Case study with PSL

1. Introduction 1
2. Property Specification Language 3
3. Acknowledge after 4 clock cycles 6
4. Acknowledge a request before applying a new request. 7
5. What about the reset? 8

1. Introduction

With PSL (Property Specification Language) it is possible to perform assertion based verification of the design: during simulation properties are checked.
This document gives an example how it can be used. As a case we use the system *double*. The entity description is:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY double IS
  PORT (data  : IN  positive;
        req   : IN  std_logic;
        q     : OUT natural;
        ack   : OUT std_logic;
        clk   : IN  std_logic;
        reset : IN  std_logic);  -- asynchronous reset
END double;
```

The behavior
If at the rising edge of the *clk* input *req* is ‘1’ then the system multiplies the input value with two. The output *q* is only valid for 1 clock period and is indicated with output *ack* is ‘1’. The number of clock cycles needed to calculate to output value is data dependent.

A new request is allowed after the acknowledgement of a previous request.

```
ARCHITECTURE bhv OF double IS
BEGIN
  PROCESS(reset,clk)
    VARIABLE cnt : natural := 0;
    VARIABLE rdy : boolean := TRUE;
  BEGIN
    IF reset='0' THEN
      ack <='0';
      cnt := 0;
      q <= 0;
      rdy := TRUE;
    ELSIF rising_edge(clk) THEN
      ack<= '0';
      IF req='1' THEN
        cnt := 1; rdy := FALSE;
      ELSIF (cnt < data) and not rdy THEN
        cnt := cnt + 1;
      ELSIF not rdy THEN
        rdy := TRUE;
        ack <= '1';
        q <= 2 * data;
      ELSE
        ack <= '0';
        q <= 0;
      END IF;
    END IF;
  END PROCESS;
END bhv;
```

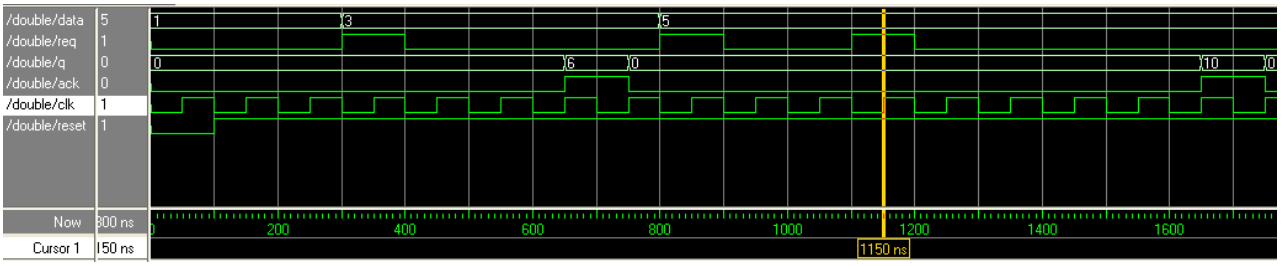

Fig 1. Waveform of a simulation run (file double.vhd and script file double.do)

A simulation of the design is shown in figure 1. Notice the request at 1150 ns whereas the request at 850 ns is not acknowledged at that time. With PSL it is possible to raise an error if an earlier request is not acknowledged before a new request is applied.

2. Property Specification Language

The following PSL property specifies that each request (req) must eventually be followed with an acknowledge (ack).

