Best Practice Coding Assertion IP (AIP) to Get More Predictable Results

SJ Wu, Leon Yin
Synopsys
Overview on Assertion IP (AIP) of Interface

• Interface Types:
  • Standard interface
  • In-house defined interface

• Reusability and Extensibility of Interface
  • AIP is worthwhile investment
  • AIP used in both simulation and formal
    • Without proper principles to follow, the created AIP will not be suitable for both simulation and formal at the same time

How to implement such reusable AIPs is essential for ensuring comprehensive checkers that can be reused everywhere.
Three phases of AIP development

1. Specification review phase
   1. What should become an assertion checkers?
   2. How to divide-and-conquer multi-channel interfaces so that a predictable schedule can be created.

2. Coding phase
   1. Several coding guidelines are listed
   2. Some formal skills to reduce the complexity in formal verification

3. Validation phase
   1. Fault analysis flow is deployed to reach a strong confidence
PHASE-I: SPECIFICATION REVIEW

A. What a standard interface should have

1) Channels

2) Mandatory components:
   1) optional and mandatory signals should be clearly documented.
   2) The assertions created for optional signals should also leave a parameter for user to switch on and off.

3) Cross channels interaction

4) Exceptions: undocumented failure valuable products from formal verification

<table>
<thead>
<tr>
<th>Specification Review Item</th>
<th>CH-A group</th>
<th>CH-B group</th>
<th>High Level Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>AR/R</td>
<td>AW/W/B</td>
<td>CH-A group and CH-B group can deploy two team to develop</td>
</tr>
<tr>
<td>2)</td>
<td>AR/AW</td>
<td></td>
<td>AR and AW channel support optional burst size with default value equal to data width</td>
</tr>
<tr>
<td>3)</td>
<td>AR</td>
<td>R</td>
<td>Read must be an active id from AR command.</td>
</tr>
<tr>
<td>3)</td>
<td>AW</td>
<td>W</td>
<td>Wdata length must be the same as the corresponding AWlen</td>
</tr>
<tr>
<td>4)</td>
<td>AR/AW</td>
<td></td>
<td>Exceptional: What happens when address run over the maximum boundary?</td>
</tr>
</tbody>
</table>

Table I. AMBA AXI4 AIP implementation specification
PHASE-I: SPECIFICATION REVIEW

B. Identify the targets to divide and conquer with Table I

1) Create the packet:
   1) Packet attributes → Struct
   2) Create common data structure:
   3) Function:
      Quick example:
      Calculating the Wdata strobe according to data beat and address for AXI4 write transaction

4) Channels:
PHASE-II: CODING/IMPLEMENTATION

A. Define reusable enum and structure:
B. Define important indicators
C. Parameterization
D. Coding for different roles
E. Coding for both simulation and formal
F. Avoid unconscious over-constraints
G. Coding optimization for formal
   1. Case splitting on AIP:
   2. Symbolic abstraction on AIP:
PHASE-II: CODING/IMPLEMENTATION

A. Define reusable enum and structure:

• Instead of connecting a specific signal, the sample struct is used to pack signals
• Below example shows the struct of AR channel for AXI4. This is an extensible approach to any channels.

```c
typedef packed struct {
    logic [ADDR_WIDTH-1:0] address;
    logic [BURST_SIZE_WIDTH-1:0] burst;
    logic [BURST_LEN_WIDTH-1:0] length;
    logic [ID_WIDTH-1:0] id;
} AR_object;
```
PHASE-II: CODING/IMPLEMENTATION

B. Define important indicators

1. Serve as the key points of debugging.
2. These events are served as the helpful indicators to identify the problem.

Example: Important indicators on AXI4 AIP

<table>
<thead>
<tr>
<th>Events</th>
<th>Implementation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Handshake</strong></td>
<td>wire AR_handshaked=ARvalid &amp;&amp; ARready;</td>
<td>Handshake completion.</td>
</tr>
<tr>
<td></td>
<td>/*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. QUEUE_DEPTH is the depth value of command queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. queue.content[i] is the i-th object that is queued inside queue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. queue.active[i] indicates i-th entry of queue is active.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*/</td>
<td></td>
</tr>
<tr>
<td><strong>Active objects in queue</strong></td>
<td>AR_object obj_in_cmd_queue[QUEUE_DEPTH]; always_comb begin</td>
<td>To show all active transaction in queue.</td>
</tr>
<tr>
<td></td>
<td>for(int i=0;i&lt;QUEUE_DEPTH;i++)begin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>obj_in_cmd_queue[i] = queue.active[i] ? queue.content[i] : 0;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
C. Parameterization

