# Create logic Port
#########################################################################
create_logic_port $SOC_VDD_CPU                    -direction in
create_logic_port $SOC_GROUND                -direction in
create_logic_port $SOC_VDD_1V1_TS_CPU        -direction in
```

```
##########################################################################
# Create logic Net
##########################################################################
create_logic_net $SOC_VDD_CPU
create_logic_net $SOC_VDD_1V1_TS_CPU
##########################################################################
# Connect logic Net
##########################################################################
connect_logic_net $SOC_VDD_CPU            -ports $SOC_VDD_CPU
connect_logic_net $SOC_GROUND             -ports $SOC_GROUND
connect_logic_net $SOC_VDD_1V1_TS_CPU      -ports $SOC_VDD_1V1_TS_CPU
```

Figure 5. Logic ports, nets definition and connection

2) Define ISO/RET strategies and how they are controlled

In the configuration file, an isolation strategy must specify clamp values consistent with the specifications in the constraint UPF.

```
############################################
# ISO and retention strategies implementation
############################################
set_isolation iso_cluster_0 \
        -domain PDSYS \
        -isolation_supply_set SS_${SOC_VDD_CPU} \
        -source SS_${SOC_VDD_CPU}_CLUSTER_GATED \
        -sink SS_${SOC_VDD_CPU} \
    -exclude_elements $DSU_CLAMP0 \
        -isolation_signal $nISOLATECPU_PDSYS \
        -isolation_sense low \
        -location parent
```

Figure 6. Isolation Strategy

If there are any retention strategies used in the power domain those are also specified in the configuration UPF. The 'set_retention' command is used to specify a retention strategy.

C. Implementation UPF

This UPF file is used to provide the implementation details and technology specific information that is needed for the implementation of the design. It contains low level details of power switches and voltage rails (supply nets). It defines which supply nets are connected to the supply sets defined for each power domain. This file also defines the formation of any power switches that have been chosen for this implementation. This UPF contains

1) Define supply and network elements for the design

The first thing that needs to be done in Implementation UPF is definition of the supply nets and creation of the supply network. This can be done by using the commands 'create_supply_port' and 'create_supply_net' as shown below.

```
##########################################################################
# Create Supply Port
##########################################################################
create_supply_port $SOC_VDD_CPU                    -direction in
create_supply_port $SOC_GROUND                -direction in
create_supply_port $SOC_VDD_1V1_TS_CPU        -direction in
##########################################################################
# Create Supply Net
##########################################################################
create_supply_net $SOC_VDD_CPU
create_supply_net $SOC_VDD_1V1_TS_CPU
create_supply_net $SOC_VDD_1V8_PLL_A55
```

Figure 7. Supply nets and Supply ports

2) Defining Power Switches

This methodology requires IPs to no longer create power switches and it's up to the SOC Integrator to handle all the validation. The design team does not have to make any decisions prior to implementation UPF about the

formation of the switch design it needs for the implementation. This also helps to keep the constraint UPF and configuration UPF files in an abstract form that is used for RTL verification purposes.

```
#########################################################################
# Power switches
#########################################################################
set nPWRUPCPU_HAMMER_PDSYS
big_little_cluster/CLUSTER/nPWRUPCPU_HAMMER_PDSYS
set nPWRUPCPU_TRICKLE_PDSYS
big_little_cluster/CLUSTER/nPWRUPCPU_TRICKLE_PDSYS
set nPWRUPCPU_HAMMER_ACK_PDSYS
big_little_cluster/CLUSTER/nPWRUPCPU_HAMMER_ACK_PDSYS_DSU
set nPWRUPCPU_TRICKLE_ACK_PDSYS
big_little_cluster/CLUSTER/nPWRUPCPU_TRICKLE_ACK_PDSYS_DSU
set nISOLATECPU_PDSYS big_little_cluster/CLUSTER/nISOLATECPU_PDSYS
create_power_switch ps_PDSYS_main -domain PDSYS \
 -output_supply_port        "VDD SS_${SOC_VDD_CPU}_CLUSTER_GATED.power" \
   -input_supply_port           "TVDD SS_${SOC_VDD_CPU}.power" \
   -control_port               [list NSLEEPIN1 $nPWRUPCPU_HAMMER_PDSYS] \
   -control_port                [list NSLEEPIN2 $nPWRUPCPU_HAMMER_PDSYS]\
   -on_state                    { on TVDD { NSLEEPIN1 & NSLEEPIN2 }} \
   -off_state                   { off     { !NSLEEPIN1 | !NSLEEPIN2 }}
```

Figure 8. Power switches

3) Connecting supply nets with supply sets

The option –update is used to add the names of the supply nets to be connected to functions power, ground of the respective supply set