```
psl  property ack_after_request is
     always ( req -> eventually! ack )
     @rising_edge(clk);

psl  ack_after_req: assert ack_after_request;
```

In bold the interesting part is highlighted. If there is a request (the *enabling* condition) then eventually there must be an acknowledge (*fulfilling condition*). The check is performed on a rising edge of the clk.

Notes

1. The enabling condition and fulfilling condition can also be a sequence.
   E.g. assume that if a='1' and in the next cycle b='1' THEN eventually c='1' followed with d='1' in the cycle after c is '1':
   ```
   always ( {a;b} -> eventually! {c;d} ) @rising_edge(clk)
   ```
   If it is known that the fulfilling condition must be in the next clock cycle after the enabling condition write:
   ```
   always ( {a;b} -> next {c;d} ) @rising_edge(clk)
   ```
2. Most assertions will use the same active edge of the clock. The default edge is specified as:
   ```
   -- PSL default clock is (rising_edge(clk));
   -- PSL property ack_after_request is always ( req -> eventually! ack );
   -- PSL ack_after_req: assert ack_after_request;
   ```

This PSL code can be part of the VHDL description (or in a separate verification unit). PSL is included in the VHDL 2008 standard.

File: double_PSL1.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY double IS
  PORT (data  : IN  positive;
        req   : IN  std_logic;
        q     : OUT natural;
        ack   : OUT std_logic;
        clk   : IN  std_logic;
        reset : IN  std_logic); -- async. reset
END double;

ARCHITECTURE bhv OF double IS
BEGIN

-- PSL default clock is (rising_edge(clk));

-- PSL  property ack_after_request is always ( req -> eventually! ack );

-- PSL  ack_after_req: assert ack_after_request;
  PROCESS(reset,clk)
    VARIABLE cnt : natural := 0;
    VARIABLE rdy : boolean := TRUE;
    ...
```

Quastasim supports PSL (Modelsim-Altera starter does not support PSL). In figure 2 a script file is given that enables PSL.

```
quit -sim
vsim -assertdebug double   # double is name of the entity
view assertions
view fcovers
add wave *
# add to wave the "name of the assertion"
add wave ack_after_req
force clk 0, 1 50ns -rep 100ns
force data 1
force req 0
force reset 0, 1 100ns
run 300ns

# correct
force data 3
force req 1, 0 100ns
run 500ns

# overlapping request
force data 5
force req 1, 0 100ns
run 300ns
force req 1, 0 100ns
run 300ns
```

fig. 2. file double_PSL1.do

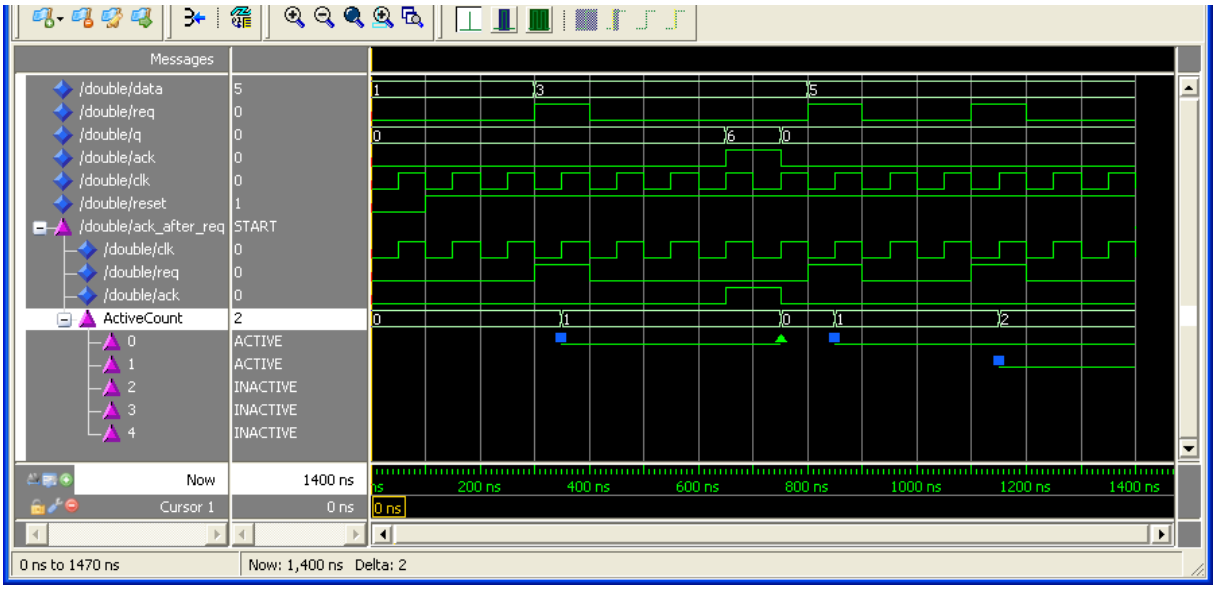

Fig 3. waveform of a simulation run with the assertion ack_after_req.

