EEL 4712: Digital Design

Instructor: Dr. Farimah Farahmandi

Midterm 1 – 10/2/2019

Time: 55 Minutes

Name: …….

UFID: …….

Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
As always, the best answer gets the most points.

By taking this exam, you agree to follow all rules specified below or face receiving a 0% for this exam. We will be implementing a Zero-Strike policy.

- **No electronic devices or printed materials allowed**
- **Communication between students is NOT ALLOWED**
- **Under no circumstances will you share the questions from this test until after the solutions are formally released by the instructor**

Re-Grade Information:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Problem	Points	Score
1	10	
2	10	
3	15	
4	15	
5	15	
6	20	
7	15	
Total	100	

```
ENTITY __entity_name IS
        PORT(__input_name, __input_name        : IN  STD_LOGIC;
             __input_vector_name               : IN  STD_LOGIC_VECTOR(__high downto __low);
             __bidir_name, __bidir_name        : INOUT     STD_LOGIC;
             __output_name, __output_name      : OUT       STD_LOGIC);
END __entity_name;

ARCHITECTURE a OF __entity_name IS
        SIGNAL __signal_name : STD_LOGIC;
        SIGNAL __signal_name : STD_LOGIC;
BEGIN
        -- Process Statement
        -- Concurrent Signal Assignment
        -- Conditional Signal Assignment
        -- Selected Signal Assignment
        -- Component Instantiation Statement
END a;

SIGNAL __signal_name : __type_name;
__instance_name: __component_name   GENERIC MAP (__component_par =>__connect_par)
                                    PORT MAP  (__component_port => __connect_port,
                                               __component_port => __connect_port);

WITH __expression SELECT
        __signal <= __expression WHEN __constant_value,
                    __expression WHEN __constant_value,
                    __expression WHEN __constant_value,
                    __expression WHEN __constant_value;
```

SELECT assignment statement

```
__signal <= __expression WHEN __boolean_expression ELSE
            __expression WHEN __boolean_expression ELSE
            __expression;
```

Conditional assignment statement

```
IF __expression THEN
   __statement;
   __statement;
ELSIF __expression THEN
   __statement;
   __statement;
ELSE
   __statement;
   __statement;
END IF;
```

```
<optional_label>:
        FOR <loop_id> IN <range> LOOP
                -- Sequential Statement(s)
        END LOOP;
```

```
WAIT UNTIL __expression;

CASE __expression IS
        WHEN __constant_value =>
           __statement;
           __statement;
        WHEN __constant_value =>
           __statement;
           __statement;
        WHEN OTHERS =>
           __statement;
           __statement;
END CASE;
```

```
<generate_label>:
        FOR <loop_id> IN <range> GENERATE
                -- Concurrent Statement(s)
        END GENERATE;
```

1) (10 points) Fill in the waveform for the signals “result_add” and “result_sub” in decimal values

```
entity ALU is
  port (
    clk, rst : in std_logic;
    input1 : in std_logic_vector(7 downto 0);
    input2 : in std_logic_vector(7 downto 0);
    result_add, result_sub : out std_logic_vector(7 downto 0);
    overflow_add : out std_logic
  );
end ALU;

architecture BHV of ALU is
  signal temp_Add: std_logic_vector(8 downto 0);

begin
  process(clk, rst)
  variable temp_sub: std_logic_vector(8 downto 0);
  begin
      if(rst = '1') then
            result_add <= "00000000";
            result_sub <= "00000000";
            overflow_add <= '0';
      elsif (clk='1' and clk'event) then
            temp_Add <= std_logic_vector(unsigned("0" & input1) +
            unsigned("0" & input2));
            temp_sub:= std_logic_vector(unsigned("0" & input1) -
            unsigned("0" & input2));
            result_add <= temp_Add(7 downto 0);
            result_sub <= temp_sub(7 downto 0);
            overflow_add <= temp_Add(8);
      end if;
  end process;
end BHV;
```

Input1

150	200	81	30	110	

Input2

100	57	48	20	40	

Result_add

X	X	250	1 (overflow)	129	50

Result_sub

X	50	143	33	10	70

b) Explain the behavior of “result_add’s” value, mentioning when signals get updated.
Signals are updated at the end of process, variables are updated immediately.
Since temp_Add is defined as a signal, we need two cycles to see the correct value of Result_add.

