The University of Alabama in Huntsville
ECE Department
CPE 526 01
Midterm Exam Solution
Spring 2016

1. (15 points) Write a VHDL function that accepts a std_logic_vector of arbitrary length and an integer that specifies the number of bits the std_logic_vector is to be rotated to the left and returns the rotated std_logic_vector. Issue an error message if the integer is greater than the length of the input. Make no assumptions about the range of the indices. For example:

   Input: 0101111, 2
   Output: 0111101

```
function rotate (invect : std_logic_vector; n : integer) return
std_logic_vector is
    variable temp : std_logic_vector(invect'length-1 downto 0);
  begin
    if (n > invect'length) then
      report "Error, can't rotate that much"
      severity warning;
      return invect;
    end if;
    temp := invect;
    for i in n downto 1 loop
      temp := temp(invect'length-2 downto 0) & temp(invect'length-1);
    end loop;
    return temp;
  end function rotate;
```

2. (10 points) . (6 points) Translate the following VHDL to two with-select-when statements:

```
case state is
  when idle => a <= "11"; b <= "00";
  when terminate | increase => a <= "01"; b <= "--";
  when maintain | decrease => a <= "10"; b <= "11";
  when others => a <= "11"; b <= "01";
end case;
```

```
with state select
    a <= "11" when idle,
         "01" when terminate | increase,
         "10" when maintain | decrease,
         "11" when others;
  with state select
    b <= "00" when idle,
         "--" when terminate | increase,
         "11" when maintain | decrease,
         "01" when others;
```

3. (1 point) All statements inside of a process are _**sequential**_

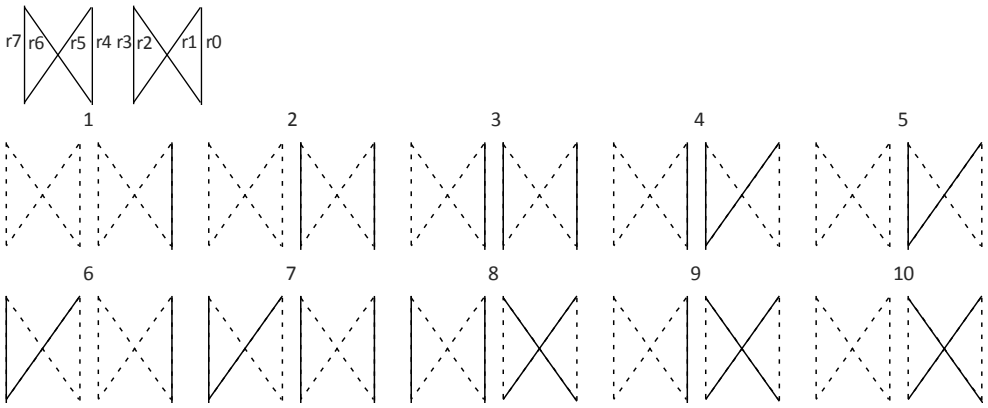
4. (1 point) _**std_logic_vector, bit_vector, etc.**_ is an example of an unconstrained array.

5. (1 point) _**False**_ (True or False)Multiple assignments to a signal within a process can cause that signal to have multiple drivers.

6. (1 point) _**False**_ (True or False) A process may have both a sensitivity list and wait statements

7. (1 point) _**False**_ (True or False) Functions are primary design units.

8. (15 points) Design a system which has as its inputs a number from 1 to 10 and provides as its outputs (eight of them) the signals to drive the following display. The inputs are labeled w, x, y, and z and are normal binary. The input combinations 0000, 1011, 1100, 1101, 1110, and 1111 will never occur; they are to be treated as don't cares. The display allows for the representation of Roman numerals (except that IIX is used to represent 8, whereas it is normally written as VIII). There are a total of eight segments in the display, labeled A through H, as shown. To light a segment, a 1 is placed on the appropriate display input (r7, r6, r5, r4, r3, r2, r1, r0). The following illustration shows all digits as they should be coded for each of these, with a lit segment represented by a bold line and an unlit segment represented by a dashed line.