```
#########################################################################
# Create Supply Set
#########################################################################
create_supply_set SS_$SOC_VDD_CPU       -update            -function "power $SOC_VDD_CPU"
  -function "ground $SOC_GROUND"  -function "pwell $SOC_GROUND"    -function "nwell $SOC_VDD_CPU"
create_supply_set SS_${SOC_VDD_CPU}_CLUSTER_GATED -update    -function "power ${SOC_VDD_CPU}_CLUSTER_GATE
D"    -function "ground $SOC_GROUND"  -function "pwell $SOC_GROUND"    -function "nwell $SOC_VDD_CPU"
create_supply_set SS_${SOC_VDD_CPU}_CORE0_GATED   -update -function "power ${SOC_VDD_CPU}_CORE0_GATED"
 -function "ground $SOC_GROUND"  -function "pwell $SOC_GROUND"    -function "nwell $SOC_VDD_CPU"
```

Figure 9. supply sets

```
#########################################################################
# Connect Supply Net
#########################################################################
connect_supply_net $SOC_VDD_CPU            -ports $SOC_VDD_CPU
connect_supply_net $SOC_GROUND             -ports $SOC_GROUND
connect_supply_net $SOC_VDD_1V1_TS_CPU     -ports $SOC_VDD_1V1_TS_CPU
```

Figure 10. supply nets

4) Define Supply Voltages in power states

```
#########################################################################
# Power States
#########################################################################
add_power_state SS_$SOC_VDD_CPU \   -update
       -state       "SS_${SOC_VDD_CPU}_VM"       "-supply_expr       \{power       ==
`\{FULL_ON,$SOC_VDD_CPU_VM_MIN,$SOC_VDD_CPU_VM_NOM,$SOC_VDD_CPU_VM_MAX\}\}   -simstate
NORMAL" \       -update
       -state       "SS_${SOC_VDD_CPU}_GND"       "-supply_expr       \{ground       ==
`\{FULL_ON,$SOC_GROUND_VM_MIN,$SOC_GROUND_VM_NOM,$SOC_GROUND_VM_MAX\}\}      -simstate
NORMAL" \       -update
       -state      "SS_${SOC_VDD_CPU}_OFF"     "-supply_expr \{power ==  `\{OFF\}\} -
simstate CORRUPT"
```

Figure 11. power states

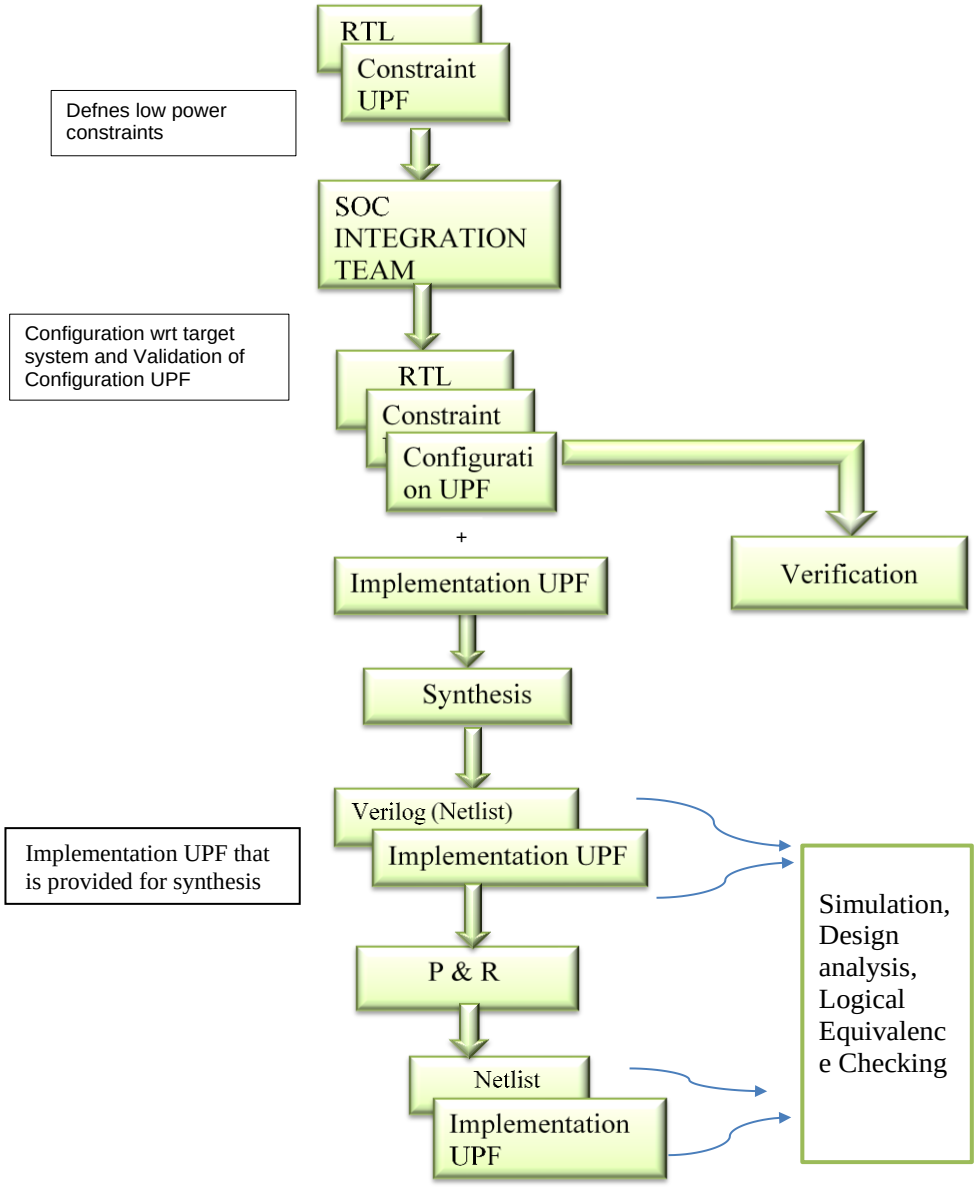

Figure 12. Design Low Power Flow Diagram with Successive refinement flow

Successive refinement design flow is illustrated in Figure 12 . The IP developer creates Constraint UPF that goes along with the RTL for a soft IP component. The Integrator adds Configuration UPF describing his system design including how all the IP blocks in the system are configured and power managed. The Configuration UPF loads in the constraint UPF for each IP so that tools can check that the constraints are met in the configuration. Once the configuration specified by the Integrater has been validated, the implementer adds Implementation UPF to specify implementation details and technology mapping. This complete UPF specification then drives the implementation process.

III. RESULTS

In Intel Client-SoC, we have implemented successive refinement methodology to decouple front-end UPF from BE-UPF and it was successfully accepted. Also, we have piloted it on IOTG-SoC design and work is in progress to scale it to multiple subsystems.

One of our clients SoC accepted IP power intent either:

- In process and project agnostic way – using “successive refinement” methodology

- Or in aligned with project SD(Structural Design) and topology – using traditional delivery

Internal IP delivered UPF following the first approach. UPF commands and options were limited by currently supported by all vendor flow tools. IP level validation was limited by tools as well.

- All layers (including implementation) were required for logic validation and simulation. LRM claims implementation portion only for structural design.
- Only UPF2.1 commands and options are used. E.g., associate_supply_set command -handle option was used to connect SoC with IP supplies.
- Updating of power state (add_power_state) with -illegal option was not allowed by tools. To avoid conflict between IP and SoC PST last one was disabled after integration for structural design.

SoC used bottom-up approach and consumed IP integration wrapper, which included constraint and configuration files. SPA with driver/receiver supply were put to configuration part. Most of configuration IP UPF were not updated by SoC (except removing redundant ISO after disabling internal gated PD). Implementation layer one per SD entity (e.g., partition) was automatically created by SoC. Partition UPF looks schematically:

