Chapter 14 VHDL for Engineers Solutions

January 20, 2009 9:00 am

14.1 The two stimulus signals always required in a testbench for a synchronous system are a system clock and a reset. The basic criterion for these signals is that they must accurately model the clock and reset that will exist in the UUT's application environment.

14.2 The expression `period`/2 specifies an integer division. This leads to the possibility of an error in the period of the actual clock signal produced due to truncation of the quotient of the division. Whether the error occurs or not depends on the resolution of the simulation. For example, if the value of the constant `period` is specified as 15 ns and the simulation resolution is 1 ns, the period of the actual clock signal produced is 14 ns. However, if the constant `period` is specified as 15 ns and the resolution is 1 ps, the period of the actual clock signal produced is the desired 15 ns. This is because the division is 15000/2 = 7500, in units of ps, rather than 15/2 = 7, in units of ns. Simulators usually allow the resolution for a specific simulation to be specified.

14.3 A clock with other than a 50% duty cycle can be created with a concurrent signal assignment statement. For example to create a clock with a 40% duty cycle the following conditional concurrent signal assignment statement can be used.

```
clk <= '0' after 0.4 * period when clk = '1' else '1' after 0.6 * period;
```

14.4 A concurrent signal assignment statement can be used to describe a clock that is stopped by a boolean signal. The following statement creates a clock with a 50% duty cycle that is stopped by a boolean signal.

```
clk <= not clk after period/2 when end_sim = false else '0';
```

Signal `end_sim` must be initialized to false.

A clock with other than a 50% duty cycle that is stopped by a boolean signal can also be described by a concurrent signal assignment. The following statement creates a clock with a 40% duty cycle that is stopped by a boolean signal.

```
clk <= '0' after 0.4 * period when (clk = '1' and end_sim = false)
else '1' after 0.6 * period when end_sim = false else '0';
```

14.5 (a) The process given in the problem statement successfully implements a clock with a 50% duty cycle. The code works because a process without a sensitivity list is an infinite loop.

(b) The code below shows `end_sim` being added to the clock process.

```
clock : process
begin
    clk <= '0';
    wait for period/2;
    clk <= '1';
    wait for period/2;
    if end_sim = true then
```

```
            wait;
        end if;
    end process;
```

14.6 The requested procedure is shown below in a design entity that invokes the procedure to create a 10 MHz waveform with a 30% duty cycle.

```
library ieee;
use ieee.std_logic_1164.all;

entity clock is
    port(clk_out : out std_logic);
end clock;

architecture cncur_proc of clock is

    procedure clk_gen (constant freq, duty_cycle : in real; signal clk : out std_logic) is
        constant period : time := 1.0 us / freq;
        constant clk_high : time := duty_cycle * period;
    begin
        loop
            clk <= '0';
            wait for period - clk_high;
            clk <= '1';
            wait for clk_high;
        end loop;
    end procedure ;
begin

    clk_gen (freq => 10.0, duty_cycle => 0.30, clk => clk_out);

end cncur_proc;
```

14.7

```
    rst <= ‘1’, ‘0’ after 250 ns;
```

14.8 The request procedure follows in an example that includes its invocation.

```
library ieee;
use ieee.std_logic_1164.all;

entity clk_n_rst_entity is
    port(
        clk_out : inout std_logic;
        reset_out : inout std_logic
        );
end clk_n_rst_entity;

architecture behavioral of clk_n_rst_entity is

    -- procedure requested in problem 14.8
    procedure clk_n_rst (
        period : in time;
        rst_asserted : in std_logic ;
        constant rst_duration : in integer;
        clk_initial : in std_logic ;
        clks_after_rst : in integer;
        signal reset : inout std_logic ;
        signal clk : inout std_logic ) is
    begin
        clk <= clk_initial;
        reset <= rst_asserted;

        for i in 1 to 2 * rst_duration loop
            wait for period/2;
            clk <= not clk;
        end loop;

        reset <= not rst_asserted;

        for i in 1 to 2 * clks_after_rst loop
            wait for period/2;
            clk <= not clk;
        end loop;
        wait;

    end procedure clk_n_rst;

begin
    -- sample invokation of the procedure
    clk_n_rst (period => 50 ns, rst_asserted => '0', rst_duration => 2,
    clk_initial => '0', clks_after_rst => 6, reset => reset_out,
    clk => clk_out);

end behavioral;
```

