# EL 310 Hardware Description Languages – Midterm

| 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|
| | | | | | |

Name:

ID :

**Notes:**

**1) Please answer the questions in the provided space after each question.**

**2) Duration is 110 minutes**

**3) Closed books and closed notes.**

**Question 1 – VHDL (10 pts)**

**i)** Write down a signal assignment statement (using the concatenation operator - &) that creates

a. a **rotate right** operation by **seven** bit positions (**1 pts**)

b. a **logic shift right** by **four** bit positions (**1 pts**)

c. an **arithmetic shift right** by **three** bit positions (**1 pts**)

on the following signal:

```
SIGNAL s : std_logic_vector(15 DOWNTO 0);
```

a. `s(15 downto 0) <= s(6 downto 0) & s(15 downto 7);`

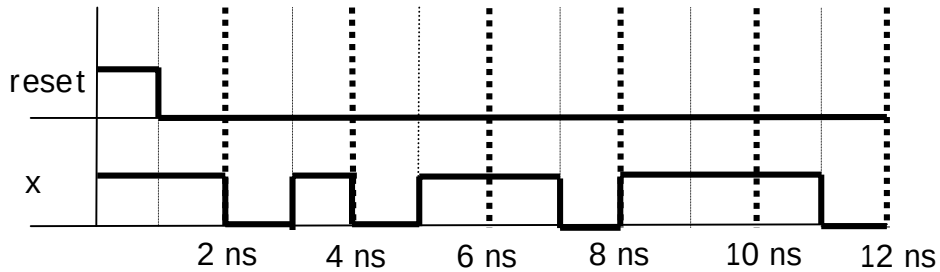
b. `s(15 downto 0) <= “0000” & s(15 downto 4);`

c. `s(15 downto 0) <= s(15)&s(15)&s(15)&s(15 downto 3);`

**ii)** Circle the statement type (**4 pts**)

```
a <= b;                          sequential   concurrent   either
mux: mux2to1
  port map(a=>bi b=>c);          sequential   concurrent   either
happy<='1' when(exam=easy)       sequential   concurrent   either
           else '0';
a:=b and c;                      sequential   concurrent   either
```

**iii)** Generate the following *repeating* waveform x and non-repeating reset signal using VHDL constructs. (**3pts**)

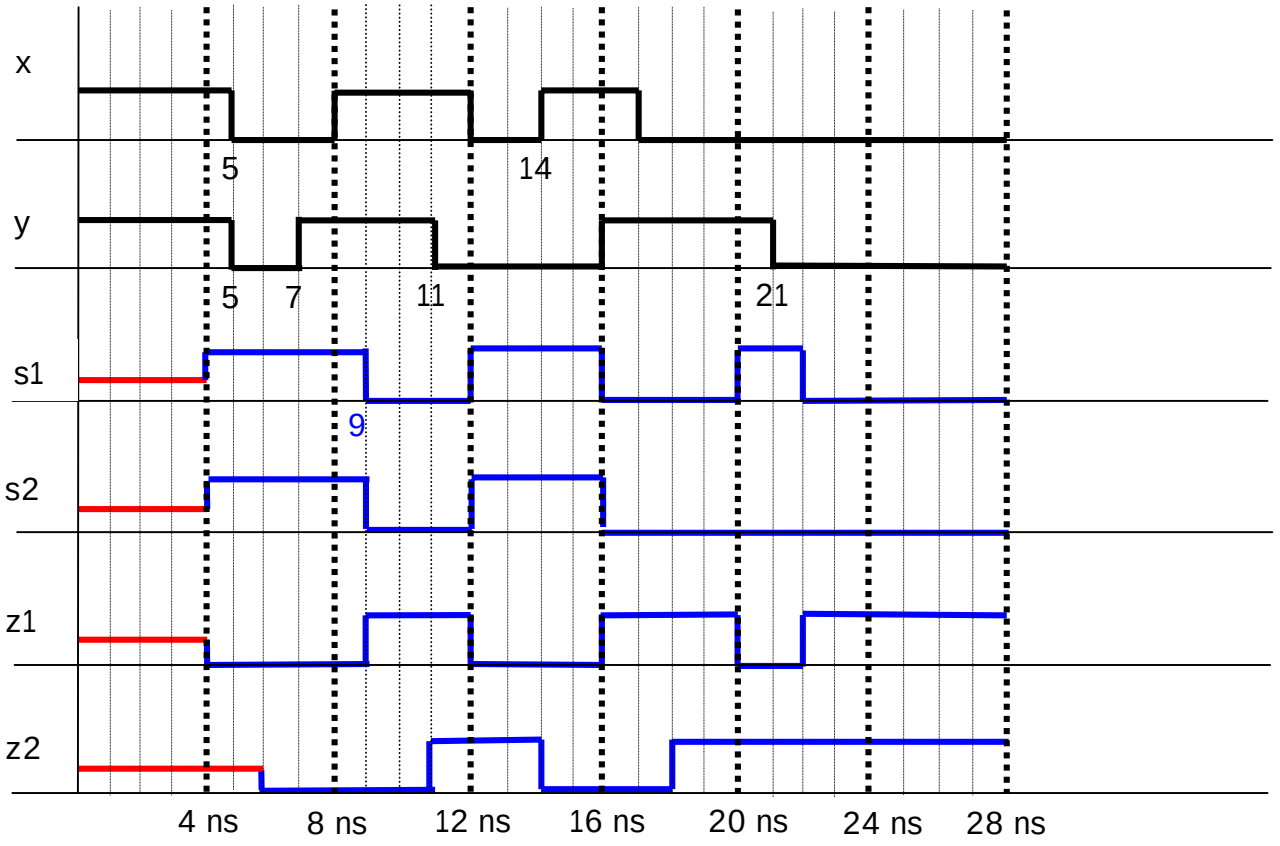


```
architecture beh of two-signal is
begin
  reset_process: reset <= ‘1’, ‘0’ after 10 ns;
  x_process: process
  begin
    x <= ‘1’, ‘0’ after 2 ns, ‘1’ after 3 ns, ‘0’ after 4 ns,
         ‘1’ after 5 ns, ‘0’ after 7 ns, ‘1’ after 8 ns,
         ‘0’ after 11 ns;
    wait for 12 ns;
  end process x_process;
end architecture;
```