```
load IP#1_wrapper.upf -scope IP#1
load IP#2_wrapper.upf -scope IP#2
…
load IP#N_wrapper.upf -scope IP#N
source partition implementation.upf
```

- If IP UPF is delivered using IEEE 1801 UPF format i.e., if the UPF contains implementation details it is consumed in the SoC and validated using traditional method

IP UPF in "successive refinement" format keeps backward compatibility with traditional way. All layers (including implementation) produce the same traditional UPF: the same content is described by about the same commands are specified in different order and in incremental way.

IV. SUMMARY

In one of our IP, we have implemented successive refinement methodology to decouple front-end UPF from BE-UPF and it was successfully accepted. Also, we have piloted it on CPU Subsystem design and work is in progress to scale it to multiple subsystems.

Below are some of the highlights

- A UPF power intent specification for a SoC with multiple IPs having different levels of physical hierarchical implementation and UPF specifications was created
- The UPF specifications for individual IPs were verified for structural checks through static checks and formal methods.
- The power models created for memories and PHY were elegant and re-use of the power model was achieved in the successive refinement process.
- The power intent of hard macros is modelled with power states for the hard macro.
- The higher-level interface of each IP was modelled through port attributes based on UPF interface scenario – IP or system level
- Each IP was verified with the UPF in their block level verification environment and at the top level using the SoC level verification environment.
- The use of UPF for IP block verification, IP hard macro implementation, SoC verification and SoC implementation was seamless with no changes to the IP or SoC UPF between processes

REFERENCES

[1] IEEE Std 1801™-2009, "IEEE Standard for Design and Verification of Low-Power Integrated Circuits," 27 March 2009. [Online].

[2] IEEE Std 1801™-2015, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems," 5 December 2015. [Online].

[3] [3] A. Khan, J. Biggs, E. Quigley and E. Marschner, "Successive Refinement: A methodology for incremental specification of power," DVCON, March 2015.K. Elissa, "Title of paper if known," unpublished.