UPF GENERIC REFERENCES:
UNLEASHING THE FULL POTENTIAL

Jitesh Bansal (Mentor Graphics)
Durgesh Prasad (Mentor Graphics)
Agenda

• Introduction
• Power Aware Verification using Unified Power Format (UPF)
• UPF Generic References
• Usage
  – Isolation Specification and Verification
  – Retention Specification and Verification
• UPF Extensions required
• Conclusion
Introduction

Today's SoCs

- Are incredibly Complex
- Have sophisticated power management strategies for highly power efficient design
- Integrate variety of implementation cells like isolation and retention

They Must

- Verify the power management
  - early in the design flow
Unified Power Format (UPF)

- RTL is augmented with a UPF specification
  - To define the power architecture for a given implementation

- RTL + UPF drives implementation tools
  - Synthesis, place & route, etc.

- RTL + UPF also drives power-aware verification
  - Ensures that verification matches implementation
PA Verification with UPF

• Different Systems have different power management
• Power Gating
  • Isolation
  • Retention
• Multi-Voltage
  • Level Shifting
• Body Bias and DVFS
• UPF provides commands to
  • express the power management strategies e.g. set_isolation
  • verify the power architecture e.g. bind_checker
**UPF Generic References**

- UPF generic references were first introduced in UPF 2.0 (IEEE_1801_2009).
- The generic references provide reserved keywords that cannot be redefined and would accept values depending on the context of their use.
- Types of UPF Generic References –
  - **UPF_GENERIC CLOCK** - Refers to the clock of a sequential logic.
  - **UPF_GENERIC_DATA** - Refers to the data pin of a sequential logic or isolation input pin.
  - **UPF_GENERIC_ASYNC_LOAD** - Refers to the asynchronous pins of a sequential flop.
  - **UPF_GENERIC_OUTPUT** - Refers to the output of a sequential logic or isolation cell.
always@(posedge reset, posedge clk)
begin
    if(reset)
        q <= 1’b0;
    else
        q <= d;
end

<table>
<thead>
<tr>
<th>UPF Generic References</th>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPF_GENERIC_CLOCK</td>
<td>clk</td>
</tr>
<tr>
<td>UPF_GENERIC_DATA</td>
<td>d</td>
</tr>
<tr>
<td>UPF_GENERIC_ASYNC_LOAD</td>
<td>reset</td>
</tr>
<tr>
<td>UPF_GENERIC_OUTPUT</td>
<td>q</td>
</tr>
</tbody>
</table>
UPF Generic References

UPF Generic References in Isolation Cell Insertion

<table>
<thead>
<tr>
<th>UPF Generic References</th>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPF_GENERIC_DATA</td>
<td>O1 (‘actual’ for the ‘formal’ port ‘IN1’)</td>
</tr>
<tr>
<td>UPF_GENERIC_OUTPUT</td>
<td>IN1</td>
</tr>
<tr>
<td>UPF_GENERIC_CLOCK/UPF_GENERIC_ASYNC_LOAD</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>
Isolation Specification

- map_isolation_cell/use_interface_cell
  - Used for custom isolation cell

module SFX_ISO(input ISO, input I, output Z);

set_isolation ISO -domain PD -elements {IN1 IN2 IN3} \
-isolation_signal {iso_en}
Isolation Specification

- **Benefits**
  - No need to find out the *actual*
  - multiple *map* commands not required.
  - No worry about *effective* elements list

```plaintext
map_isolation_cell ISO \
    -domain PD -elements {IN1}\ 
    -lib_model_name {SFX_ISO}\ 
        -ports "ISO io_en" \ 
        -ports "I O1" \ 
        -ports "Z IN1"
map_isolation_cell ISO \
...
```

```plaintext
map_isolation_cell ISO \
    -domain PD \
    -lib_model_name {SFX_ISO} \ 
    -ports "ISO iso_en" \ 
    -ports "I UPF GENERIC DATA" \ 
    -ports "Z UPF GENERIC OUTPUT"
```
Isolation Verification

- ISO cell acts like buffer in non-isolation period
  - Use bind_checker

checker module interface

iso_checker(input iso_in, output iso_out, input iso_en);

