COMP3211/COMP9211 Computer Architecture
Week 4 Tutorial Exercises (Solution)

Q1. [Y4.4] Consider the construction of a register file with 8 registers, where each register is 32 bits. Implement the model with two processes. One process reads the register file, while another writes to the register file. You can implement the registers as signals declared within the architecture and therefore visible to each process.

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

entity q2 is
     port (clk, reset : in std_logic;
          reg_num : in std_logic_vector(2 downto 0);
          reg_in : in std_logic_vector(31 downto 0);
          reg_out : out std_logic_vector(31 downto 0);
          read_write : in std_logic);
     end q2;

architecture behav of q2 is
     type reg_file is array (0 to 7) of std_logic_vector(31 downto 0);
     signal reg_content : reg_file;
begin -- behav
read_process: process (clk, reset, read_write)
begin -- process read_process
     if reset = '1' then -- asynchronous reset (active high)
          reg_out <= (others => '0');
     elsif clk'event and clk = '1' and read_write = ‘1’ then
          reg_out <= reg_content(conv_integer(reg_num));
     end if;
end process read_process;

write_process: process (clk, reset, read_write)
begin -- process write_process
     if reset = '1' then -- asynchronous reset (active high)
          reg_content(0) <= (others => '0');
          reg_content(1) <= (others => '0');
          reg_content(2) <= (others => '0');
          reg_content(3) <= (others => '0');
          reg_content(4) <= (others => '0');
          reg_content(5) <= (others => '0');
          reg_content(6) <= (others => '0');
          reg_content(7) <= (others => '0');
     elsif clk'event and clk = '1' and read_write = ‘0’ then
          reg_content(conv_integer(reg_num)) <= reg_in;
     end if;
end process write_process;
end behav;
```

Q2. [Y4.9] Implement and test a VHDL model for the state machine for a traffic controller described in the following state table:

Current State	Input	Next State	Output
0	0	1	10
0	1	2	00
1	0	0	01
1	1	2	00
2	0	0	01
2	1	2	00

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

entity q4 is
     port(clk, reset : in std_logic;
          in1 : in std_logic;
```

```
        out1 : out std_logic_vector(1 downto 0));
end q4;

architecture behav of q4 is
signal current_state, next_state : std_logic_vector(1 downto 0);
begin -- behav
    cur_state: process (clk, reset)
            begin -- process cur_state
                if reset = '1' then -- asynchronous reset (active high)
                    current_state <= "00";
                elsif clk'event and clk = '1' then -- rising clock edge
                    current_state <= next_state;
                end if;
            end process cur_state;

    nx_state: process (current_state, in1)
            begin -- process nx_state
            case current_state is
                when "00" =>
                    if in1 = '0' then
                        next_state <= "01";
                    else
                        next_state <= "10";
                    end if;
                when "01" =>
                    if in1 = '0' then
                        next_state <= "00";
                    else
                        next_state <= "10";
                    end if;
                when "10" =>
                    if in1 = '0' then
                        next_state <= "00";
                    else
                        next_state <= "10";
                    end if;
                when others =>
                    next_state <= "00";
            end case;
        end process nx_state;

    out_logic: process (current_state, in1)
    begin -- process out_logic
        case current_state is
            when "00" =>
                if in1 = '0' then
                    out1 <= "10";
                else
                    out1 <= "00";
                end if;
            when "01" =>
                if in1 = '0' then
                    out1 <= "01";
                else
                    out1 <= "00";
                end if;
            when "10" =>
                if in1 = '0' then
                    out1 <= "01";
                else
                    out1 <= "00";
                end if;
            when others =>
                out1 <= "00";
        end case;
    end process out_logic;
end behav;
```

Q3. [Y5.3] You are part of a software group developing algorithms for processing speech signals for a new digital signal processing chip. To test your software your options are to construct i) a detailed hierarchical model

of the chip comprised of gate level models at the lowest level of the hierarchy or ii) a behavioural level model of the chip that can implement the algorithms that you wish to use. Your goal is to produce correct code for a number of algorithms prior to detailed testing on a hardware prototype. How would you evaluate these choices and what are the tradeoffs
in picking one approach over the other?

i) Advantage:
- Optimise design results
Disadvantage:
- Slow design time
- complex Design

ii) Advantage;
- Fast design time
Disadvantage:
- Design optimisation depends on the tools

Q4. [Y5.5] Consider the circuit shown below. Construct a structural model comprised of two components: a generic N-input AND gate and a two-input OR gate. By passing the appropriate generic value we can instantiate the same basic AND gate component as a two-input or three-input AND gate.

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

entity n_and is
    generic (input_num : integer := 2);
    port (n_input : in std_logic_vector(input_num-1 downto 0);
        c : out std_logic);
end n_and;

architecture rtl of n_and is
begin -- rtl
    nWay_and: process (n_input)
    variable c_tmp : std_logic;
    begin -- process nWay_and
        c_tmp := '1';
        for i in n_input'length-1 downto 0 loop
            c_tmp := c_tmp and n_input(i);
        end loop; -- i
        c <= c_tmp;
    end process nWay_and;
end rtl;
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+
library IEEE;
use IEEE.std_logic_1164.all;

entity or2 is
    port (a, b : in std_logic;
        c : out std_logic);
end or2;

architecture rtl of or2 is
begin -- rtl
    c <= (a or b);
end rtl;
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+
library IEEE;
use IEEE.std_logic_1164.all;

entity q6 is
    port (in1, in2, in3, in4, in5 : in std_logic;
    c : out std_logic);
```