14.9 All signals that have new values assigned to them during a particular simulation cycle have those values assigned during the update phase, at the beginning of the cycle. Thus, if a new value is assigned to the D input of a flip-flop in the same simulation cycle that a 1 is assigned to its clock, those assignments both take place during the update phase. After these signals have been updated to their new values a determination is made as to which signals have events on them. If clock was previously 0, then there was an event on clock and all processes sensitive to this event are placed in the active processes queue. Accordingly, the process that describes the D flip-flop is placed in the active processes queue. This process is not executed until the execution phase of the same simulation cycle. When it is executed, the D input already has its new value and that value is stored in the flip-flop.

14.10 One can argue that a stimulus that shifts a single 1 through the simple shift register of Listing 9.2.1 provides a comprehensive verification. The problem states that vectors must be used to provide the stimulus and expected response. The stimulus provided in the testbench that follows includes the shifting of a single 1 through the shift register as the first part of the stimulus vector.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity shiftreg_rb_tb is
end shiftreg_rb_tb;

architecture tb_architecture of shiftreg_rb_tb is

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal si : std_logic;
    signal clr_bar : std_logic;
    signal clk : std_logic := '0';
    -- Observed signals - signals mapped to the output ports of tested entity
    signal qout : std_logic_vector(3 downto 0);

    constant period : time := 20 ns;
    signal end_sim : boolean := false;

    -- stimulus and response signal vectors
    constant inputs : std_logic_vector :=  "1000011011100110001011";
    constant outputs : std_logic_vector := "0001000011011100110001";
begin

    -- Unit Under Test port map
    UUT : entity shiftreg_rb
    port map (
        si => si,
        clr_bar => clr_bar,
        clk => clk,
        qout => qout
        );

    -- system clock generation
    cpu_clock_gen : clk <= not clk after period/2 when end_sim = false else unaffected;

    -- system reset generation
    reset: process
    begin
        clr_bar <= '0';
        for i in 1 to 2 loop
            wait until clk = '0';
        end loop;
        clr_bar <= '1';
        wait;
    end process;

    -- stimulus and response process
    stim_respon : process
    begin
        wait until clr_bar = '1';
        for i in inputs'range loop
            si <= inputs(i);
            wait until clk = '0';
            assert qout(0) = outputs(i)
            report "system failure at " & integer'image(i) & " clocks after reset"
            severity error;
        end loop;
        end_sim <= true;
        wait;
    end process;
end tb_architecture;
```

14.11 The counter in Listing 9.4.1 is a simple 4-bit up counter. Clearing the counter and then causing it to count through its full range and then roll over provides an exhaustive verification.

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity counter_4bit_tb is
end counter_4bit_tb;

architecture tb_architecture of counter_4bit_tb is

    -- Stimulus signals
    signal clk : std_logic := '0';
    signal reset_bar : std_logic;
    -- Observed signals
    signal count : std_logic_vector(3 downto 0);

    signal period : time := 50 ns;
    signal end_sim : boolean := false;

begin

    -- Unit Under Test port map
    UUT : entity counter_4bit
    port map (
        clk => clk,
        reset_bar => reset_bar,
        count => count
        );

    -- clock
    clk <= not clk after period/2 when end_sim = false else '0';

    -- reset
    reset_bar <= '0', '1' after 2 * period;

    monitor: process
        constant n : integer := 4; -- number of bits in counter
    begin

        wait until reset_bar = '1';

        wait for 1 ns;
        -- verify counter’s output after each clock
        for i in 0 to 2**n loop -- count until count rolls over to 0
            assert count = (std_logic_vector(to_unsigned(i mod 2**n, n)))
            report "Count of " & integer'image(i mod 2**n) & " failed"
            severity error;
            wait until clk = '1';
            wait for 1 ns;
        end loop;
        end_sim <= true;
        wait;
    end process;

end tb_architecture;
```

