The University of Alabama in Huntsville
ECE Department
CPE 426 01
Midterm Solution
Spring 2013

1.. (1 point) A(n) **_entity, architecture, package, configuration__** is an example of a design unit.

2. (1 point) A **_port_** is a signal used in describing the interface of a VHDL model.

3. (10 points) Create a VHDL architecture representing a structural model of an 8-bit odd-parity checker using the entity given and instances of the exclusive-or gate entity given.

```
entity ODD_PARITY_8 is
  port ( I : in std_logic (7 downto 0);
         P : out std_logic) ;
end entity ODD_PARITY_8;

entity XOR2 is
  port ( A, B : in std_logic;
         F : out std_logic);
end entity XOR;
```

The logic equation describing the parity checker is

$$P = ((I0 \oplus I1) \oplus (I2 \oplus I3)) \oplus ((I4 \oplus I5) \oplus (I6 \oplus I7))$$

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

architecture STRUCT of ODD_PARITY_8 is
  signal TEMP : std_logic_vector ( 5 downto 0);
begin
  U1 : entity XOR2 port map (I(0), I(1), TEMP(0));
  U2 : entity XOR2 port map (I(2), I(3), TEMP(1));
  U3 : entity XOR2 port map (I(4), I(5), TEMP(2));
  U4 : entity XOR2 port map (I(6), I(7), TEMP(3));
  U5 : entity XOR2 port map (TEMP(0), TEMP(1), TEMP(4));
  U6 : entity.XOR2 port map (TEMP(2), TEMP(3), TEMP(5));
  U7 : entity XOR2 port map (TEMP(4), TEMP(5), P);
end STRUCT;
```

4. (7 points) Write the equivalent process for the conditional signal assignment statement

```
mux_logic:
  z <= a and not b after 5 ns when enable and not sel else
       x or y after 6 ns when enable and sel else
      '0' after 4 ns;
```

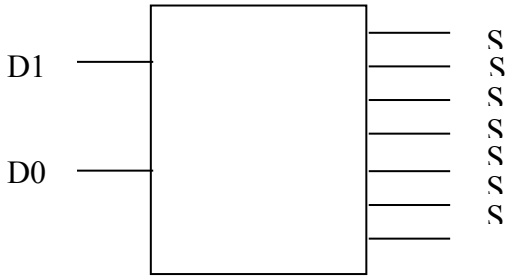
```
library ieee;
use ieee.std_logic_1164.all;

entity MUX is
end entity MUX;

architecture PROCESS_EQUIV of MUX is
  signal a, b, enable, sel, x, y, z : std_logic;
begin
  process (a, b, enable, sel, x, y) is
  begin
    if (enable and not sel) then
      z <= a and not b after 5 ns;
    elsif (enable and sel) then
      z <= x or y after 6 ns;
    else
      z <= '0' after 4 ns;
    end if;
  end process;
end architecture;
```

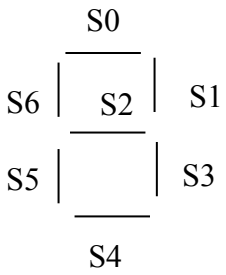
5. (15 points) (a) (10 points) Write a VHDL procedure called align_address that aligns a binary encoded address in a bit-vector variable parameter. The procedure has a second parameter that indicates the alignment size. If the size is 1, the address is unchanged. If the size is 2, the address is rounded to a multiple of 2 by clearing the least significant bit. If the size is 4, two bits are cleared, and if the size is 8, three bits are cleared. The default alignment size is 4. (b)(5 points) Show an architecture that includes two calls to the function with the following properties. 1 - returns a bit_vector with size of 1 length, 2 – passes only an address, not a size.