UPF bind checker command (No generic references)

bind_checker IN1_iso_checker_inst -module iso_checker \ 
-ports {{iso_in  O1} {iso_out IN1} {iso_en  iso_en}}

bind_checker IN2_iso_checker_inst -module iso_checker \ 
-ports {{iso_in  O2} {iso_out IN2} {iso_en  iso_en}}
Isolation Verification

Users need not to worry about specifying input/output of isolation cell.
RetentionPolicy

UPF provides set_retention command with following options to customize retention behavior

• **elements** — To select a particular set of sequential elements.
• **save_signal/-restore_signal** — Logic signals to save and restore the register value.
• **save_condition/-restore_condition/-retention_condition** — Boolean conditions that should be true for save/restore/retention operations.
Retention Specification

• Basic Retention Strategy
  – Retain all the sequential elements of power domain
  – Save/Restore signals are only used to control

```bash
set_retention RET -domain PD -save_signal {SAVE posedge} \
-restore_signal {RESTORE posedge}
```

• Advanced Retention Strategy
  – save_condition/restore_condition/retention_condition are used along with save/restore signals to control retention
  – Save/Restore event depends on clock/async_load of sequential element

```bash
set_retention RET -domain PD -save_signal {SAVE posedge} \
-restore_signal {RESTORE posedge} -restore_condition {!CLOCK}
```
Retention Specification

Controlling Retention behavior according to clock/async_load is not easy and poses following problems

• Clock/Async load signals vary with each sequential element
  – Difficult to determine clock/async_load for each sequential element

• Need to write multiple retention strategies
  – One strategy for each unique clock/async_load signal
  – Numerous retention strategies will make the UPF file huge and error-prone

• Custom assertions for retention cells, if written by users, are numerous and tough to maintain.

• Design is not scalable
  – Modification in RTL require changes in retention strategies
### Retention Specification

<table>
<thead>
<tr>
<th>Sequential Elements</th>
<th>Retention Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>always @(posedge CLOCK1, posedge RESET) begin if(RESET) Q1 &lt;= 1'b0; else Q1 &lt;= D; end</td>
<td>set_retention RET1 -domain PD -save_signal {SAVE posedge} -restore_signal {RESTORE posedge} -restore_condition {!CLOCK1} -elements {Q1}</td>
</tr>
<tr>
<td>always @(posedge CLOCK2, posedge RESET) begin if(RESET) Q2 &lt;= 1'b0; else Q2 &lt;= D; end</td>
<td>set_retention RET2 -domain PD -save_signal {SAVE posedge} -restore_signal {RESTORE posedge} -restore_condition {!CLOCK2} -elements {Q2}</td>
</tr>
</tbody>
</table>
Retention Specification

Use of UPF generic references – UPF_GENERIC_CLOCK and UPF_GENERIC_ASYNC_LOAD

```bash
set_retention RET -domain PD -save_signal {SAVE posedge} \  
  -restore_signal {RESTORE posedge} \  
  -restore_condition {!UPF_GENERIC_CLOCK}
```

- Only one set_retention strategy is required to cover all different types of clock/async_load signals.
- EDA tools parse these UPF generics in restore_condition and replace it with the specific clock of each sequential element for application of retention strategy.
- Automated, robust, and easier flow for designers.
- Scalable design, addition or deletion of sequential elements in a design does not require any change in the UPF.
### Sequential Elements

```verilog
define always @(posedge CLOCK1, posedge RESET) begin
  if(RESET)
    Q1 <= 1'b0;
  else
    Q1 <= D;
end

always @(posedge CLOCK2, posedge RESET) begin
  if(RESET)
    Q2 <= 1'b0;
  else
    Q2 <= D;
end
```

### UPF generic Retention Strategy

```verilog
set_retention RET
  -domain PD
  -save_signal {SAVE posedge}
  -restore_signal {RESTORE posedge}
  -restore_condition {!UPF_GENERIC_CLOCK}
```
Retention Verification

• UPF `bind checker` command is used to verify the retention behavior of design

• Assertions are written in a checker module and bind to the specific instances of design using `bind_checker` command

Checker module interface -

```verilog
module ret_checker(input restore_condition,
                   input ret_ff, input restore_signal);

bind_checker Q1_ret_checker -module ret_checker \\
    -bind_to inst1 -ports {{restore_condition !CLOCK1} \\
    {ret_ff Q1} {restore_signal RESTORE} }

bind_checker Q2_ret_checker -module ret_checker \\
    -bind_to inst1 -ports {{restore_condition !CLOCK2} \\
    {ret_ff Q2} {restore_signal RESTORE} }
```
Retention Verification

