

# LAB4 ASSIGNMENT

## 1. Carry Lookahead Adder

### 1.1 Adder

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.  use ieee.numeric_std.all;

4.  entity adder is
5.  port(
6.    x, y, cin : in  std_logic ;
7.    sum       : out std_logic
8.  );
9.  end entity;

10. architecture beh of adder is
11. begin
12. sum <= x xor y xor cin;
13. end beh;
```

### 1.2 Carry Block

```
1.  entity carry_block is
2.  port(
3.    cin : in std_logic;
4.    x, y: in std_logic_vector(3 downto 0);
5.    cout : out std_logic_vector(3 downto 0)
6.  );
7.  end entity ;
```

```

8.  architecture beh of carry_block is
9.  signal P0, P1, P2, P3 : std_logic;
10. signal G0, G1, G2, G3 : std_logic;
11. begin
12. P0 <= x(0) or y(0);
13. G0 <= x(0) and y(0);
14. P1 <= x(1) or y(1);
15. G1 <= x(1) and y(1);
16. P2 <= x(2) or y(2);
17. G2 <= x(2) and y(2);
18. P3 <= x(3) or y(3);
19. G3 <= x(3) and y(3);
20. cout(0) <= G0 or (P0 and cin);
21. cout(1) <= G1 or (P1 and cout(0));
22. cout(2) <= G2 or (P2 and cout(1));
23. cout(3) <= G3 or (P3 and cout(2));

24. end beh; library ieee;

```

## 2.3. Carry Lookahead Adder

```

1.  entity carry_lookahead_adder is
2.  port(
3.  x, y : in std_logic_vector(3 downto 0);
4.  cin : in std_logic;
5.  sum : out std_logic_vector(3 downto 0);
6.  cout : out std_logic
7.  );
8.  end entity;
9.  architecture struct of carry_lookahead_adder is
10. signal s_internal_carries : std_logic_vector(3 downto 0);
11. component adder is
12. port(x, y, cin: in std_logic ;
13. sum : out std_logic
14. );
15. end component;
16. component carry_block is
17. port(
18. cin : in std_logic;
19. x, y: in std_logic_vector(3 downto 0);
20. cout : out std_logic_vector(3 downto 0)
21. );
22. end component;

```

```

23. begin
24. CB : carry_block port map
25. (cin , x, y ,s_internal_carries);

26. adder0 : adder port map
27. (x(0) , y(0), cin, sum(0));

28. adder1 : adder port map
29. (x(1) , y(1), s_internal_carries(0), sum(1));

30. adder2 : adder port map
31. (x(2) , y(2), s_internal_carries(1), sum(2));

32. adder3 : adder port map
33. (x(3) , y(3), s_internal_carries(2), sum(3));

34. cout <= s_internal_carries(3);

35. end struct ;

```

## 1.4. Carry Lookahead Adder Test Banch

```

1. entity carry_lookahead_adder_tb is
2. end carry_lookahead_adder_tb;
3.
   architecture behavior of carry_lookahead_adder_tb is
4. signal x_tb, y_tb : std_logic_vector(3 downto 0);
5. signal cin_tb : std_logic;
6. signal sum_tb : std_logic_vector(3 downto 0);
7. signal cout_tb : std_logic;
8.
   component carry_lookahead_adder is
9. port(
10. x, y : in std_logic_vector(3 downto 0);
11. cin : in std_logic;
12. sum : out std_logic_vector(3 downto 0);
13. cout : out std_logic
14. );
15. end component;

```

```

16. begin
    DUT: carry_lookahead_adder port map(
17. x => x_tb,
18. y => y_tb,
19. cin => cin_tb,
20. sum => sum_tb,
21. cout => cout_tb
22. );
23.
    stimulus_proc : process
24. begin
25. -- Test case 1: Add 4 + 5 without carry
26. x_tb <= "0100"; -- 4
27. y_tb <= "0101"; -- 5
28. cin_tb <= '0';
29. wait for 2 ns;
30.
    -- Test case 2: Add 7 + 3 with carry
31. x_tb <= "0111"; -- 7
32. y_tb <= "0011"; -- 3
33. cin_tb <= '1';
34. wait for 2 ns;
35.
    -- Test case 3: Add 15 + 15 without carry
36. x_tb <= "1111"; -- 15
37. y_tb <= "1111"; -- 15
38. cin_tb <= '0';
39. wait for 2 ns;
40.
    -- Test case 3: Add 15 + 15 with carry
41. x_tb <= "1111"; -- 15
42. y_tb <= "1111"; -- 15
43. cin_tb <= '1';
44. wait for 2 ns;
45.
    -- Test case 4: Add 0 + 0 with carry
46. x_tb <= "0000"; -- 0
47. y_tb <= "0000"; -- 0
48. cin_tb <= '1';
49. wait for 2 ns;
50.
    wait;
51. end process;
52. end behavior;

```



## 2. Wallace Tree Multiplier

### 2.1 Full Adder