The waveform with the PSL assertion is shown in figure 3. The last wave "/double/ack_after_req" shows that the assertion becomes active at time 350 ns (req is '1') (blue box) and after an acknowledge it is inactive again. The green triangle at the end of the active period indicates that the property is passed. If you stop the simulation there are still two requests waiting for an acknowledge. If you type in QuestaSim:

quit –sim

you will see the following two errors in the transcript window:

```
# ** Error: Assertion failed
#    Time: 1400 ns Started: 850 ns  Scope: /double/ack_after_req File:
C:/tutorial/double_PSL1.vhd Line: 19
# ** Error: Assertion failed
#    Time: 1400 ns Started: 1150 ns  Scope: /double/ack_after_req File:
C:/tutorial/double_PSL1.vhd Line: 19
```

If the same simulation is repeated and an extra simulation run over 500 ns is performed both assertions pass (see figure 4).

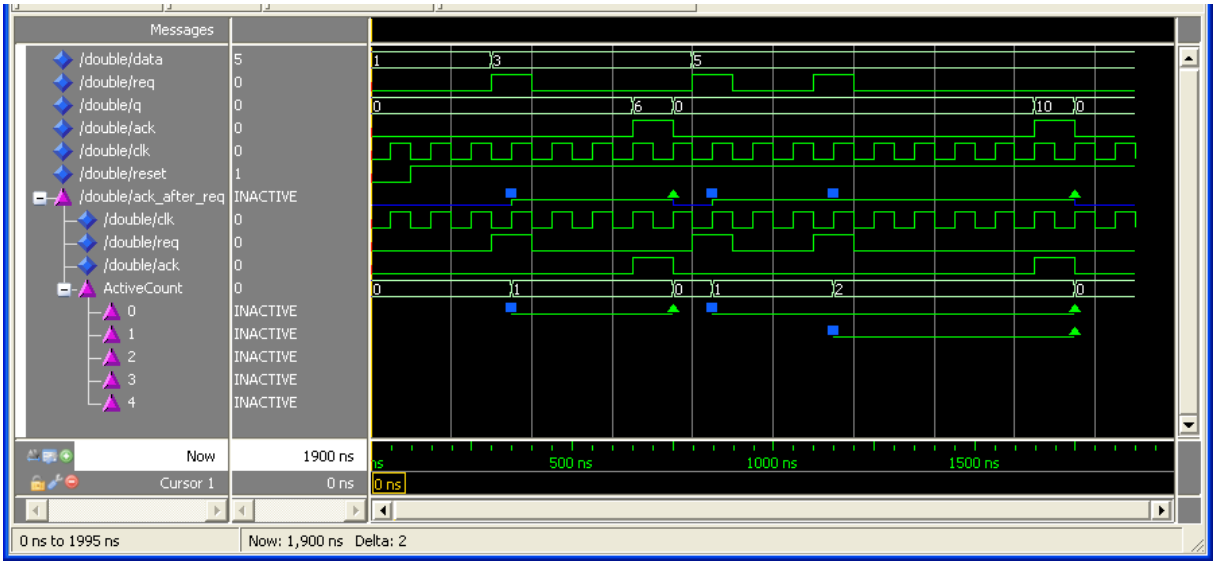

Fig 4. waveform of a simulation run with the assertion (script file double.do and an extra run over 500 ns).

Notice that for the assertion the last acknowledge “belongs” to the last two requests!

3. Acknowledge after 4 clock cycles

Assume a request must be acknowledged exactly after 4 clock cycles. This could be specified in PSL as:

```
-- psl  property ack_4_cycles_after_request is
--                              always ( req -> {[*4] ; ack } );

-- psl  ack_4cycles_req: assert ack_4_cycles_after_request;
```

Compile file `double_PSL1_4cycles.vhd` and perform a simulation until 1900 ns (the script file `double_PSL1_4cycles.do` and an additional run of 500 ns) a request at 850 ns expects an acknowledge at 1250 ns (figure 5). The red triangle indicates that the assertion failed. QuestaSim also generates a table (figure 6) with the results.

Note: it is also possible to move the PSL statements from the assertions tab to the wave window instead of changing the script file.

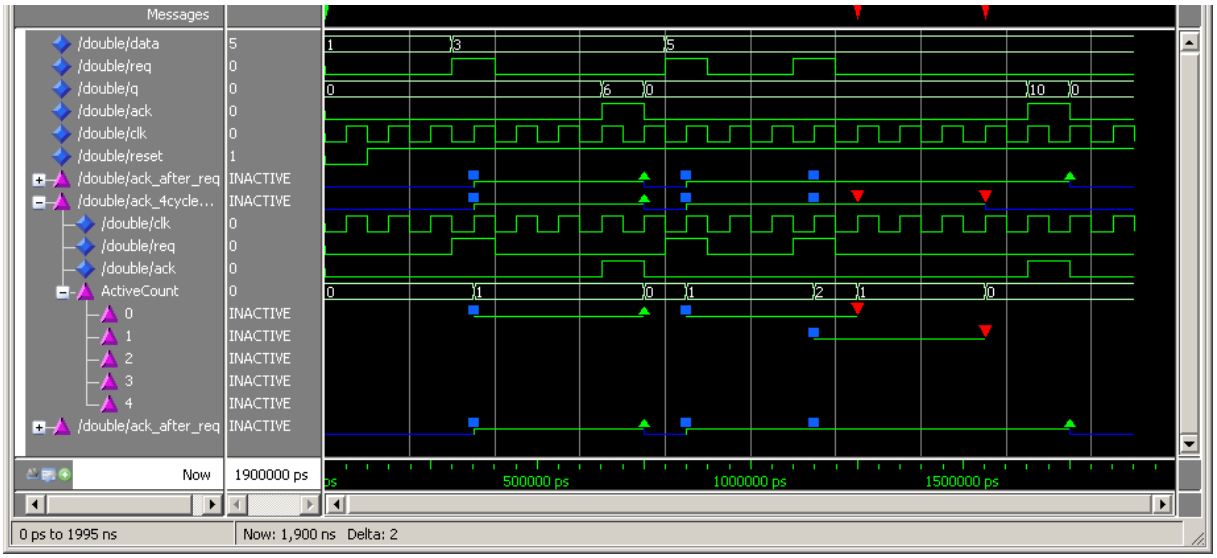

Fig 5. Acknowledge expected at clock cycle 4 after request.

