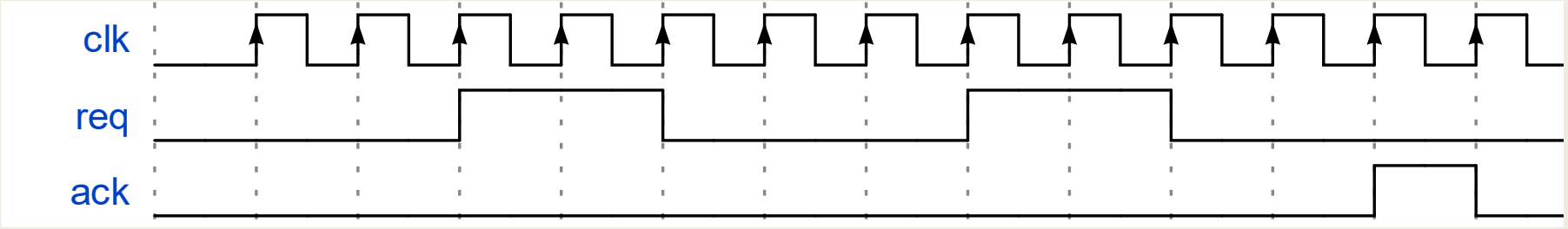
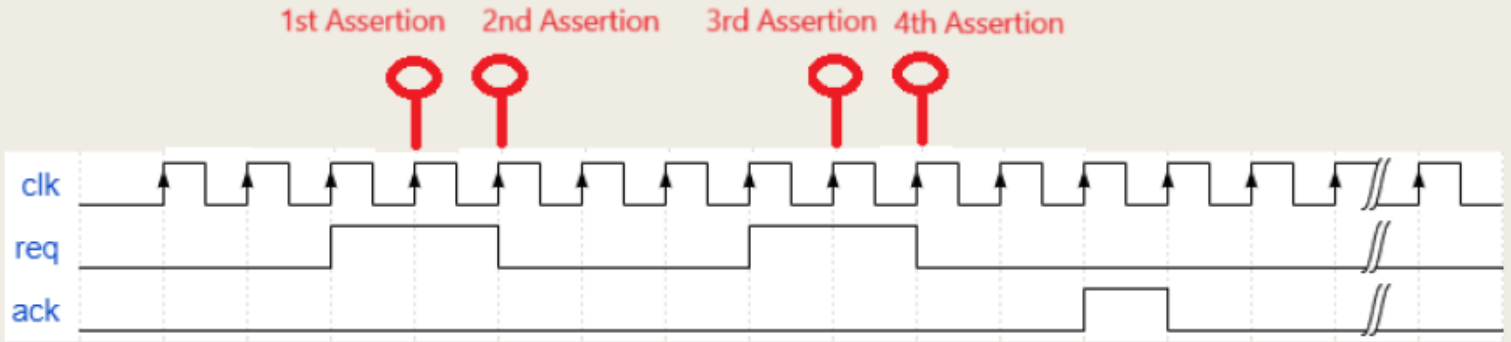
Waveform and Specs Overview



1. *Requests and acknowledgments are valid at the rising edge of the clock.*
2. *An acknowledgment must always follow a request.*
3. *No acknowledgment should occur without a preceding request.*
4. *The acknowledgment for a request must occur within 3 to 16 clock cycles after the request is made.*
5. *Each request must be followed by an acknowledgment.*
6. *An acknowledgment is not required to respond to the current request before the next request occurs.*
7. *A new request cannot be acknowledged in the same cycle it is raised.*
8. *An acknowledgment for a previous request may occur in the same cycle as a new request is raised.*

SVA Property

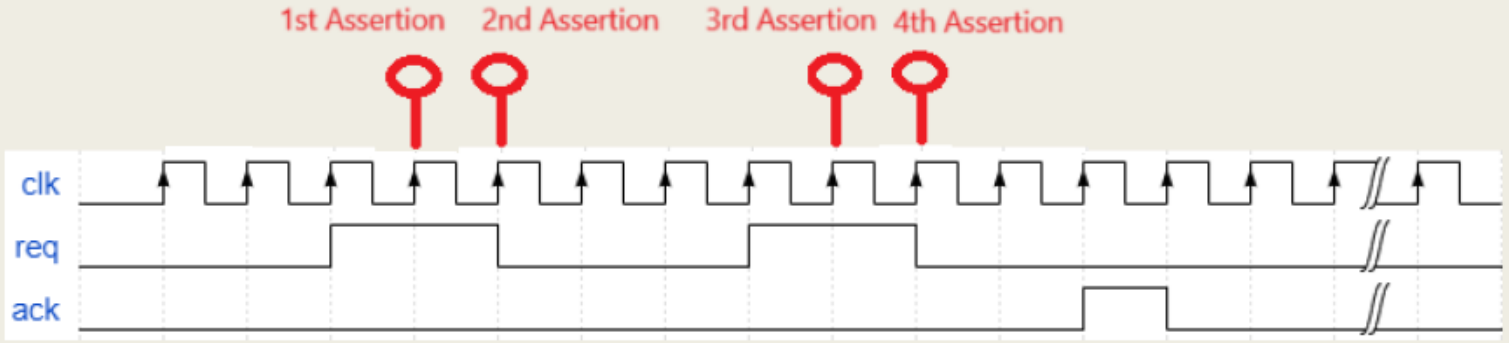
```
always clk = #5 ~ clk;

default clocking my_clk @(posedge clk); endclocking


SVA_Porp_Alone: assert property (req |-> ##[3:16] ack)
$display ("\033[32m SVA assertion passed @%0t", $time);
else
 $display ("\033[31m SVA assertion failed @%0t", $time);
```



SVA Property...