```
package MINE is
  procedure ALIGN_ADDRESS ( ADDRESS : inout bit_vector;
                            SIZE : in integer := 4);
end package MINE;

package body MINE is
  procedure ALIGN_ADDRESS ( ADDRESS : inout bit_vector;
                            SIZE : in integer := 4) is
  begin
    if (size = 2) then
      ADDRESS := ADDRESS (ADDRESS'length - 1  downto 1) & '0';
    elsif (size = 4) then
      ADDRESS := ADDRESS (ADDRESS'length - 1 downto 2) & "00";
    elsif (size = 8) then
      ADDRESS := ADDRESS (ADDRESS'length - 1 downto 3) & "000";
    end if;
  end procedure;
end package body;

use work.MINE.all;
entity ALIGN_ADDRESS_TEST is
end entity ALIGN_ADDRESS_TEST;
```

```
architecture IT of ALIGN_ADDRESS_TEST is
  signal A : bit_vector(11 downto 0);
  signal B : bit_vector(11 downto 0);
begin
  process
    variable VA : bit_vector(11 downto 0) := "100011101111";
    variable VB : bit_vector(11 downto 0) := "100011101111";
  begin
    ALIGN_ADDRESS (VA, 1);
    A <= VA;
    ALIGN_ADDRESS (VB);
    B <= VB;
    wait;
  end process;
end architecture;
```

6. (8 points) Consider the following combinational digital system, called a light-emitting diode (LED) driver. The LED driver converts a 2-bit binary number (D_1D_0) into an LED-displayed numeral. For example, $D_1D_0 - 10_2$ is displayed by asserting S_0, S_1, S_2, S_4, and S_5.






```
entity LED_DRIVER is
  port (D : in bit_vector (1 downto 0);
        S : out bit_vector(6 downto 0));
end entity LED_DRIVER;
```

Use concurrent signal assignments to model the LED driver.

```
entity LED_DRIVER is
  port ( D : in bit_vector (1 downto 0);
         S : out bit_vector (6 downto 0));
end entity LED_DRIVER;

architecture CONCURRENT of LED_DRIVER is
begin
  S <= "1111011" when D = "00" else
       "0001010" when D = "01" else
       "0110111" when D = "10" else
       "0011111" when D = "11";
end CONCURRENT;
```

7. (1 point) 'RANGE is an example of a VHDL _**attribute**_.

8. (1 point) VHDL is a strongly typed language _**True**__ (True/False)

9. (2 points) Create a COLLEGE enumeration data type that has the values of the colleges at UAH.

```
type college is (BUSINESS, ENGINEERING, LIBERAL_ARTS, NURSING, SCIENCE);
```

10. (1 point) A function is a primary design unit. (True/False) _**False**_.

11. (3 points) Write a declaration of a two-dimensional table, TABLE_2D, with index values and table entries all of type bit (which has been declared elsewhere and is visible). Initialize all elements of the array to '1';

```
variable TABLE_2D is (bit, bit) of bit := (others => '1');
```

12. (20 points) Given the following VHDL, indicate all transactions and events. Give the values of A, B, C, D, E, and F each time a change occurs. Carry this out until no further change occurs.