```
end q6;

architecture struct of q6 is
signal s1, s2 : std_logic;
signal and2_in : std_logic_vector(1 downto 0);
signal and3_in : std_logic_vector(2 downto 0);

component or2
port (a, b : in std_logic;
c : out std_logic);
end component;

component n_and
generic (input_num : integer);
port (n_input : in std_logic_vector(input_num-1 downto 0);
c : out std_logic);
end component;

begin -- struct
    and2_in <= in1 & in2;
    and3_in <= in3 & in4 & in5;
    and2: n_and
        generic map (input_num => 2)
        port map (n_input => and2_in,
                  c => s1);
    and3: n_and
        generic map (input_num => 3)
        port map (n_input => and3_in,
                  c => s2);
    or2_i: or2
             port map (a => s1,
    b => s2,
    c => c);
end struct;
```

Q05. [YE5.2] The goal of this exercise is to introduce trade-offs in building models at different levels of abstraction and trading accuracy for simulation speed.

STEP 1 – 1 bit ALU

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

entity alu is
   Port ( a : in std_logic;
          b : in std_logic;
          c_in : in std_logic;
          opcode : in std_logic_vector(1 downto 0);
          result : out std_logic;
          c_out : out std_logic);
end alu;

architecture Behavioral of alu is
constant gate_delay: Time := 10 ns;

begin

p1:  process (a, b, c_in, opcode)
  variable tmp:std_logic;
  variable tmp_cout:std_logic;
begin

     tmp_cout:= (((not c_in) and b and a) or (c_in and (a or b)));

  if (opcode="00") then
     tmp := (a and b);
  elsif (opcode="01") then
     tmp := (a or b);
```

```
   elsif (opcode="10") then
      tmp := (((not c_in) and (a xor b)) or (c_in and (a xnor b)));
   else
      tmp :='X';
   end if;

   result <= tmp after gate_delay;
   c_out <= tmp_cout after gate_delay;
end process;

end Behavioral;
```

STEP 3 – 4 bit ALU

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

entity fourbit_ALU is
    Port ( a : in std_logic_vector(3 downto 0);
           b : in std_logic_vector(3 downto 0);
           c_in : in std_logic;
           opcode : in std_logic_vector(1 downto 0);
           result : out std_logic_vector(3 downto 0);
           c_out : out std_logic);
end fourbit_ALU;

architecture Behavioral of fourbit_ALU is
signal cout0,cout1,cout2: std_logic;

component ALU
   Port ( a : in std_logic;
          b : in std_logic;
          c_in : in std_logic;
          opcode : in std_logic_vector(1 downto 0);
          result : out std_logic;
          c_out : out std_logic);
end component;

begin
ALU1: ALU
     port map (a => a(0), b => b(0), c_in => c_in, opcode => opcode, result => result(0),
c_out => cout0);
ALU2: ALU
     port map (a => a(1), b => b(1), c_in => cout0, opcode => opcode, result => result(1),
c_out => cout1);
ALU3: ALU
     port map (a => a(2), b => b(2), c_in => cout1, opcode => opcode, result => result(2),
c_out => cout2);
ALU4: ALU
     port map (a => a(3), b => b(3), c_in => cout2, opcode => opcode, result => result(3),
c_out => c_out);

end Behavioral;
```

STEP 3 – 8 bit ALU

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

entity eightbit_ALU is
    Port ( a : in std_logic_vector(7 downto 0);
           b : in std_logic_vector(7 downto 0);
           c_in : in std_logic;
           opcode : in std_logic_vector(1 downto 0);
           result : out std_logic_vector(7 downto 0);
           c_out : out std_logic);
end eightbit_ALU;
```

```
architecture Behavioral of eightbit_ALU is
signal cout_connect: std_logic ;
component fourbit_ALU
Port ( a : in std_logic_vector(3 downto 0);
         b : in std_logic_vector(3 downto 0);
         c_in : in std_logic;
         opcode : in std_logic_vector(1 downto 0);
         result : out std_logic_vector(3 downto 0);
         c_out : out std_logic);
end component;

begin
   ALU_low: fourbit_ALU
           port map (a => a(3 downto 0), b=> b(3 downto 0), c_in=> c_in,opcode => opcode,
                    result => result(3 downto 0), c_out => cout_connect);
   ALU_high: fourbit_ALU
           port map (a => a(7 downto 4), b => b(7 downto 4), c_in=> cout_connect, opcode
                    => opcode, result => result(7 downto 4), c_out =>  c_out);
end Behavioral;
```

step 6 – Behavioural 8 bit adder

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

entity eightbit_ALU1 is
    generic (delay: time := 10 ns);
    Port ( a : in std_logic_vector(7 downto 0);
          b : in std_logic_vector(7 downto 0);
          c_in : in std_logic;
          opcode : in std_logic_vector(1 downto 0);
          result : out std_logic_vector(7 downto 0);
          c_out : out std_logic);
end eightbit_ALU1;

architecture Behavioral of eightbit_ALU1 is
begin

process (a, b, c_in, opcode)
variable tmp_result:std_logic_vector(7 downto 0);
variable tmp_carry: std_logic_vector(8 downto 0);
begin

   tmp_carry(0) := c_in;

   for i in 0 to 7 loop
      tmp_carry(i+1) := (((not tmp_carry(i)) and (a(i) and b(i))) or (tmp_carry(i)
      and (a(i) or b(i))));
   end loop;

   case opcode is
   when "00" =>
      tmp_result := (a and b);
   when "01" =>
      tmp_result := (a or b);
   when "10" =>
      tmp_result := a + b + c_in ;
   when others =>
      tmp_result := (others =>'X');
   end case;

   result <= tmp_result after delay;
   c_out <= tmp_carry(8) after delay;
end process;

end Behavioral;
```