14.12 The 12-bit counter in Listing 9.4.4 is an up-down counter with count enable and direction inputs. The testbench which follows provides a comprehensive verification of the counter by first clearing the counter and disabling it from counting. Five clocks occur with the counter disabled. The counter is then enabled to count up and allowed to count through its entire range with the count verified before each triggering clock edge. The counter's direction is then changed to down and the counter is allowed to count down through its entire range with the count verified before each triggering clock edge.

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity binary_cntr_tb is
end binary_cntr_tb;

architecture tb_architecture of binary_cntr_tb is

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal clk : std_logic := '0';
    signal cnten : std_logic;
    signal up : std_logic;
    signal rst_bar : std_logic;
    -- Observed signals - signals mapped to the output ports of tested entity
    signal q : std_logic_vector(11 downto 0);

    constant period : time := 40 ns;
    signal end_sim : boolean := false;
begin

    -- Unit Under Test port map
    UUT : entity binary_cntr
    port map (
        clk => clk,
        cnten => cnten,
        up => up,
        rst_bar => rst_bar,
        q => q
        );

    -- clock
    clk <= not clk after period/2 when end_sim = false else '0';

    -- reset
    rst_bar <= '0', '1' after 2 * period;

    stim_monitor: process
        constant n : integer := 12; -- number of bits in counter
    begin
        cnten <= '0';    -- disable counter
        up <= '1';       -- set direction to count up

        wait until rst_bar = '1';    -- wait until reset is complete
        wait for 1 ns;

        for i in 0 to 4 loop         -- five clocks with cnten unasserted
            wait until clk = '1';    -- counter must not count
        end loop;

        cnten <= '1';                -- enable counter to count
        wait for 1 ns;

        -- verify counter's output after each clock while counting up
        for i in 0 to 2**n loop -- count until count rolls over to 0
            assert q = (std_logic_vector(to_unsigned(i mod 2**n, n)))
            report "Count of " & integer'image(i mod 2**n) & " failed"
            severity error;
            wait until clk = '1';
            wait for 1 ns;
        end loop;
        -- count is 1 when this loop is completed

        up <= '0';
        wait until clk = '1';    -- count is 0 after this clock
        wait for 1 ns;

        -- verify counter's output after each clock while counting down
        for i in 2**n downto 0 loop -- count until count rolls over to 0
            assert q = (std_logic_vector(to_unsigned(i mod 2**n, n)))
            report "Count of " & integer'image(i mod 2**n) & " failed"
            severity error;
            wait until clk = '1';
            wait for 1 ns;
        end loop;
        end_sim <= true;
        wait;
    end process;

end tb_architecture;
```

14.13 This testbench includes a clock generation procedure that, when called, generates the number of clock cycles specified by a parameter passed to the procedure. The testbench uses this procedure to cause the counter to count a specific number of times and then verifies the count. Intermediate counts are not verified. This techniques is repeated at several points through the counter's range when counting up and then when counting down.