**Question 2 – Delays (15 pts)**

Write the two lines in VHDL to generate input signals x and y. Complete the timing diagram of s1, s2, z1, and z2 for given input signals x and y. The output signals are derived from input signals x and y as follows:

```
x <= '1', '0' after 5 ns, '1' after 8 ns,
     '0' after 12 ns,'1' after 14 ns, '0' after 17 ns;
y <= '1', '0' after 5 ns, '1' after 7 ns, '0' after 11 ns,
     '1' after 16 ns, '0' after 21 ns;
s1 <= transport x and y after 4 ns;
s2 <= reject 2 ns inertial (x and y) after 4 ns;
z1 <= s1 nor s2;
z2 <= s1 nor s2 after 2 ns;
```

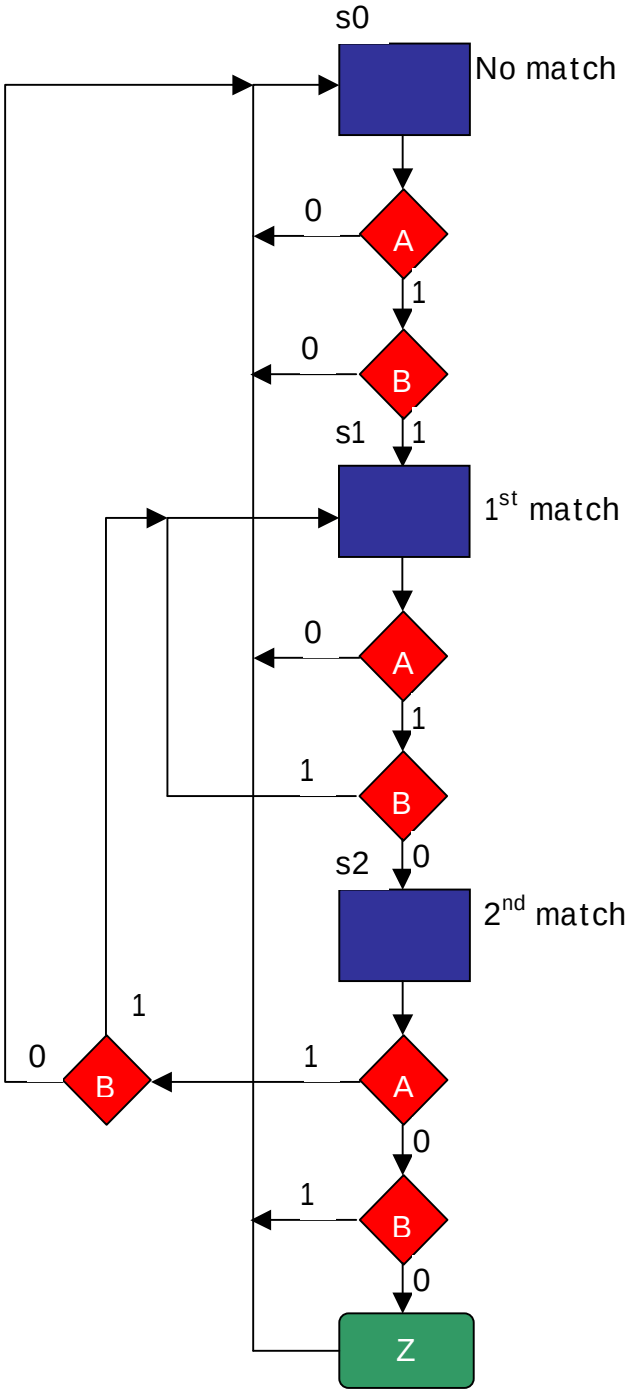


**Question 3 – Design with ASM Charts (20 pts)**

A state machine has two inputs, A and B, and one output, Z. If the sequence of input pairs

A = 1, B = 1 (1st pair)
A = 1, B = 0 (2nd pair)
A = 0, B = 0 (3rd pair)

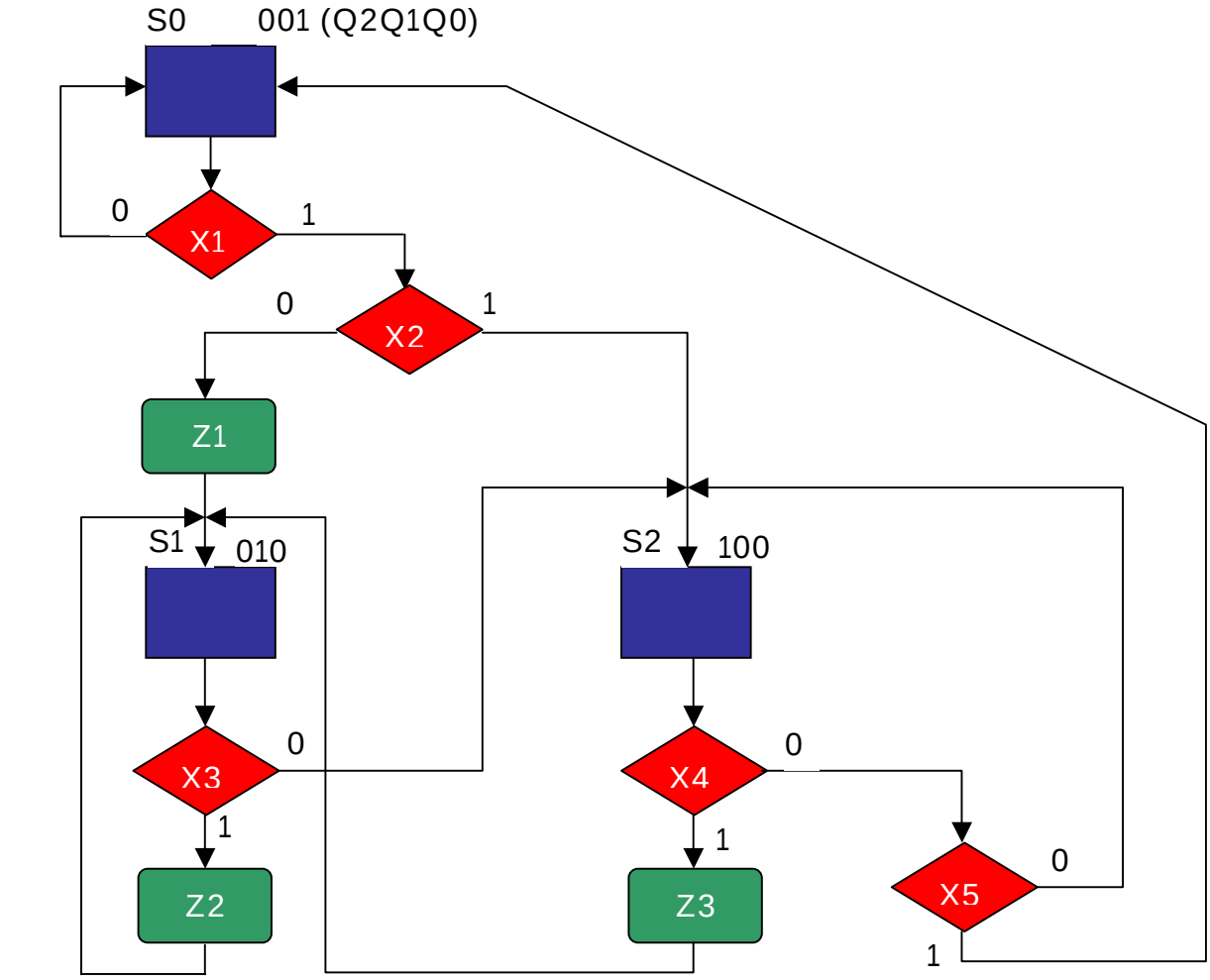
is detected, Z is asserted during the final cycle of the sequence, otherwise output remains at 0. Draw the ASM chart first, and utilize it to write a two-process VHDL model of a state machine to implement this system. Choose one-hot state encoding for the state encoding.



