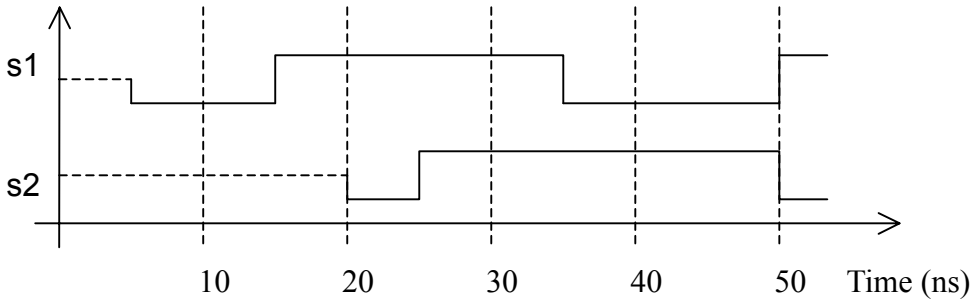
03S2 COMP3211/9211 Computer Architecture

Tutorial Week 3 Solutions

Newton Cheung
ncheung@cse.unsw.edu.au
10 August 2003

Q01. [Y3.3] Sketch the output waveform produced by the following VHDL concurrent signal assignment statements:

s1 <= '0' after 5 ns, '1' after 15 ns, '0' after 35 ns, '1' after 50 ns;
s2 <= '0' after 20 ns, '1' after 25 ns, '0' after 50 ns;



Q02. [Y3.6] Write and simulate the entity-architecture description of a 3-bit decoder using the conditional signal assignment statement. Test the model with all possible combinations of inputs and plot the decoder output waveform. Try to use the ISE/WebPACK tool for this and following exercises.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder is
        port ( input1, input2, input3 : in std_logic;
                output : out std_logic_vector(7 downto 0));
end decoder;

architecture behavioral of decoder is
signal tmp : std_logic_vector(2 downto 0);
begin
        tmp <= input3 & input2 & input1;
        with tmp select
                output <=
```

```
        x"01" when "000",
        x"02" when "001",
        x"04" when "010",
        x"08" when "011",
        x"10" when "100",
        x"20" when "101",
        x"40" when "110",
        x"80" when others;
end behavioral;
```

Q03. Write and simulate the entity-architecture description of a single bit slice of a universal shifter using the conditional signal assignment statement. The slice should have left, right, and load inputs, a 2-bit direction input, and a single output bit. Test the model with all possible combinations of inputs and plot the shift slice output waveform.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity bitS is
    port ( Load, Right, Left : in std_logic;
        Sel: in std_logic_vector(1 downto 0);
        Z : out std_logic);
end bitS;

architecture behavioral of bitS is
begin
    Z <= Load when Sel = "00" else
        Left when Sel = "01" else
        Right when Sel = "10" else
        '0';
end behavioral;
```

Q04. [Y3.9] Why are the concepts of delta events and delta delays necessary for the correct event simulation of digital circuits?

A delta event/delta cycle is an infinitesimal delay used by the discrete event simulator to correctly sequence signal flows through gates when there are no delays specified - without them, incorrect output signal can end up being assigned.

Q05. [Y4.2] Explain why you cannot have both a sensitivity list and wait statements within a process

"Sensitivity list is the list of signals to which the process is sensitive. This is a set of signals - enclosed in parentheses - which the simulator monitors for events (changes of value). Any events on any of the signals in the sensitivity list causes the process to be executed once. That is, all the statements in the statement part will be executed and then the process will stop and wait for further activity. Every time the process is activated it will run in its entirety."

"Process with the wait statement at the end will be executed in their entirety at the start of simulation whereas processes with the wait statement at the beginning will not be executed at the start of simulation."

Andrew Rushton, "VHDL for Logic Synthesis "

"It is an error if a wait statement appears in an explicit process statement that includes a sensitivity list or in a procedure that has a parent that is such a process statement."

"IEEE Standard VHDL Language Reference Manual "

Q06. [Y4.6] Show an example of VHDL code that transforms an input periodic clock signal to an output signal at half the frequency. (Harder: Implement a procedure that doubles the input frequency.)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity half_freq is
        port (clk_in, reset : in std_logic;
                clk_out : out std_logic);
end half_freq;

architecture behavioral of half_freq is
signal clk_out_tmp : std_logic;
begin
        half_period: process (clk_in, reset)
        begin -- process half_period
                if reset = '1' then -- asynchronous reset (active high)
```

```
                clk_out_tmp <= ‘0’;
            elsif clk_in'event and clk_in = '1' then -- rising clock edge
                clk_out_tmp <= not(clk_out_tmp); -- a temporary value for read/write
            end if;
        end process half_period;
        clk_out <= clk_out_tmp;
    end behavioral;
```

++

```
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.STD_LOGIC_ARITH.ALL;
    use IEEE.STD_LOGIC_UNSIGNED.ALL;

    entity double_freq is
        Port ( clk : in std_logic;
               reset : in std_logic;
               clk_out : out std_logic);
    end double_freq;

    architecture Behavioral of double_freq is
    signal period: Time := 10 ps;
    signal tmp: Time;
    signal clk_out_tmp: std_logic := '0';
    begin
        find_period: process(clk, reset, start)
        begin
            if clk'event and clk = '1' then
                tmp <= now;
            elsif clk'event and clk = '0' then
                period <= now - tmp;
            end if;
        end process find_period;

        new_freq: process
        begin
            clk_out_tmp <= not (clk_out_tmp);
```

```
            wait for period/2;
        end process new_freq;
        clk_out <= clk_out_tmp;
end Behavioral;
```

This procedure is only stable after first few clock cycles.

Q07. Model a synchronous 4-bit counter using behavioural VHDL.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity syn_4b_counter is
        port (clk, reset : in std_logic;
            output : out std_logic_vector(3 downto 0);
end syn_4b_counter;

architecture behavioral of syn_4b_counter is
signal tmp_out : std_logic_vector(3 downto 0);
begin
        counter : process (clk, reset, tmp_out)
        begin
            if reset = '1' then -- asynchronous reset (active high)
                tmp_out <= “0000”;
            elsif clk'event and clk = '1' then -- rising clock edge
            if tmp_out = “1111” then
                tmp_out <= “0000”;
            else
                tmp_out <= tmp_out + “0001”;
            end if;
            end if;
        end process counter;
        output <= tmp_out;
end behavioral;
```