```
library ieee;
use ieee.numeric_std.all;
```

```vhdl
use ieee.std_logic_1164.all;

entity bcd_2dec_tb is
end bcd_2dec_tb;

architecture tb_architecture of bcd_2dec_tb is

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal clk : std_logic;
    signal rst_bar : std_logic;
    signal up_bar : std_logic;
    signal cnt_en1_bar : std_logic;
    signal cnt_en2_bar : std_logic;
    -- Observed signals - signals mapped to the output ports of tested entity
    signal qbcd0 : std_logic_vector(3 downto 0);
    signal qbcd1 : std_logic_vector(3 downto 0);
    constant period : time := 100 ns;

    procedure clk_gen (signal clk : out std_logic; constant n : in integer) is
        constant period : time := 100 ns;
    begin
        for i in 1 to n loop
            clk <= '0';
            wait for period/2;
            clk <= '1';
            wait for period/2;
        end loop;
    end procedure;

begin

    -- Unit Under Test port map
    UUT : entity bcd_2dec
    port map (
        clk => clk,
        rst_bar => rst_bar,
        up_bar => up_bar,
        cnt_en1_bar => cnt_en1_bar,
        cnt_en2_bar => cnt_en2_bar,
        qbcd0 => qbcd0,
        qbcd1 => qbcd1
        );

    rst_bar <= '0', '1' after (3 * period)/4;

    stim_mon : process
    begin
        up_bar <= '0';      -- initialize count direction
        cnt_en1_bar <= '1'; -- counter not enabled
        cnt_en2_bar <= '0';

        clk_gen(clk,1);   -- generate one clock

        assert qbcd1 = "0000" and qbcd0 = "0000"    -- check counter reset
        report "counter did not reset"
        severity error;

        cnt_en1_bar <= '0'; -- enable counter

        clk_gen(clk, 29); -- count up 29 times (counter result 29)

        assert qbcd1 = "0010" and qbcd0 = "1001"
        report "counter failed to count up to 29"
        severity error;

        clk_gen(clk, 8); -- count up 37 times (counter result 5)

        assert qbcd1 = "0000" and qbcd0 = "0101"
        report "counter failed count to roll over on count up"
        severity error;

        up_bar <= '1'; -- change direction of count

        clk_gen(clk, 4); -- count down 4 times (counter result 1)

        assert qbcd1 = "0000" and qbcd0 = "0001"
        report "counter failed count to count down to 1"
        severity error;

        clk_gen(clk, 3); -- count down 3 times (counter result 30)

        assert qbcd1 = "0011" and qbcd0 = "0000"
        report "counter failed count to underflow"
        severity error;

        wait;

    end process;

end tb_architecture;
```

14.14 The testbench generates the waveforms in Figure 9.5.2 for the positive edge detector of Listing 9.5.1.

```
library ieee;
use ieee.std_logic_1164.all;

entity posedge_tb is
end posedge_tb;

architecture tb_architecture of posedge_tb is
    signal a : std_logic;
    signal a_pe : std_logic;
    signal clk : std_logic := '1';
    signal clr_bar : std_logic;
    signal end_sim : boolean := false;
    constant period : time := 100 ns;
begin
```

```
    UUT : entity posedge
    port map(
        a => a,
        a_pe => a_pe,
        clk => clk,
        clr_bar => clr_bar
        );

    clock : process
    begin
        if end_sim = false then
            clk <= not clk;
        else
            wait;
        end if;
        wait for period/2;
    end process;

    stimulus : process
    begin
        a <= '0';
        clr_bar <= '0';
        wait for 10ns;
        clr_bar <= '1';
        wait for 100ns;
        a <= '1';
        wait for 200ns;
        a <= '0';
        wait for 160ns;
        a <= '1';
        wait for 210ns;
        a <= '0';
        wait for 100ns;
        a <= '1';
        wait for 20ns;
        a <= '0';
        wait for 200ns;
        end_sim <= true;
        wait;
    end process;

