Assertion IP for H2A protocol

1. Command Channel

1. rdy /vld flow control protocol	
Assertion Name	Description
Command_channel_assert_vld_until_rdy	vld should keep asserted, if rdy is not asserted. Vld should keep assert till ready.
Command_channel_rdy	Rdy should assert eventually
Command_channel_stable_cmd_operand	When valid is asserted and data should be stable
Command_channel_assert_rdy	Rdy should keep asserted before the stable data.
Command_channel_assert_vld_rdy_unknown	Vld and rdy should not be assert unknow(x/z).
2. CMD Sequence Checks	
Command_assert_RST_INIT	RST should assert before INIT.
Command_assert_RST_INIT_HLT	RST,INIT should assert before HLT.
Command_assert_RST_INIT_HLT_OP	RST,INIT,HLT should assert before OP cmd
Command_assert_no_xz_operation	After RST is asserted, INIT and HLT should not be unknown(x/z)

3. Response Channel

Assertion Name	Description
Response_channel_done	When command is not completed, done pulse should not assert.
Response_done_cmd	done will assert when command finishes.
Response_done_in_order_check	Done should be in order.
Response_done_not_unknown	Done should not assert unknown (x/z) value when not in reset.

M2. Write SVA Code

Read the specifications for H2A protocol Download H2A protocol and based on the assertion plan created in Milestone M1, write the SVA code to verify the H2A protocol.

. 1. rdy /vld flow control protocol

1.
```
property P1 ;
  (@(posedge clk) h2a_valid && !a2h_ready |=> h2a_valid );
endproperty
Command_channel_assert_vld_until_rdy:   assert property (P1);
```

2.
```
property P2 ;
  (@(posedge clk) h2a_valid  |=> strong(##[0:$] a2h_ready));
endproperty
Command_channel_rdy:   assert property (P2);
```

3.
```
property P3 ;
(@(posedge clk) h2a_valid && ! a2h_ready |=> $stable(h2a_cmd) );
endproperty
Command_channel_stable_cmd_operand:   assert property (P3);
```

4.
```
property P4 ;
(@(posedge clk) a2h_ready |=> $stable(h2a_cmd) );
endproperty
Command_channel_assert_rdy:   assert property (P4);
```

5.
```
property P5 ;
(@(posedge clk) h2a_valid && ! a2h_ready |-> ##1 (not($isunknown(h2a_valid)) &&
not($isunknown(a2h_ready)) );
endproperty
Command_channel_assert_vld_rdy_unknown:   assert property (P5);
```

CMD Sequence Checks

```
sequence s_h2a_hs;
  a2h ready && h2a valid [ ->1];
endsequence

sequence cmd_is_rst;
  (h2a_cmd == rst)
endsequence

sequence cmd_is_rst;
 cmd_is_init
endsequence

sequence cmd_is_op;
 (h2a_cmd == 0p1) || (h2a_cmd == 0p2) || (h2a_cmd == 0p3) ||
(h2a_cmd == Op4) || (h2a_cmd == Op5); ire cmd_is_hlt = (h2a_cmd == Hlt)
endsequence
```

6. property P6 ;

```
(@(posedge clk) $fell(rst) |->  ##0 cmd_is_init);
endproperty

Command_assert_RST_INIT:   assume property (P6);
```

7. property P7 ;

```
(@(posedge clk) disable iff (rst)  ! cmd_is_init |-> ! h2a_valid || ! cmd_is_hlt);

endproperty

Command_assert_RST_INIT_HLT:   assume property (P7);
```

8. property P8 ;

```
(@(posedge clk) disable iff (rst)  ! cmd_is_init |-> (!h2a_valid || ! cmd_is_hlt ||
cmd_is_op);


endproperty

Command_assert_RST_INIT_HLT_OP:   assume property (P8);
```

9. property P9;
```
(@(posedge clk) disable iff (rst)    |-> ##1 ($unknown(cmd_is_init)) && ($isunknown(cmd_is_hlt)) );

endproperty

Command_assert_no_xz_operation:   assume property (P9);
```

10. property P10;
```
(@(posedge clk) rst  |-> ##1 (not($unknown(a2h_done))  );

endproperty

Response_channel_done:   assume property (P10);
```

11. sequence S1;
```
@ (posedge clk) a2h_done |-> not(a2h_done) throughout  (sym_cmd);

endsequence

Response_channel_done:   assume property (S1);
```

12. property P12;
```
(@(posedge clk) $rose(sym_cmd ) |-> ##1 a2h_done) ;

endproperty

Response_done_cmd:   assume property (P12);
```

13. property P13;
```
(@(posedge clk) disable iff (rst)  |-> $unknown(a2h_done));


endproperty

Response_done_cmd:   assume property (P13);
```