```
entity prob is
  port (D : inout bit);
end prob;
architecture PROB of PROB is
  signal A, B, C, E, F : bit;
begin
  process
    A <= '1' after 5 ns,
         '0' after 12 ns;
    wait;
  end process;
  P1: process (A, C)
  begin
    B <= A after 2 ns;
    E <= C after 7 ns;
  end process P1;
  C1: C <= A and B after 6 ns;
  P2: process (C, E)
  begin
    F <= C and E after 4 ns;
  end process P2;
  C2: D <= A or B or C or F;
end PROB;
```

Time	A	B	C	D	E	F
0 ns	0	0	0	0	0	0
5 ns	1	0	0	0	0	0
5+Δ ns	**1**	**0**	**0**	**1**	**0**	**0**
7 ns	**1**	**1**	**0**	**1**	**0**	**0**
12 ns	**0**	**1**	**0**	**1**	**0**	**0**
14 ns	**0**	**0**	**0**	**1**	**0**	**0**
14+Δ ns	**0**	**0**	**0**	**0**	**0**	**0**

Time	Event	Processes Triggered	Scheduled Transactions	Event?
5 ns	A → 1	P1	B, '1' at 7 ns	Yes
		P1	E, '0', at 12 ns	No
		C1	C 0 at 11 ns	No
		C2	D, '1' at 5 + Δ ns	Yes
5 + Δ ns	D → 1	None		
7 ns	B → 1	C1	C, '1', 13 ns	Yes
		C2	D, '1', 7 + Δ ns	No
12 ns	A → 0	P1	B, '0', 14 ns	Yes
		P1	E, '0', 19 ns	No
		C1	C, '0', 16 ns	No, overwrite C, '1', 13 ns
		C2	D, '1', 12 + D ns	No
14 ns	B → 0	C1	C, '0', 20 ns	No
		C2	D, '0', 14 + Δ ns	Yes
14 + Δ ns	D → 0	None		

New before existing	Overwrite existing	Overwrite existing
New after existing	Append new	If $v_{new} = v_{existing}$, append new Elsif $t_{new}-t_{existing} >$ reject append new Else overwrite existing

13. (15 points) Develop a behavioral model for a D-latch with tristate output. The entity declaration is

```
entity d_latch is
  port ( latch_en, out_en, d : in std_logic;
         q : out std_logic );
end entity d_latch;
```

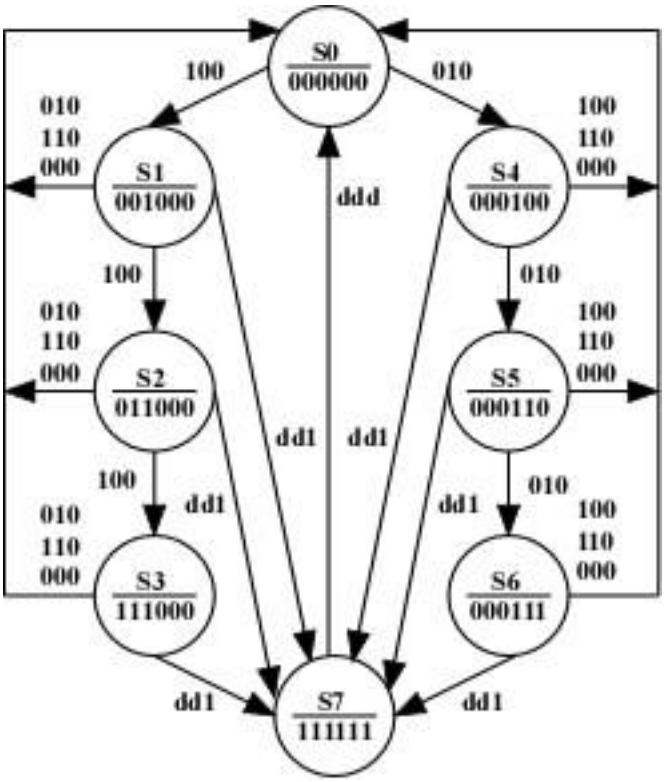
When `latch_en` is asserted, data from the `d` input enters the latch. When `latch_en` is negated, the latch maintains the stored value. When `out_en` is asserted, data passes through to the output. When `out_en` is negated, the output has the value 'Z' (high-impedance). The propagation delay from `latch_en` to `q` is 3 ns and from `d` to `q` is 4 ns. The delay from `out_en` asserted to `q` active is 2 ns and from `out_en` negated to `q` high-impedance is 5 ns.

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

entity D_LATCH is
  generic ( TPH : time := 3 ns);
  port ( LATCH_EN, OUT_EN, D : in std_logic;
         Q : out std_logic);
end entity D_LATCH;

architecture TIMING of D_LATCH is
  signal TEMP, LATCH_EN_DELAYED : std_logic;
begin
  LATCH_EN_DELAYED <= LATCH_EN'DELAYED(TPH);
  process (LATCH_EN_DELAYED)
  begin
    if (LATCH_EN_DELAYED = '1') then
      assert D'stable (TPH)
      report "Hold time violated"
      severity warning;
    end if;
  end process;

  process (D,  LATCH_EN)
  begin
    if (D'event and LATCH_EN = '1') then
      TEMP <= D after 4 ns;
    elsif (LATCH_EN'event and LATCH_EN = '1') then
      TEMP <= D after 3 ns;
    end if;
  end process;

  process ( TEMP, OUT_EN )
  begin
    if OUT_EN then
      Q <= TEMP after 2 ns;
    else
      Q <= 'Z' after 5 ns;
    end if;
  end process;
end TIMING;
```