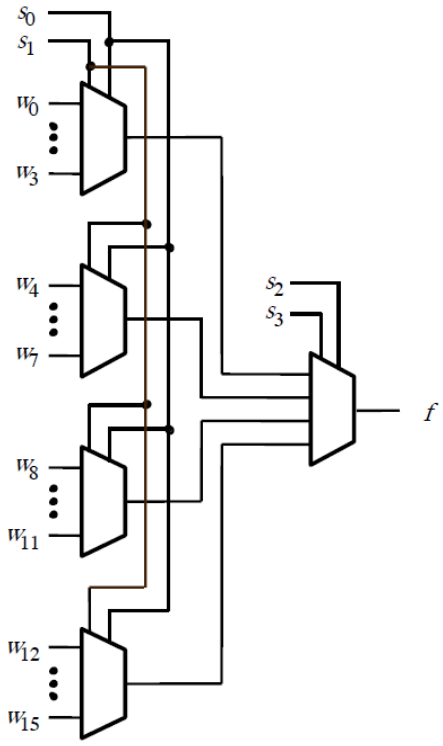
2) (10 points) Identify violations (if any) of the synthesis guidelines for synthesizable logic and the effect on the synthesized circuit.

```
entity test is
port(
  input    : in std_logic_vector(3 downto 0);
  clk, rst      : in std_logic;
  output   : out std_logic_vector(3 downto 0));
end test;

architecture BHV of test is
begin
process(clk, rst)
begin
    if (rst = '1') then
        output <= "0000";
    elsif(clk'event and clk = '1') then
        output <= std_logic_vector(unsigned(input)+ 1) after 10ns;
   end if;
end process;

process(input)
begin
    output <= std_logic_vector(unsigned(input) + 5);
end process;

end BHV;
```

Wait/time is not synthesizable!

Output is derived in multiple processes → Multiple drivers for a signal

3) (15 points) A 16x1 multiplexer is provided as reference. Fill in the code for designing this module using structural descriptions of 4x1 multiplexers and generate statement.




```
entity mux4to1 is
port (w0, w1, w2, w3: in std_logic;
s : in std_logic_vector(1 downto 0) ;
f : out std_logic) ;
end mux4to1;

architecture BHV of mux4to1 is
begin
with s select
f <= w0 when "00",
w1 when "01",
w2 when "10",
w3 when others;
end BHV;
```

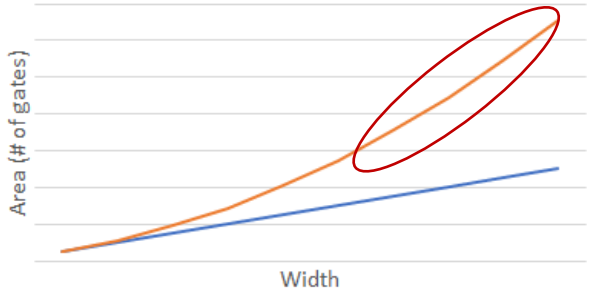
```
entity mux16to1 is
port (w : in std_logic_vector(0 downto 15);
s : in std_logic_vector(3 downto 0);
f : out std_logic);
end mux16to1;

architecture STR of mux16to1 is
      component mux4to1
      port (w0, w1, w2, w3: in std_logic;
            s : in std_logic_vector(1 downto 0);
            f : out std_logic );
      end component;

      signal m: std_logic_vector(3 downto 0);
      begin
      --code to be inserted
      MuxGEN: for i in 0 to 3 generate
      Muxes: mux4to1 port map
            (w0=>w(4*i), w1=>w(4*i+1), w2=>w(4*i+2), w3=>w(4*i+3),
            s=>s(1 DOWNTO 0), f=>m(i) ) ;
      END GENERATE ;

      --code to be inserted
      Mux_last: mux4to1 port map
            (w0=>m(0), w1=>m(1), w2=>m(2), w3=>m(3), s=>s(3 DOWNTO 2),
            f=>f ) ;


end STR;
```