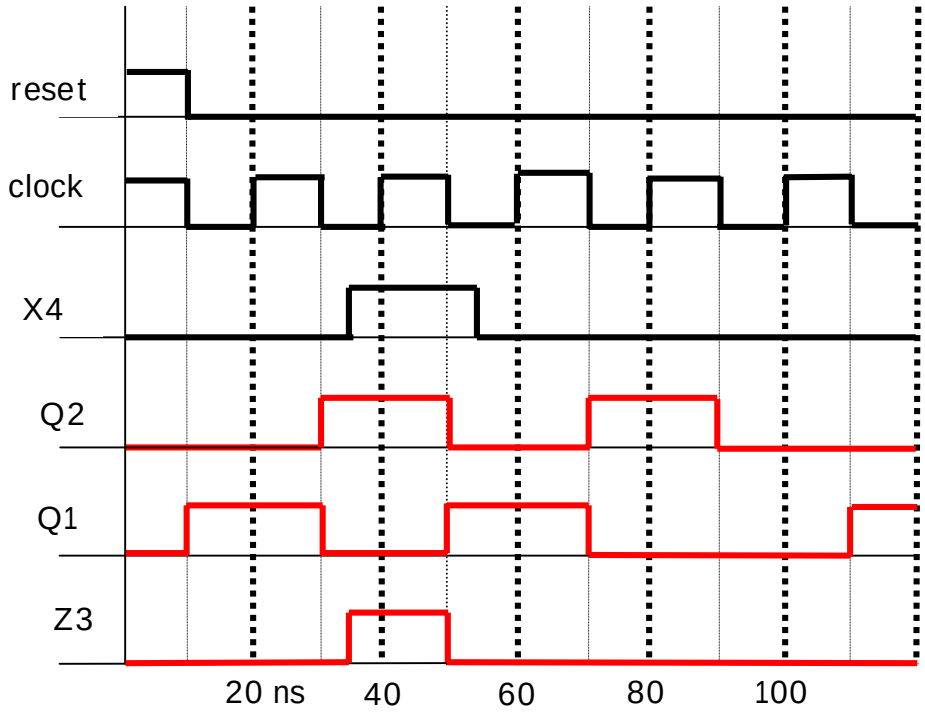
```
entity asm02 is
port(clock, reset, A, B: in std_logic; Z: out std_logic);
end entity asm02;

architecture beh of asm02 is
  signal state, next_state: std_logic_vector(2 downto 0);
  constant s0: std_logic_vector(2 downto 0) := "001"I
  constant s1: std_logic_vector(2 downto 0) := "010";
  constant s2: std_logic_vector(2 downto 0) := "100";
begin

clk_process: process(clock) is
begin
  if (reset = '1' ) then state <= s0;
  else
    if (falling_edge(clock)) then state <= next_state;
    end if;
  end if;
end process;

com_process: process(state, A, B) is
begin
  Z <= '0';
  next_state <= s0;
  case state is
    when s0 =>
      if A = '0' then next_state <= s0;
      else
        if B = '0' then next_state <= s0;
        else next_state <= s1;
        end if;
      end if;
    when s1 =>
      if A = '0' then next_state <= s0;
      else
        if B = '1' then next_state <= s1;
        else next_state <= s2;
        end if;
      end if;
    when s2 =>
      if A = ‘1’ then
        if B = ‘1’ then next_state <= s1;
        else next_state <= s0;
      else
        next_state <= s0;
        if B = '0' then Z <= '1';
      end if;
    when others => null;
  end case;
end process;
end architecture;
```

**Question 4 – Timing in ASM Charts (20 pts)**

For the given ASM chart:




Complete the following timing diagram (assume that X1=1, X2=0, X3=0, X5=1, and X4 is as shown). Flip-flops change state on the falling edge of clock.