14. (15 points) Draw the state diagram for the following state machine. Is it a Moore machine or a Mealy machine?

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

entity THUNDERBIRD is
  port (L, R, H, CLK : in std_logic;
        LC, LB, LA, RA, RB, RC : out std_logic);
end THUNDERBIRD;

architecture BEHAVE of THUNDERBIRD is
  type STATE_TYPE is (S0, S1, S2, S3, S4, S5, S6, S7);
  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
begin
  process(CURRENT_STATE, L, R, H)
    variable INPUTS : std_logic_vector(2 downto 0);
  begin
    INPUTS := L&R&H;
    case CURRENT_STATE is
      when S0 => if (INPUTS = "000") then
                   NEXT_STATE <= S0;
                 elsif (INPUTS = "100") then
                   NEXT_STATE <= S1;
                 elsif (INPUTS = "010") then
                   NEXT_STATE <= S4;
                 else
                   NEXT_STATE <= S7;
                 end if;
      when S1 => if (INPUTS(0) = '1') then
                   NEXT_STATE <= S7;
                 elsif (INPUTS = "100") then
                   NEXT_STATE <= S2;
                 else
                   NEXT_STATE <= S0;
                 end if;
      when S2 => if (INPUTS(0) = '1') then
                   NEXT_STATE <= S7;
                 elsif (INPUTS = "100") then
                   NEXT_STATE <= S3;
                 else
                   NEXT_STATE <= S0;
                 end if;
       when S3|S6 => if (INPUTS(0) = '1') then
                   NEXT_STATE <= S7;
                 else
                   NEXT_STATE <= S0;
                 end if;
       when S4 => if (INPUTS(0) = '1') then
                   NEXT_STATE <= S7;
                 elsif (INPUTS = "010") then
                   NEXT_STATE <= S5;
                 else
                   NEXT_STATE <= S0;
                 end if;
       when S5 => if (INPUTS(0) = '1') then
                   NEXT_STATE <= S7;
                 elsif (INPUTS = "010") then
                   NEXT_STATE <= S6;
                 else
                   NEXT_STATE <= S0;
                 end if;
      when S7 => NEXT_STATE <= S0;
     end case;
   end process;
```

```
  process (CLK)
  begin
    if (CLK'event and CLK = '1') then
      CURRENT_STATE <= NEXT_STATE;
    end if;
  end process;
  process(CURRENT_STATE)
  begin
    LC <= '0'; LB <= '0'; LA <= '0';
    RA <= '0'; RB <= '0'; RC <= '0';
    case CURRENT_STATE is
      when S0 => null;
      when S1 => LA <= '1';
      when S2 => LA <= '1'; LB <= '1';
      when S3 => LA <= '1'; LB <= '1'; LC <= '1';
      when S4 => RA <= '1';
      when S5 => RA <= '1'; RB <= '1';
      when S6 => RA <= '1'; RB <= '1'; RC <= '1';
      when S7 => LA <= '1'; LB <= '1'; LC <= '1';
                 RA <= '1'; RB <= '1'; RC <= '1';
    end case;
  end process;
end BEHAVE;
```



Moore, outputs depend only on the state.