4) (8 points) (a) Define each carry bit of a 4-bit carry lookahead adder (c(1), c(2), c(3), c(4)) in terms of the *propagate* and *generate* functions, and carry in(c(0)).

c(1) = g0 + p0 c(0)
c(2) = g1 + p1 c(1) = g1+p1g0+p1p0c(0)
c(3) = g2 + p2 c(2) = g2+p2 g1+p2 p1g0+p2p1p0c(0)
c(4) = g3+p3 c(3) = g3+p3g2+p3p2g1+p3p2p1g0+p3p2p1p0c(0)

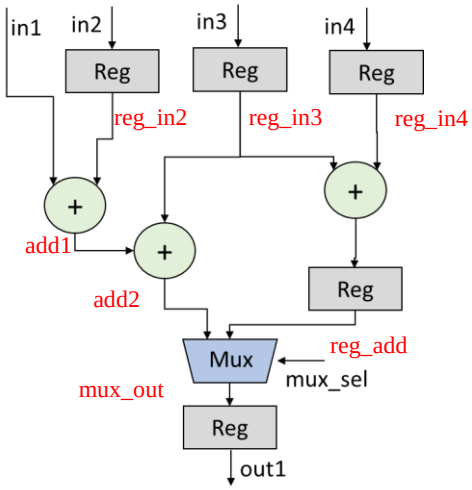
(b) (2 points) Circle the Delay vs width graph that seems appropriate for carry lookahead adder (CLA):

(c) (2 points) Circle the Area (# of gates) vs width graph that seems appropriate for carry lookahead adder (CLA):



(d) (3 points) True/false. A hierarchical carry-lookahead adder reduces area overhead compared to a single-level carry-lookahead adder without increasing propagation delay. False

5) (15 points) Fill in the following behavioral VHDL to implement the illustrated circuit. Assume that clk and rst connect to every register. All wires and operations are width bits. Ignore overflow from the adders.




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

entity test1 is
generic (width : positive := 8);
port (
clk, rst, mux_sel : in std_logic;
in1, in2, in3, in 4 : in std_logic_vector(width-1 downto 0);
out1: out std_logic_vector(width-1 downto 0));
end test1;

architecture BHV of test1 is
signal reg_in2,reg_in3, reg_in4, add1, add2, add3, reg_add, mux_out:
std_logic_vector(width-1 downto 0);


begin
process(clk, rst)

begin
if (rst = '1') then
      reg_in2 <= (others =>'0');
      reg_in3 <= (others =>'0');
      reg_in4 <= (others =>'0');
      reg_add <= (others =>'0');
      out1 <= (others =>'0');
elsif (rising_edge(clk))

      reg_in2 <= in2;
      reg_in3 <= in3;
      reg_in4 <= in4;
      reg_add <= std_logic_vector(unsigned(reg_in3)+unsigned(reg_in4));

      if mux_sel = '0' then
            out1 <= add1;
      else
            out1 <= reg_add;
      end if;

end if;
end process;
add1 <= std_logic_vector(unsigned(in1) + unsigned(reg_in2));
add2 <= std_logic_vector(unsigned(add1) + unsigned(reg_in3));

end BHV;
```

6) (20 points) Given the following entity definition for a generic adder, Modify the testbench on the next page to:

(a) To test the "adder" entity as a **32-bit** adder.

(b) Make the testbench code general; i.e., can be used to test for any number of bits by changing only the value of **NBITS**.

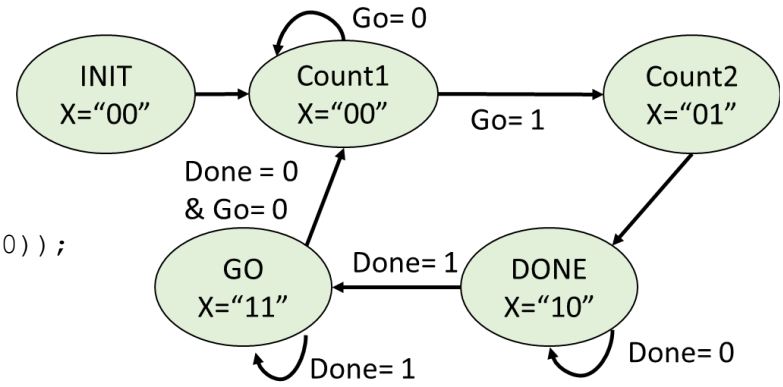
You need to make all necessary changes to make it work with the generic adder from the previous page **(put changes right on the next page)**