**Question 5 – Counter Designs (35 pts)**

**i)** Binary counter: Write a VDHL code for binary up/down counter. It must have an asynchronous reset to initialize it to 0. In addition, the range of the counter and whether it is counting up or down must be parameterized. It must have two outputs: one indicating current counting value (count) and another output (z) that must be asserted when the counter reaches its upper or lower limit. Ensure that no latch is inferred for z. When the counter reaches this limit, it must wrap around and the next value will be zero. (**10 pts**)

```
entity updown is
generic(n:integer:=4; up:integer:=-1);
port(clk, reset:  in std_logic;
     count: out integer range -n+1 to n-1; z: out std_logic);
end entity updown;
architecture beh of updown is
begin
process(clk, reset)
  variable cnt: integer range -n+1 to n-1;
begin
  z <= '0';
  if reset = '1' then
    cnt := 0;
  elsif(rising_edge(clk)) then
    if cnt = n-1 then cnt := 0; z <= '1';
    elsif cnt = -n+1 then cnt := 0; z <= '1';
    else cnt := cnt + up;
   end if;
  end if;
count <= cnt;
end process;
end architecture beh;
```

**ii)** Möbius counter: Write a VHDL code for Möbius (or Johnson) counter which is counting in a different way: the least significant value of the counter is inverted and fed back to the most significant bit in the next clock cycle. All the other bits are shifted one bit to the right. A normal counting sequence of a Möbius counter must include all zeros string. The number of bits (say n) must be parameterized. Calculate the number of states in Möbius counter and give the formula for number of states in terms of n. What is the advantage of Möbius counter? (**10 pts**)

```
architecture beh of mobius is
begin
process(clk, reset)
  variable reg: std_logic_vector(n-1 downto 0);
begin
  if reset = '1' then
    reg := (when others => '0');
  elsif(rising_edge(clk)) then
    reg := not reg(0) & reg(n-1 downto 1);
  end if;
  count <= reg;
end process;
end architecture beh;
```

It is a very simple synchronous counter. It takes little space.

**iii)** Self-correcting Möbius counter: Möbius counter has many unused state that it never enters in the normal circumstances. Because of these unused states, there are many other autonomous counters. If the counter enters one of these unused states as a result of a parasitic influence, such as the one of a power supply glitch, the counter can never return to its normal sequence. An example for this phenomenon is given as follows:

| Normal counting sequence | Parasitic counting sequence |
|---|---|
| 0000 | 0010 (an error occurs in the 2nd bit) |
| 1000 | 1001 |
| 1100 | 0100 |
| 1110 | 1010 |
| 1111 | 1101 |
| 0111 | 0111 |
| 0011 | 0110 |
| 0001 | 0101 |

Whatever the size of n, the unused states form a single parasitic counter. Propose a practical solution to make the counter self-correcting. In other words, when the counter enters one of the illegal states, it must be able to return to its normal counting sequence eventually. Write a VHDL program implementing your solution. (**15 pts**)

The only legal state that has 0 in both the most significant and the least significant bits is the all zeros state. On the other hand, three of the parasitic states have zeros in those positions. The solution is that whenever the counter enters a state that has zeros in those positions, the next state is set '1000'.

```
architecture beh of self_correcting_mobius is
begin
process(clk, reset)
  variable reg: std_logic_vector(n-1 downto 0);
begin
  if reset = '1' then
    reg := (when others => '0');
  elsif(rising_edge(clk)) then
    if reg(n-1) = '0' and reg(0) = '0' then
      reg := (others => '0');
      reg(n-1) := '1';
    else
      reg := not reg(0) & reg(n-1 downto 1);
    end if;
  end if;
  count <= reg;
end process;
end architecture beh;
```

**Appendix**

You can utilize the following template in your answers.

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

entity entity_name is
  generic(param: type := default-value)
  port(input signals: in type;
       output signals: out type);
end entity entity_name;

architecture arch_name of entity_name is

  component component1-name is
  generic(param: type := default-value)
  port (input signals: in type;
       output signals: out type);
  end component component1-name ;

  component component2-name is
  generic(param: type := default-value)
  port (input signals: in type;
        output signals: out type);
  end component component2-name ;

signal internal signals: type:=initialization;

begin
-- label each component and connect its ports to signals or --
-- entity ports
Label1: component1-name port map(port=>signal, ...);
Label2: component2-name port map(port=>signal, ...);

concurrent signal assignment-1;

concurrent signal assignment-2;

process-1:...

process-2:...

end architecture arch_name;
```