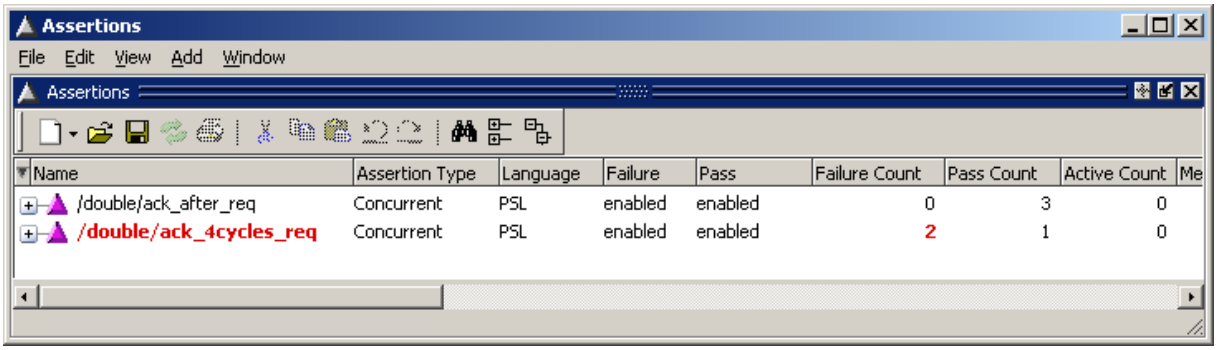

Fig 6. The assertion window summarizes the results.

4. Acknowledge a request before applying a new request.

In the previous example two consecutive requests without an acknowledge occurs. If this is prohibited the following property can be used:

```
psl  property no_overlapping_request is
     always ( req -> next (ack before req) );

psl  no_overlapping_req: assert no_overlapping_request;
```

The highlighted part specifies that after a request there must be first an acknowledge before a new request can be applied. This assertion is added in file `double_PSL2_no_overlap_req.vhd` (and script file `double_PSL2_no_overlap_req.do`).

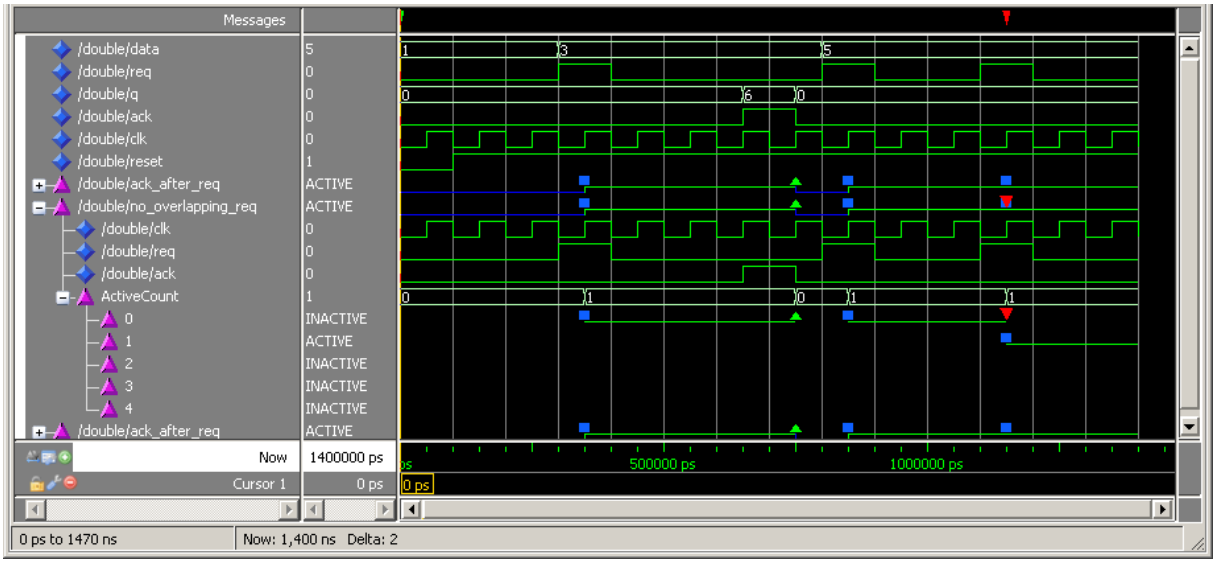

Fig 7. Violation of overlapping requests is detected.