```
entity adder is
     generic (WIDTH : positive := 8);
     port map (input1, input2 : in std_logic_vector(WIDTH-1 downto 0);
             carry_in : in std_logic;
             sum: out std_logic_vector (WIDTH-1 downto 0);
             carry_out: out std_logic);
end adder;
```

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

entity adder_tb is
end adder_tb;

architecture TB of adder_tb is
constant NBITS : positive: 1632;
signal input1,input2,sum : std_logic_vector(3NBITS-1 downto 0);
signal carry_in, carry_out : std_logic;
begin  -- TB

UUT : entity work.adder
         port map (
            input1    => A input1,
            input2    => B input2,
            carry_in  => C carry_in,
            sum       => D sum,
            carry_out => E carry_out);

 process


-- function definitions here




 begin
    for i in 0 to 15 2**NBITS-1 loop
      for j in 0 to 15 2**NBITS-1 loop
        for k in 0 to 1 loop
          input1   <= std_logic_vector(to_unsigned(i, 4 NBITS))
          input2   <= std_logic_vector(to_unsigned(j, 4 NBITS))
          carry_in <= std_logic(to_unsigned(k, 1)(0));
          wait for 40 ns;
          assert(sum = sum_test(i,j,k)) report "Sum incorrect";
          assert(carry_out = carry_test(i,j,k)) report "Carry incorrect";
        end loop;  -- k
      end loop;  -- j
    end loop;  -- i
    report "SIMULATION FINISHED!";
    wait;
end process;
end TB;
```

```
function X carry_test (
    constant a, b, c:
integer)
    return std_logic is
 begin
    if (a + b + c > 15) then
return '1';
          else return '0';
    end if;
 end X carry_test;

function Y (
    constant a, b, c:
integer)
    return std_logic is
 begin
    return
std_logic_vector(to_unsign
ed(
              (a+b-c) mod
16, 4));
 end Y;

function Z sum_test (
    constant a, b, c:
integer)
    return std_logic_vector
is
 begin
    return
std_logic_vector(to_unsign
ed(
              (a+b+c) mod
16 2**NBITS, 4 NBITS));
 end Z sum_test;
```

7) (15 points) Fill in the skeleton code to implement the following Moore finite state machine, *using the 2-process FSM model*. Assume that if an edge does not have a corresponding condition, that edges always taken on a rising clock edge. Assume that INIT is the start state. Use the next page if extra room is needed.




```
library ieee;
use ieee.std_logic_1164.all;
entity fsm is
port (
clk, rst, go, Done : in std_logic;
X : out std_logic_vector(1 downto 0));
end fsm;


architecture PROC2 of fsm is
type STATE_TYPE is (INIT, COUNT1, COUNT2, DONE, GO);
signal state, next_state : STATE_TYPE;
begin
process(clk, rst)
begin
      if (rst = '1') then
            Stata <= INIT;

      elsif (clk'event and clk = '1') then

            State <= next_state;
      end if;
end process;

process( go, done, state)
begin
      next_state <= state;
      X<="00";
      Case state is
            When INIT =>
                  Next_state <= COUNT1;
                  X <= "00";
            When COUNT1 =>
                  If go = '1' then
                        Next_state <= COUNT2;
                  End if;
                  X <= "00";
            When COUNT2 =>
                  Next_state <= DONE;
                  X <= "01";
            When DONE =>
                  If done = '1' then
                        Next_state <= GO;
                  End if;
                  X <= "10";
```

```
When GO =>
            If done = '0' and go = '0' then
                  Next_state <= COUNT1;
            End if;
            X <= "11";
When others => null;
End case;
```

```
end process;
end PROC2;
```