```
1. entity full_adder is
2. port(
3. x, y, cin : in std_logic;
4. sum, cout : out std_logic
5. );
6. end entity;
7.
   architecture beh of full_adder is
8. begin
9. sum <= x xor y xor cin ;
10.cout <= (x and y ) or ((x or y) and
    cin);
11.end beh;
```

### 2.2 Wallace Tree Multiplier

```
1. entity wallace_tree_multiplier is
2. port(
3. x, y : in std_logic_vector(3 downto 0);
4. p : out std_logic_vector(7 downto 0)
5. );
6. end entity;
7.
   architecture struct of wallace_tree_multiplier is
8. signal s_bottom_level_carry_out : std_logic_vector(4 downto 0) := "00000";
9. signal s_middle_level_carry_out : std_logic_vector(3 downto 0) := "0000";
10.signal s_top_level_carry_out : std_logic_vector(1 downto 0) := "00";
11.
    signal s_middle_level_sum : std_logic_vector(3 downto 0) := "0000";
12.signal s_top_level_sum : std_logic_vector(1 downto 0) := "00";
13.
    component full_adder is
14.port(
15.x, y, cin : in std_logic;
16.sum, cout : out std_logic
17.);
18.end component;
19.begin
20.
    p(0) <= (x(0) and y(0));
21.
    --mapping the full adders
22.FA_0 : full_adder
23.port map(
24.x => (x(0) and y(1)),
25.y => (x(1) and y(0)),
26.cin => '0',
27.sum => p(1),
28.cout => s_bottom_level_carry_out(0)
29.);
```

```

30.FA_1 : full_adder
31.port map(
32.x => (x(2) and y(0)),
33.y => (x(1) and y(1)),
34.cin => '0',
35.sum => s_middle_level_sum(0),
36.cout => s_middle_level_carry_out(0)
37.);

```

```

38.FA_2 : full_adder
39.port map(
40.x => (x(3) and y(0)),
41.y => (x(2) and y(1)),
42.cin => '0',
43.sum => s_top_level_sum(0),
44.cout => s_top_level_carry_out(0)
45.);

```

```

46.FA_3 : full_adder
47.port map(
48.x => s_middle_level_sum(0),
49.y => (x(0) and y(2)),
50.cin => s_bottom_level_carry_out(0),
51.sum => p(2),
52.cout => s_bottom_level_carry_out(1)
53.);

```

```

54.FA_4 : full_adder
55.port map(
56.x => (x(0) and y(3)),
57.y => s_top_level_sum(0),
58.cin => (x(1) and y(2)),
59.sum => s_middle_level_sum(1),
60.cout => s_middle_level_carry_out(1)
61.);

```

```

62.FA_5 : full_adder
63.port map(
64.x => (x(3) and y(1)),
65.y => (x(2) and y(2)),
66.cin => '0',
67.sum => s_top_level_sum(1),
68.cout => s_top_level_carry_out(1)
69.);

```

```

70.FA_6 : full_adder
71.port map(
72.x => s_middle_level_sum(1),
73.y => s_middle_level_carry_out(0),
74.cin => s_bottom_level_carry_out(1),
75.sum => p(3),
76.cout => s_bottom_level_carry_out(2)
77.);

```

```

78.FA_7 : full_adder
79.port map(
80.x => (x(1) and y(3)),
81.y => s_top_level_sum(1),
82.cin => s_top_level_carry_out(0),
83.sum => s_middle_level_sum(2),
84.cout => s_middle_level_carry_out(2)
85.);

```

```

86.FA_8 : full_adder
87.port map(
88.x => s_middle_level_sum(2),
89.y => s_middle_level_carry_out(1),
90.cin => s_bottom_level_carry_out(2),
91.sum => p(4),
92.cout => s_bottom_level_carry_out(3)
93.);

```

```

94.FA_9 : full_adder
95.port map(
96.x => (x(3) and y(2)),
97.y => (x(2) and y(3)),
98.cin => s_top_level_carry_out(1),
99.sum => s_middle_level_sum(3),
100.cout => s_middle_level_carry_out(3)
101.);

```

```

102.FA_10 : full_adder
103.port map(
104.x => s_middle_level_sum(3),
105.y => s_middle_level_carry_out(2),
106.cin => s_bottom_level_carry_out(3),
107.sum => p(5),
108.cout => s_bottom_level_carry_out(4)
109.);

```

```

110.FA_11 : full_adder
111.port map(
112.x => (x(3) and y(3)),
113.y => s_middle_level_carry_out(3),
114.cin => s_bottom_level_carry_out(4),
115.sum => p(6),
116.cout => p(7)
117.);
118.end struct;

```

## 2.3. