Refining Successive Refinement

Desinghu PS, Adnan Khan - ARM Ltd UK

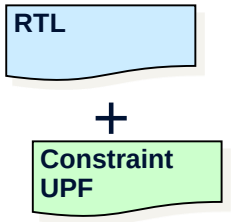
Erich Marschner, Gabriel Chidolue – Mentor Graphics

Agenda

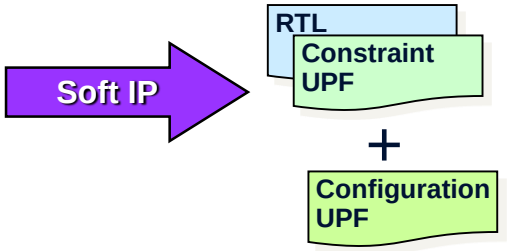
- Successive refinement flow – Overview
- Successive refinement Challenges in Complex SoC and recommended solutions
  - Handling Hard Macros
  - Isolation of UPF created control signals
  - Effective power state definitions
- Recommendations for solving challenges
- Results of application on recent design SoC
- Questions

Successive Refinement using UPF 2.0

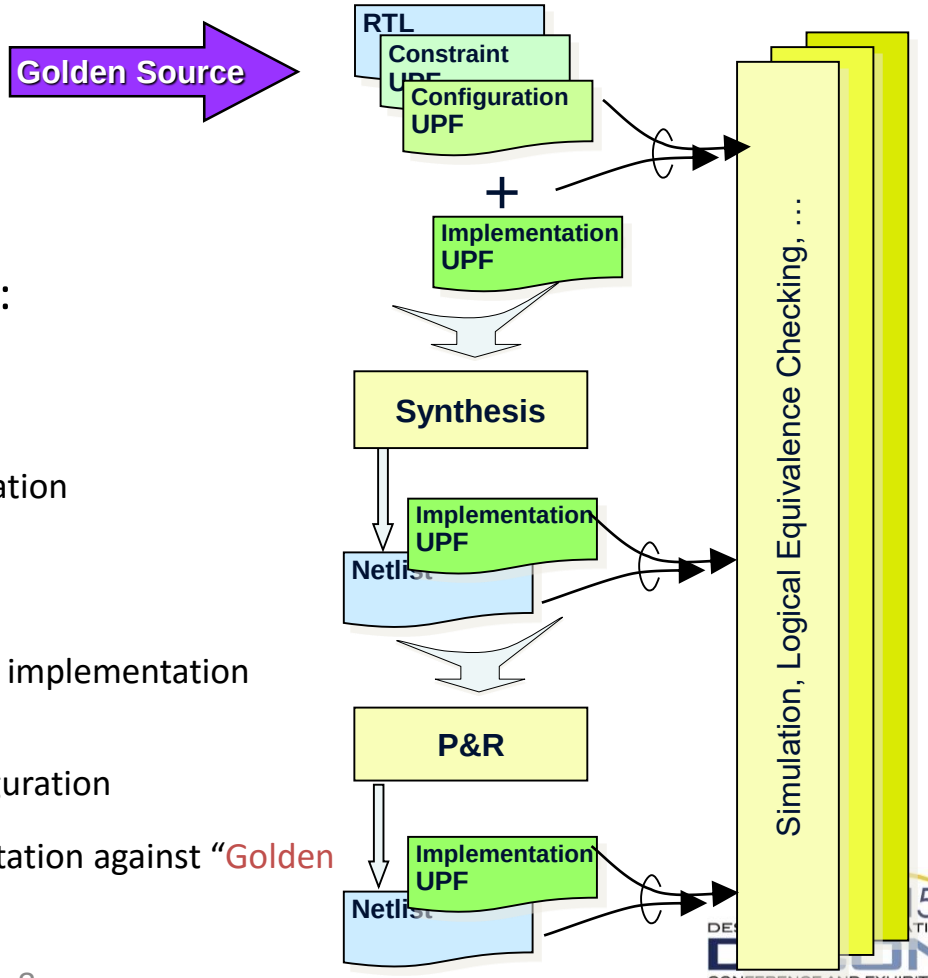
IP/Block Creation



IP/System Configuration



System Implementation



IP/Block Provider:

- Creates IP source
- Defines low power implementation constraints

IP/SoC Integrator:

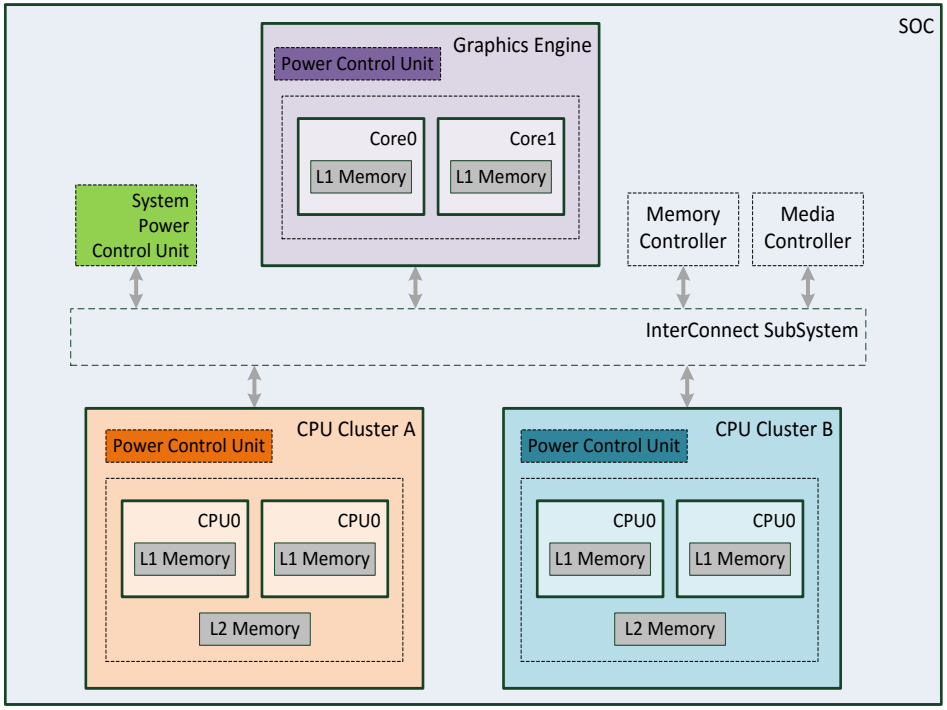
- Configures all IP's for target system
- Validates configuration
- Freezes "Golden Source"
- Adds power mgmt implementation detail
- Implements configuration
- Verifies implementation against "Golden Source"

* Used with permission

Constraints, Configuration and Implementation UPF content

- Logical / Technology independent UPF
  - Constraints UPF
    - Isolation, retention, atomic power domains, fundamental power states
  - Configuration UPF
    - Isolation & retention strategies, supply sets, power states
- Implementation UPF
  - Supply nets, power switches, supply expression for supply sets, other technology mapping info
- Separation of Concerns
  - Logical vs Implementation view of power architecture
  - Allows easier retargeting
  - Eases debug
- Early verification
  - Static checking of configuration UPF
  - Early dynamic verification of power architecture

Successive refinement: SoC Implementation Challenges



- Successive refinement involves incremental specification
- Bottom up implementation complicates the flow
  - RTL Subtree implemented as Hard macros and used in soft macro context
  - Hard macro integration
  - UPF needs to be adjusted as a result of subtree hardening
- New Challenges
  - Subtree must be configured before hardening (to drive implementation)
  - Effective Power state Modelling
  - Isolation of UPF created power controls
  - Power supply considerations for retention and isolation strategies

Hard Macro Integration Challenge

- Traditional Hard Macros , exemplified by memory
  - Typically supplied as HDL behavioural model
    - May be non-Power Aware (PA), partially PA or fully PA
  - No UPF used for implementation of macro
  - Liberty defines some of its implemented power architecture
    - Interface characteristics : related_supply on logic ports, pg_pins, etc
    - Missing internal power states definition for macro with embedded switch
- Need a generic integration solution for a generic memory models
  - Create a UPF Power model for the Hard macro
  - Reuse in different contexts

Power Model Creation

```
proc ram_power_model {pd_name mem_instance pg_en {ret_en "no_ret"}} {

 create_power_domain  $pd_name -elements $mem_instance
 create_supply_net   my_vdd_$pd_name  . . .  # internal switched supply net
 create_supply_set $pd_name.primary -update -function "power  my_vdd_$pd_name" . . .
 . . .
# optional retention support
 if  {$ret_en =="no_ret"}  then  {
  add_power_state $pd_name.primary -update
    -state  ON   " -logic_expr {$pg_en  == 0 } -supply_expr  { . . . }
} else {
   add_power_state  $pd_name.primary -update
    -state ON  "-logic_expr {$pg_en == 0  && ret_en  == 1} -supply_expr { . . . }
}
# Define power states ON, OFF and optional RET of power domain in terms of supply_set
power states
 . . .
create_power_switch sw_internal … # for internal switched supply net
. . .
# Define  related supplies on ports of Memories . Can override liberty
set_port_attributes -ports $mem_instance/$ports
  -related_power_port $pd_name.primary.power -related_ground_port $pd_name.primary.ground
      -exclude_ports "$mem_instance/PGEN $mem_instance/RET_EN"
}
```

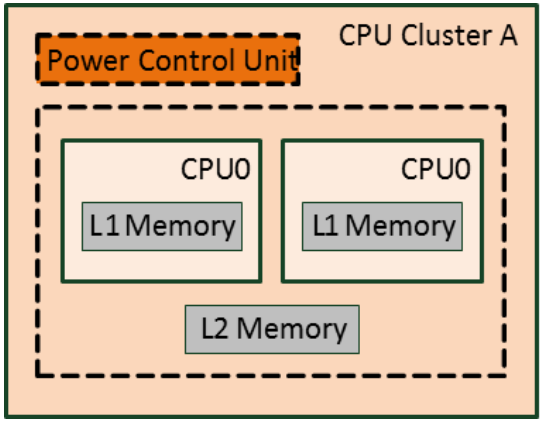
Tcl procedure containing Power Model made up of UPF commands for Memories

Power model of Hard Macro Integration

```
Cpu Cluster A Configuration.upf :

#Integrate Power model for L2 Memory
ram_power_model  PDL2MEM $L2MEM_instance PWR_CLUS_A
RET_CLUS_A

# update CPU Cluster A power state dependency in terms of
states of PDL2MEM
 add_power_state PD_CLUS_A -update \
  -state ON      { -logic_expr  {PDL2MEM == ON}} \
  -state MEM_OFF { -logic_expr  {PDL2MEM == OFF}} \
  -state RET     { -logic_expr  {PDL2MEM == RET}} \
  -state OFF     { -logic_expr  {PDL2MEM == OFF}}
```



- L2 Mem Power model called in Cluster A configuration UPF
- CPU Cluster A power states updated with dependencies on L2 Memory states
- Configuration UPF compatible with Implementation and Verification

Soft Macro Hardening Process Considerations

- RTL Subtree, carved out for implementation
  - Requires Self-contained UPF (constraints, configuration and implementation UPF)
  - Hardened Soft Macro
- Hardening process involves modelling external context of the macro based on available supplies in Macro
  - The rest of the SoC also needs context information of the carved out Hardened Soft Macro for their own implementation
- Verification done at full SoC context – flat view
  - Potential for Verification and implementation views to differ

Soft Macro Hardening Solution

- Align RTL-Sub tree and power domains in preparation for implementation
  - Self-contained constraint/config/implementation UPF for each RTL Sub-tree
- Three interface scenarios to handle
  - Implementation of Soft Macro
  - Implementation of the higher level (hierarchical) context
  - Verification / non-hierarchical context
- Model external Interface context in implementation.upf

Implementation.upf:

```
if {$env(CORE_UPF) == 1 && $env (TOP_UPF) == 1 } then
 set_port_attributes –ports $intf_ports … -driver_supply ss_set1
} elseif
 set_port_attributes –ports $intf_ports … -receiver_supply ss_set1
```

Soft Macro Hardening Solution II

Environment variables used to select appropriate condition

Constraints.upf :

```
if { $env(CORE_UPF) == 1 } then {
    set env(CORE_UPF) $env(FLAT_DESIGN)
    set regTopValue   $env(TOP_UPF)
    set env(TOP_UPF)  0
    puts "\nINFO: Loading UPF for CPU"
    load_upf cpu.upf -scope u_cpu0
    set env(CORE_UPF) 1
    set env(TOP_UPF)  $regTopValue
  }
```

Propagate design topology setting across nested load_upf calls

Lower level UPF loaded for implementation of soft macro or for flat_view verification

Power States Challenges

- Power state definition for power domains and supply sets can use logic_expr and supply_expr (for supply sets only)
  - No restrictions on the expressions
  - Complexity of expression, unintended state overlap
- Power states can be updated with unexpected side effects
  - Update semantics not clearly defined
  - Potential for multiple update failure when creating dependencies
- Can potentially define technology detail ie supply_expr in constraints / configuration UPF
  - Breaks separation of Logical view and Technology specific view of Successive Refinement

Recommendation for Power State specification and refinement

Separate configuration and implementation concerns

Configuration UPF:

```
add_power_state PDA.primary  \
   -state ON { -logic_expr { sw_ctrl == 1}}
```

Supply_set power state specified in terms of power control signals in logic_expr

Implementation UPF:

```
 add_power_state PDA.primary -update \
   -state ON { -supply_expr {FULL_ON 1.0}}
```

supply_set power state updated with supply_expr

Avoid redundancy and ensure clean composition

Configuration UPF:

```
add_power_state PDA \
  -state ON { -logic_expr {PDA.primary == ON}}
```

Power domain states specified in terms of states of its supply sets and states of lower level power domains

Other Challenges and resolution approach

- Isolation of UPF Created power control signal
  - Needed for implementation of larger context of hardened soft macro
  - Typically on the lower boundary of the power domains of the larger context
- UPF 2.1 semantics inconsistent and limited tool support
  - Command precedence and processing of set_port_attributes vs create_logic_port
  - Static checking limited when checking for level_shifter and isolation requirements of UPF created power control signals
- User defined supply sets for Isolation and retention strategies
  - DEFAULT_ISOLATION and DEFAULT_RETENTION were not used
  - Better control over availability of supplies

Observations and Results

- Standards based issues were fed back to p1801 working group for clarification
  - Most are addressed in IEEE p1801-2015 UPF 3.0
- Achieved reasonable multi-vendor tool flow with the UPF subset that we ended using
- Power Aware Coverage was sign-off criteria
  - Initial verification leveraged static checking to ensure sound power architecture earlier in the process
  - Coverage of power states, power state dependencies and power state transitions
  - Tool generated power state coverage augmented with
    - User defined System Level power state coverage

Results

- Clean static check report of constraints/configuration/implementation UPF
  - Applied waivers to static checks that did not make sense in our design context
  - Some tool issues with False negatives
- Areas of improvement :
  - Tools: Language support for UPF 2.1 and interoperability among tools
  - Language : Continued Improvements to UPF LRM

Questions