1. Parameterization commonly comes to different configuration or specification requirements

2. AIP can also design a parameter to switch on/off an incomplete function which proactively prevents the problems from hiding with hard codes during the early development
PHASE-II: CODING/IMPLEMENTATION

D. Coding for Role changing

1. Four usage models in AIPs as shown

2. The role changing between assumptions and assertions.

3. Leverage the benefit of this coding style to assert or assume the property according to the required role.
Although symbolic abstraction is a powerful approach in formal verification, we still must provide the general version of a checker.

```verilog
property symb_id_stable;
 @(posedge clk) disable iff(!rstn) ##1 Stable(symb_id);
endproperty

property rid_is_legal; //general version of rid is valid, check rid_match_an_active_read_cmd whenever Rdata comes.
 @(posedge clk) disable iff(!rstn) Rvalid |-> rid_match_an_active_read_cmd==1;
endproperty

property rid_is_legal_formal_check; //symbolic abstraction to check rid is valid only when it matches a symbol id.
 @(posedge clk) disable iff(!rstn) (symb_in && !symb_out && Rvalid && Rid==symb_id)
 |-> rid_match_an_active_read_cmd==1;
endproperty
```

```verilog
generate
if (AIP_in_sim == TRUE) begin
 //Simulation testbench has no concept of selecting a symbol id value as a generalized variable to check.
 //So rid_is_legal_formal_check cannot be used here.
 assert property(rid_is_legal);
end else begin
```

F. Avoid unconscious over-constraints

- Over-constraints limit the state space of a testbench.
- It implies bug can lurk inside.

Reviewing the table as a proper way to avoid unintentional constraints on AIP

<table>
<thead>
<tr>
<th>Roles</th>
<th>Assume</th>
<th>Assert</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTER</td>
<td>P1</td>
<td>P3</td>
</tr>
<tr>
<td>SLAVE</td>
<td>P3</td>
<td>P1</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>P1, P3</td>
<td>NA</td>
</tr>
<tr>
<td>CHECKER</td>
<td>NA</td>
<td>P1, P3</td>
</tr>
</tbody>
</table>

Properties: P1; P2; P3;
PHASE-II: CODING/IMPLEMENTATION

G. Coding optimization for **formal**

1. Case splitting on AIP: by enumerating all the cases in multiple assertions, convergence becomes easier to get.

<table>
<thead>
<tr>
<th>Index</th>
<th>Without Case Splitting</th>
<th>With Case Splitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof time</td>
<td>One write data strobe check</td>
<td>8 splitted write data strobe checks</td>
</tr>
<tr>
<td></td>
<td>Inconclusive after 24hr</td>
<td>All proven in 1hr.</td>
</tr>
</tbody>
</table>

2. Symbolic abstraction on AIP: symbolic abstraction uses one symbol value to analyze if there is any possible violated case exists.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Helpful for What</th>
</tr>
</thead>
<tbody>
<tr>
<td>rid/bid</td>
<td>Check if this id is valid</td>
</tr>
<tr>
<td>wdata beat</td>
<td>Check write strobe of one arbitrary beat</td>
</tr>
</tbody>
</table>

some candidate symbols applicable for AXI4
PHASE-III VALIDATION

- 2 testbenches to validate AIP:
  - back-to-back
  - leveraging the existed high-quality AIP
CONCLUSION

1. Three systematical phases for building the AIP from scratch:
   1. Phase-I helped to schedule out development milestones
   2. Phase-II listed out the coding rules for both simulation and formal
   3. Phase-III proposed a approach to boost the AIP quality with the fault analysis flow

2. This development flow is already rolled out successfully in Synopsys IP team
   1. It makes AIPs development time move faster
   2. 16 issues were found on a newly developed internal interface AIP without impacting IP team AIP users
   3. the iterations between AIP developers and users are greatly reduced
Thanks!