With both properties we have checked that:

- a request must always be followed by an acknowledge
- after a request there must be an acknowledge before a new request can be applied

5. What about the reset?

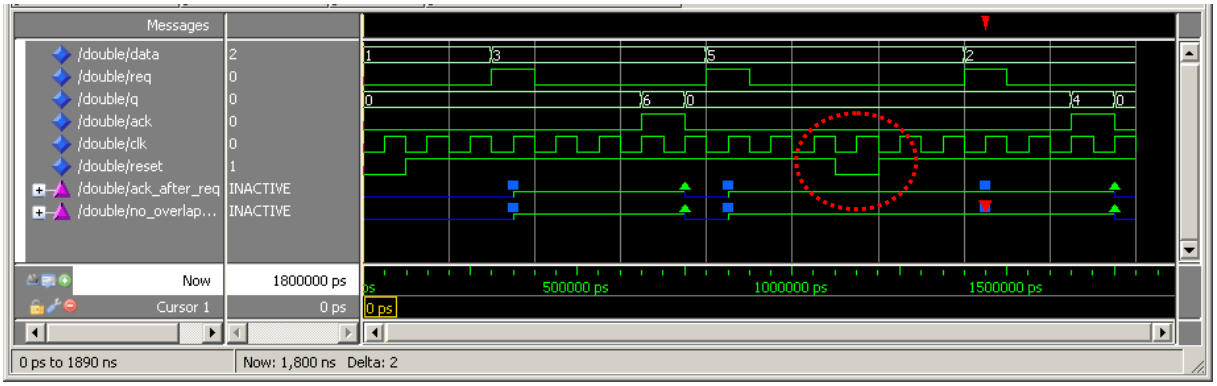

Fig 7. Violation of overlapping requests detected (script file double_reset.do).

In figure 7 a simulation is performed of the design in file `double_PSL2_no_overlap_req.vhd` but with script file `double_PSL3_reset.do`. A reset is applied and then an assertions should be aborted. The abort condition is specified in PSL:

```
-- PSL default clock is (rising_edge(clk));

-- psl property ack_after_request is
--      always ( ( req -> eventually! ack ) abort reset='0');

-- psl property no_overlapping_request is
--      always ( ( req -> next (ack before req) ) abort not reset);

-- psl ack_after_req: assert ack_after_request;
-- psl no_overlapping_req: assert no_overlapping_request;
```

Signal reset is of type std_logic but in PSL this is also interpreted as a Boolean: a '1' is true, and a '0' is false. File `double_PSL2_no_overlap_req.vhd` include these assertions.

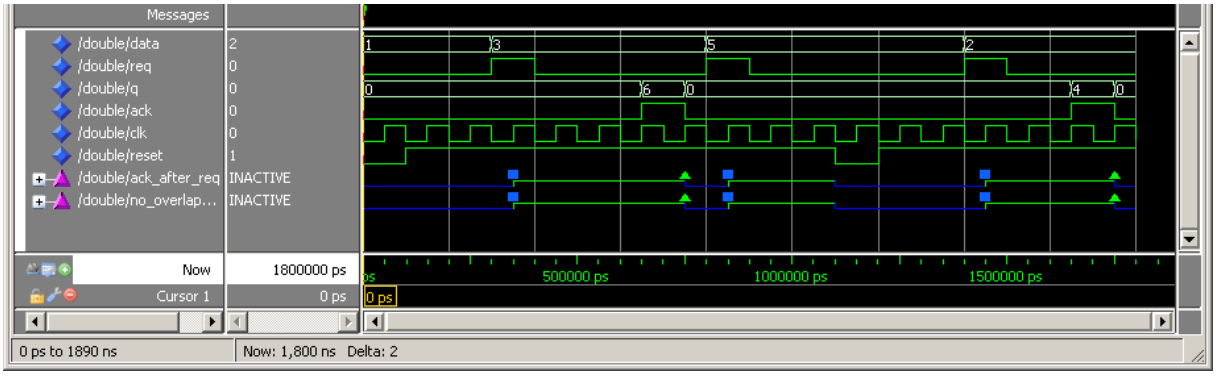

Fig 8. No violation of overlapping requests due to the abort condition.