end tb_architecture;
```

14.15 The testbench for the single shot creates each possible trigger condition and the measures the length of the pulse generated and compares it with the expected length.

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity single_shot_tb is
end single_shot_tb;
```

```
architecture tb_architecture of single_shot_tb is

    -- Stimulus signals
    signal a : std_logic;
    signal b : std_logic;
    signal clr_bar : std_logic;
    signal clk : std_logic := '0';
    -- Observed signals
    signal q : std_logic;

    signal end_sim : boolean := false;
    constant period : time := 20 ns;
    constant n : integer := 5;      -- same as load value for single shot

begin
    -- Unit Under Test port map
    UUT : entity single_shot
    port map (
        a => a,
        b => b,
        clr_bar => clr_bar,
        clk => clk,
        q => q
        );

    clr_bar <= '0', '1' after 1.5 * period;

    clk <= not clk after period/2 when end_sim = false else '0';

    stim_mon : process
        variable start, stop, duration : time := 0 ns;
    begin

        a <= '0';       -- initialize a and b
        b <= '0';

        wait until clk = '1';
        wait until clk = '1';

        b <= '1';       -- fire single shot, because a = 0 and pos. edge on b

        wait until q = '1'; -- timestamp start of ss pulse
        start := now;

        wait until q = '0'; -- timestamp end of pulse
        stop := now;

        duration := stop - start;

        report "actual pulse duration equals " & time'image(duration);
        report "expected pulse duration equals " & time'image(n * period);

        assert duration = (n * period)
        report "pulse not correct length"
        severity error;

        wait until clk = '1';

        a <= '1';

        wait until clk = '1';

        a <= '0';       -- fire single shot, because b = 1 and neg. edge on a

        wait until q = '1'; -- timestamp start of ss pulse
        start := now;

        wait until q = '0'; -- timestamp end of pulse
        stop := now;

        duration := stop - start;

        report "actual pulse duration equals " & time'image(duration);
        report "expected pulse duration equals " & time'image(n * period);

        assert duration = (n * period)
        report "pulse not correct length"
        severity error;

        end_sim <= true;
        wait;

    end process;

end tb_architecture;
```

14.16

```
clk <= '0', '1' after period/2, '0' after period, '1' after (3* period)/2, '0' after 2 *
period,
'1' after (5 * period)/2, '0' after 3 * period, '1' after (7 * period)/2, '0' after 4 *
period,
'1' after (9 * period)/2, '0' after 5 * period, '1' after (11 * period)/2;

d <= '1', '0' after 2 * period, '1' after 5 * period;

set_bar <= '1', '0' after 3 * period, '1' after 4 * period;

clear_bar <= '0', '1' after period;
```

14.17

```
    process
        constant period : time := 20 ns;
    begin
        clk <= '0';     -- clock cycle 1
        d <= '1';
        set_bar <= '1';
        clear_bar <= '0';
        wait for period/2;
        clk <= '1';
        wait for period/2;

        clk <= '0';     -- clock cycle 2
        clear_bar <= '1';
        wait for period/2;
        clk <= '1';
        wait for period/2;

        clk <= '0';     -- clock cycle 3
        d <= '0';
        wait for period/2;
        clk <= '1';
        wait for period/2;

        clk <= '0';     -- clock cycle 4
        set_bar <= '0';
        wait for period/2;
        clk <= '1';
        wait for period/2;

        clk <= '0';     -- clock cycle 5
        set_bar <= '1';
        wait for period/2;
        clk <= '1';
        wait for period/2;

        clk <= '0';     -- clock cycle 6
        d <= '1';
        wait for period/2;
        clk <= '1';
        wait for period/2;
        wait;
    end process;
```

14.18

```
library ieee;
use ieee.std_logic_1164.all;

package signal_probe is
    procedure period_probe (signal a : in std_logic; signal period : inout time;
    signal frequency : out real);
end signal_probe;

package body signal_probe is
    procedure period_probe (signal a : in std_logic; signal period : inout time;
        signal frequency : out real) is
        variable pe1, pe2 : time := 0 ps;
    begin
        loop
            wait until a = '1';
            pe1 := now;
            wait until a = '1';
            pe2 := now;
            period <= pe2 - pe1;
            frequency <= real((1.0e12 ps/ period));
        end loop;
    end period_probe;
end signal_probe;
```