Lab 4: Comparing Adder with Different Architectures

Entity Declaration

```
-- ----------------------------------------------
-- ENTITY description for Lab 4

-- ----------------------------------------------
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
--
ENTITY adder IS
PORT (a_in, b_in : IN std_logic_vector(7 downto 0);
     c_out : OUT std_logic_vector(7 downto 0));
end adder;
--
-- Your Architecture description will follow here
-- a different one for each of the three architectures
-- for example, if your are creating an "abstract_behavior"
-- architecture

ARCHITECTURE abstract_behavior of adder IS

BEGIN


END abstract_behavior;
```

Architecture 1: abstract_behavior

```
ARCHITECTURE abstract_behavior of adder IS

BEGIN

  c_out <= a_in + b_in;

END abstract_behavior;
```

Architecture 2: ripple_carry_adder

```
ARCHITECTURE ripple_carry_behavior of adder IS

BEGIN
add_proc : PROCESS (a_in, b_in)
```

```
    variable s: std_logic_vector (7 downto 0);
    variable c: std_logic;

  BEGIN
    c := '0';
    FOR i IN 0 TO 7 LOOP
      s(i) := a_in(i) XOR b_in(i) XOR c;
      c := (a_in(i) AND b_in(i)) OR (a_in(i) AND c) OR (b_in(i) AND c);
    END LOOP;

    c_out <= s;

  END PROCESS add_proc;

END ripple_carry_behavior;
```

Architecture 3: carry_look_ahead_adder

```
ARCHITECTURE carry_look_ahead_behavior OF adder IS

SIGNAL    sum           :    STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL    g             :    STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL    p             :    STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL    c_internal    :    STD_LOGIC_VECTOR(7 DOWNTO 1);
SIGNAL    carry_in      :    STD_LOGIC;

BEGIN
sum <= a_in XOR b_in;

-- Carry_generate (g) and carry_propagate (p) signals.

g <= a_in AND b_in;
p <= a_in OR b_in;

-- Recursive expansion allowing the adder to 'look ahead'
-- and determine if carry signals will be required.

PROCESS (g,p,c_internal)
BEGIN
carry_in <= '0';
c_internal(1) <= g(0) OR (p(0) AND carry_in);

     FOR i IN 1 TO 6 LOOP
     c_internal(i+1) <= g(i) OR (p(i) AND c_internal(i));
     END LOOP;

END PROCESS;

-- Assign final values to c_out signal

c_out(0) <= sum(0) XOR carry_in;
c_out(7 DOWNTO 1) <= sum(7 DOWNTO 1) XOR c_internal(7 DOWNTO 1);

END carry_look_ahead_behavior;
```

TEST BENCH for Lab 4

```
-- ------------------------------------------------
-- TEST BENCH for Lab 4
-- ------------------------------------------------
--
LIBRARY work;
USE work.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;

--
-
ENTITY testbench_adder IS
--
END testbench_adder;
-- ------------------------------------------------

ARCHITECTURE behavioral_testbench of testbench_adder IS

COMPONENT adder

PORT (a_in, b_in : IN std_logic_vector (7 downto 0);
        c_out : OUT std_logic_vector (7 downto 0));

END COMPONENT;

-- initialization

 SIGNAL a_in   : std_logic_vector (7 downto 0) := "00000000";

 SIGNAL b_in   : std_logic_vector (7 downto 0) := "00000000";

 SIGNAL c_out  : std_logic_vector (7 downto 0) := "00000000";

 SIGNAL c_test : std_logic_vector (7 downto 0) := "11111111";
--
-- you will have to change this for each of the three
-- different architectures that you develop
-- for example, with the generic_adder model you would use
--
--

for all : adder use entity work.adder(abstract_behavior);

BEGIN

 U1 : adder PORT MAP ( a_in, b_in, c_out);

a_in <=  "00000000",
        "00000001" after 100 ns,
```

```
        "00000000" after 125 ns,
        "00000000" after 150 ns,
        "11111110" after 175 ns,
        "11111111" after 200 ns,
        "01010101" after 225 ns,
        "10101010" after 250 ns,
        "11110000" after 275 ns,
        "00000000" after 300 ns;

b_in  <= "00000000",
        "00000000" after 100 ns,
        "10000000" after 125 ns,
        "01111111" after 150 ns,
        "00000001" after 175 ns,
        "00000001" after 200 ns,
        "10101011" after 225 ns,
        "01010101" after 250 ns,
        "00011111" after 275 ns,
        "00000000" after 300 ns;

--  here"s where we compare our results to the "known
--  good" results
--
PROCESS (a_in, b_in, c_test)

 VARIABLE test_ok : std_logic;

BEGIN
c_test <= a_in + b_in;
IF c_test = c_out THEN

 test_ok := '1';

ELSE

 test_ok := '0';

END IF;

END PROCESS;

END behavioral_testbench;
```

Assignments:

1. Using the provided testbench verify that each of the three architectures are working correctly.
2. Synthesis each of the three architectures.
3. Determine the longest path from input to output of each architecture.
4. Compute the propagation gate delay of the longest path of each architecture by adding all the gate delay of each gate in the longest path, according to the following table:

Gate Delay	Gate Type
1	Inv, or, nor, and, nand
2	Xor, xnor