Log File

- `assertion passed @105`
- `assertion passed @105`
- `assertion passed @105`
- `assertion passed @105`



From the waveform and SVA_Prop_Alone, we see that there are 4 assertions corresponding to the 4 requests, but only one acknowledgment (ack). Despite this, all 4 assertions have passed, which does not align with the specifications.

Combination of Auxiliary Code and SVA

```
// Auxiliary code

bit [9:0] req_cnt;
bit [9:0] ack_cnt;
always_ff @(posedge a_clk)
  begin
        if(req)
        req_cnt <= req_cnt +1;
        if(ack)
        ack_cnt <= ack_cnt +1;
  end

// SVA property

 property handsh;
 bit [9:0] req_local_count;
(req, req_local_count = req_cnt) |->
 strong(##[2:8](ack && ack_cnt == req_local_count));
 endproperty

// if you want to check ack for the end of the simulation just
replace ##[2:8]with[1:$]
```

Combination of Auxiliary Code and SVA…

```
// SVA property

 property handsh;
 bit [9:0] req_local_count;
(req, req_local_count = req_cnt) |->
 strong(##[2:8](ack && ack_cnt == req_local_count));
 endproperty
```

- Use the strong keyword to override the default weak assertion. This change will enable you to determine whether the assertion passes or fails at the end of the simulation, based on whether the number of clocks (ranging from 3 to 16) occurred during the simulation time or not.
- each request triggers an assertion,
- when it triggers it saves the current value of request count in its local variable "req_local_count"
- 4 assertions will trigger since we drove 4 requests,
  - *the first assertion has its local var "req_local_count" = 1*
  - *the second assertion has its local var "req_local_count" = 2*
  - *the third assertion has its local var "req_local_count" = 3*
  - *the fourth assertion has its local var "req_local_count" = 4*
- The Assertions which will pass are those who had ack_cnt == req_count

Combination of Auxiliary Code and SVA…

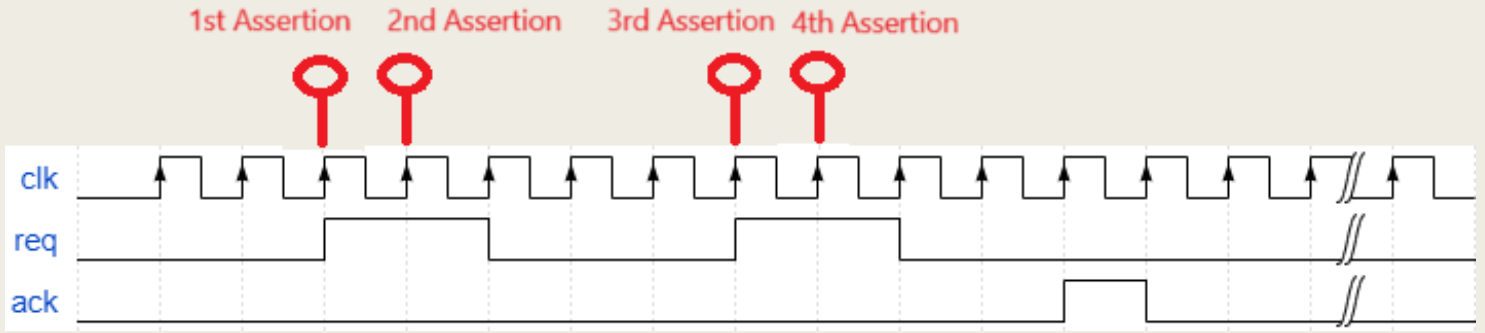
```
// SVA property

property no_ack_without_req;
!$stable(ack_cnt) |-> !(ack_cnt > req_cnt); // !stable = change
endproperty
```

Whenever ack counter change it will check if the ack counter > request counter or not.

```
req_ack : assert property(handsh)
  $display ("\033[32m with aux code assertion passed @%0t", $time);
else
  $display ("\033[31m with aux code assertion failed @%0t", $time);

 ack_without_req : assert property(no_ack_without_req)
 $display ("\033[32m no_ack_without_req assertion passed @%0t", $time);
else
 $display ("\033[31m no_ack_without_req assertion failed @%0t", $time);
```

Transcript

Log File



- `# with aux code assertion passed @105`
- `# no_ack_without_req assertion passed @115`
- `# with aux code assertion failed @155`
- `# with aux code assertion failed @155`
- `# with aux code assertion failed @155`

Based on the log file results, we currently have 4 requests and only 1 acknowledgment. This means that only 1 assertion passed, while 3 failed, which is accurate. Additionally, the number of acknowledgments is less than or equal to the number of requests, which is also correct.

Note

```
Property Assertion_Note;
@(posedge clk)
enable_of_Assertion |=> !Signal;
endproperty
```

the assertion works every posedge of the clock , if enable of assertion is not high , **assertion will pass but it's fake pass which not reported in the log file.**

whenever the enable of assertion is high , the assertion will decided if it's fail or pass based on the signal in the next cycle and it will be reported to the log file.