• Multiple bind_checker statements are required for different clock/async_load signals

• With UPF generic references, only one statement is required

```plaintext
array set RET_STRTGY [query_retention RET -domain PD -detailed]
set ELEMENTS $RET_STRTGY(elements)
set RESTORE_COND $RET_STRTGY(restore_condition)

bind_checker checker_inst -module checker -elements $ELEMENTS
    -ports {
        {restore_condition $RESTORE_COND}
        {ret_ff UPF GENERIC OUTPUT}{restore_signal RESTORE}
    }
```

- UPF GENERIC OUTPUT is retained element {Q1 Q2}
- RESTORE_COND is !UPF GENERIC CLOCK.
- Command is applied to each element of the bind_checker command and UPF GENERIC CLOCK in the restore condition is replaced by the corresponding CLOCK of each sequential element.
Sequential Element Verification

- UPF generic references can be used for custom verification of each sequential element with the help of the bind_checker command.
- Example: User wants to verify that clock should be off when async load is active.

Checker Module -

```verilog
module checker (input clk, input async_load, input seq_elem);
    always @(posedge async_load)
    begin
        assert ( !clk ) else $display ("Error : CLK is not OFF,Element value is '%b'",seq_elem);
    end
endmodule
```
Sequential Element Verification

array set RET_STRTGY [query_retention RET -domain PD -detailed]
set ELEMENTS $RET_STRTGY(elements)

bind_checker checker_inst -module checker -elements $ELEMENTS
   -ports { {clk UPF_GENERIC_CLOCK} \
      { async_load UPF_GENERIC_ASYNC_LOAD} \
      { seq_elem UPF_GENERIC_OUTPUT } } 

The above command would bind checker instance for each sequential element and the UPF generic references would be replaced by corresponding Clock/Async_load/Output in port list.
• Enhancements are required in UPF LRM to allow usage of UPF Generic References in various commands
• set_retention
  – Accept UPF generic references and their simple expressions in save_condition/restore_condition/retention_condition

```
set_retention RET -domain PD \
-retention_condition {!UPF_GENERIC_CLOCK}
```

• query_* commands
  – Allow query_retention to return various condition in terms of UPF generic references to be used in bind_checker.
UPF Extensions

• bind_checker
  – Allow -ports option to accept generic references as well as their simple expressions for port mapping

```
bind_checker checker_inst -module checker -elements $ELEMENTS
  -ports { {clock_signal UPF GENERIC CLOCK}}
```

  – -ports option also accepts the UPF generic references returned by query_commands in its port mapping

```
set_retention RET -domain PD -save_condition
{UPF GENERIC CLOCK && save}
bind_checker checker_inst -module checker -elements $ELEMENTS
  -ports { {save_cond $ret_array(save_condition)}} ...
```
UPF Extensions

• bind_checker
  – Allow -elements option to accept the list of signals. Also, the checker instance name needs to be modified for different signals in its -elements list.

```bash
bind_checker checker_inst -module checker \ 
-elements {inst1/q1 inst1/q2} ...
```

– In the above case, ‘q1’ and ‘q2’ are signals within the instance ‘inst1’. The expected behavior of the above command should be to bind two instances, namely ‘checker_inst_q1’ and ‘checker_inst_q2’ in the scope ‘inst1’.
Extensions in EDA Tools

• EDA tools should be able to process UPF generic references in their context

```bash
set_retention RET -domain PD \
-save_condition {UPF_GENERIC_ASYNC_LOAD && !UPF_GENERIC_CLOCK}
```

- EDA tools should replace save_condition for each sequential element with their respective clocks/async_loads.

```bash
set_retention RET -domain PD -elements {q1 q2} ... \
bind_checker checker_inst -module checker -elements $ELEMENTS\ 
-ports { {clock_signal UPF_GENERIC_CLOCK} ... 
```

- Tools should be able to extract the respective clocks of ‘q1’ and ‘q2’ signals and bind the ports accordingly
Conclusion

• UPF Generic references are easy to use and provide automation
• Facilitate writing of a concise, scalable, and less error-prone UPF
• Very helpful in specifying custom cells and verifying various power aware cells
  – requires few UPF extensions
Thank You