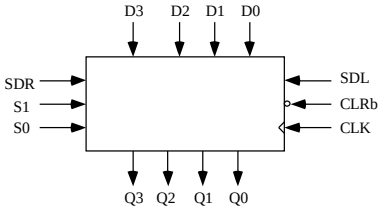


Use the following entity

```
entity roman is
  port (d : in std_logic_vector (3 downto 0);
        r : out std_logic_vector (7 downto 0));
end entity roman;

architecture behave of roman is
begin
  r <= "00000001" when d = "0001" else
       "00001001" when d = "0010" else
       "00011001" when d = "0011" else
       "00011010" when d = "0100" else
       "00001010" when d = "0101" else
       "10100001" when d = "0110" else
       "10101001" when d = "0111" else
       "10010110" when d = "1000" else
       "00010110" when d = "1001" else
       "00000110" when d = "1010" else
       "00000000";
end behave;
```

9. (18 points) A description of a 74194 4-bit bi-directional shift register follows: The CLRb input is asynchronous and active low and overrides all the other control inputs. All other state changes occur following the rising edge of the clock. If the control inputs S1 = S0 = 1, the register is loaded in parallel. If S1 = 1 and S0 = 0, the register is shifted right and SDR (serial data right) is shifted into Q3. IF S1 = 0 and S0 = 1, the register is shifted left and SDL is shifted into Q0. If S1 = S0 = 0, no action occurs. Use 74194s to build an 8-bit bi-directional shift register that has 3 select lines. The operation of the register is specified by the table below.




```
entity shifter is
  port (x : in std_logic_vector (7 downto 0);
        y : out std_logic_vector (7 downto 0);
        r, l, clrb : in std_logic;
        clock : in std_logic;
        sel : in std_logic_vector (2 downto 0));
end entity shifter;
```

sel	Operation
000	No Change
001	Shift Left serial input
010	Shift Right, serial input
011	Load
100	No Change
101	Rotate Left
110	Rotate Right
111	Load Zeros

```
architecture STRUCTURE of SHIFTER is
  signal TEMP0, TEMP1 : std_logic_vector(7 downto 0);
  signal t0, t1 : std_logic;
begin
  U1 : entity S74194 port map(CLRB => CLRB, CLK => CLOCK,
                        S1 => Sel(1), S0 => sel(0),
                        SDR => t0 , SDL => TEMP1(3),
                        D => TEMP0(7 downto 4),
                        Q => TEMP1(7 downto 4));
  U2 : entity S74194 port map(CLRB => CLRB, CLK => CLOCK,
                        S1 => sel(1), S0 => sel(0),
                        SDR => TEMP1(4), SDL => t1,
                        D => TEMP0(3 downto 0),
                        Q => TEMP1(3 downto 0));
  t0 <= R when sel = "010" else TEMP1(0);
  t1 <= L when sel = "001" else TEMP1(7);
  TEMP0 <= "00000000" when sel = "111" else x;
  y <= TEMP1;
end STRUCTURE;
```

10. (17 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 10 ns;
    wait;
  end process;
  P1: process (A, C)
  begin
    B <= A after 2 ns;
    E <= transport C after 5 ns;
  end process P1;
  C1: C <= transport A and B after 6 ns;
  P2: process (C, E)
  begin
    F <= reject 3 ns inertial C and E
         after 5 ns;
  end process P2;
  C2: D <= A or B or C or F after 2 ns;
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
7 ns	**1**	**1**	**0**	**1**	**0**	**0**
10 ns	**0**	**1**	**0**	**1**	**0**	**0**
12 ns	**0**	**0**	**0**	**1**	**0**	**0**
13 ns	**0**	**0**	**1**	**1**	**0**	**0**
16 ns	**0**	**0**	**0**	**1**	**0**	**0**
18 ns	**0**	**0**	**0**	**0**	**1**	**0**
21 ns	**0**	**0**	**0**	**0**	**0**	**0**

Time	Event	Processes Triggered	Scheduled Transactions	Event?
5 ns	**A → '1'**	**P1**	**B ('1', 7ns)**	**Y**
			E ('0', 10ns)	**N**
		C1	**C ('0, 11ns)**	**N**
		C2	**C ('1', 7ns)**	**Y**
7 ns	**B → '1'**	**C2**	**D ('1', 9ns)**	**N**
		C1	**C ('1', 13ns)**	**Y**
	D → '1'	**none**		
10 ns	**A → '0'**	**P1**	**B ('0', 12ns)**	**Y**
			E ('0', 15ns)	**N**
		C1	**C ('0', 16ns) append**	**Y**
		C2	**D ('1', 12ns)**	**N**
12 ns	**B → '0'**	**C2**	**D ('0', 14ns)**	**Y**
		C1	**C ('0', 18ns)**	**N**
	D → '1'	**none**		
13 ns	**C → '1'**	**P1**	**B ('0', 15ns)**	**N**
			E ('1', 18ns)	**Y**
		P2	**F ('0', 18ns)**	**N**
		C2	**D ('1', 15ns) overwrite (0,14)**	**N**
16 ns	**C → '0'**	**P1**	**B ('0', 18ns)**	**N**
			E ('0', 21ns) append	**Y**
		P2	**F ('0', 21ns) append**	**N**
		C2	**D ('0', 18ns)**	**Y**
18 ns	**E → '1'**	**P2**	**F ('0', 23ns)**	**N**
	D → '0'	**none**		
21 ns	**E → '0'**	**P2**	**F ('0', 26ns)**	**N**

Scheduling Rules	Transport	Inertial
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

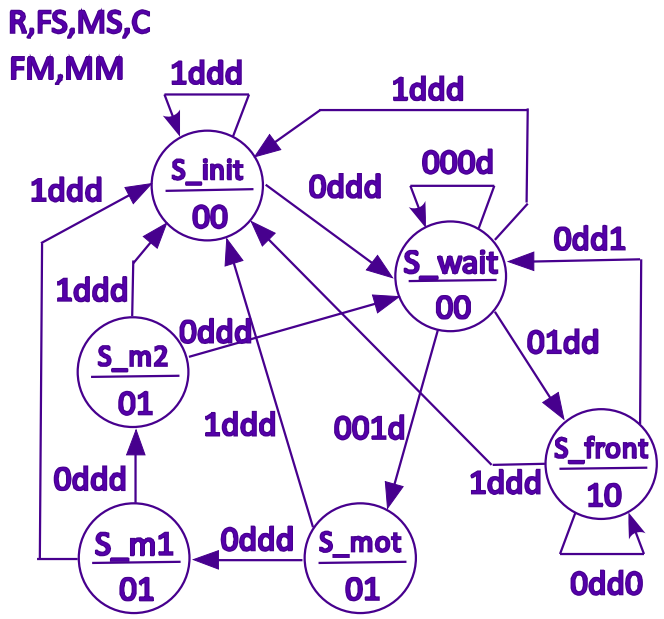
11. (10 points) Draw the state diagram for the following state machine. Is it a Moore machine or a Mealy machine?

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

entity ALARM is
  port (CLK, FS, MS, R, C : in std_logic;
        FM, MM : out std_logic);
end ALARM;

architecture SYNTH of ALARM is
  type STATE_TYPE is (S_init, S_wait, S_front,
                      S_motion, S_MM1, S_MM2);
  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
begin
  process(CLK, R)
  begin
    if (R = '1') then
      CURRENT_STATE <= S_init;
    elsif (CLK'event and CLK = '1') then
      CURRENT_STATE <= NEXT_STATE;
    end if;
  end process;
  process(FS, MS, C, CURRENT_STATE)
  begin
    case CURRENT_STATE is
      when S_init => NEXT_STATE <= S_wait;
      when S_wait => if (FS = '1') then
                       NEXT_STATE <= S_front;
                     elsif (MS = '1') then
                       NEXT_STATE <= S_motion;
                     end if;
      when S_front => if (C = '1') then
                        NEXT_STATE <= S_wait;
                      else
                        NEXT_STATE <= S_front;
                      end if;
      when S_motion => NEXT_STATE <= S_MM1;
      when S_MM1 => NEXT_STATE <= S_MM2;
      when S_MM2 => NEXT_STATE <= S_wait;
    end case;
  end process;
  process (CURRENT_STATE)
  begin
    case CURRENT_STATE is
      when S_init | S_wait => FM <= '0'; MM <= '0';
      when S_front => FM <= '1'; MM <= '0';
      when S_motion => FM <= '0'; MM <= '1';
      when S_MM1 | S_MM2 => FM <= '0'; MM <= '1';
    end case;
  end process;
end SYNTH;
```

Moore Machine



12. (10 points) Write a VHDL entity (3 points) and architecture (7 points) of a two-input OR gate with the generics, TPLH and TPHL, which reflect the time for the output to make a low to high or high to low transition, respectively.

```
entity OR_DEL is
  generic (TPLH, TPHL : time := 2 ns);
  port (x, y : in std_logic;
        f    : out std_logic);
end OR_DEL;

architecture BEHAV of OR_DEL is
begin
  process (x,y)
    variable current, old : std_logic;
  begin
    current := x or y;
    if (current = '1' and old /= '1') then
      f <= x or y after TPLH;
    elsif (current = '0' and old /= '0') then
      f <= x or y after TPHL;
    end if;
    old := current;
  end process;
end BEHAV;
```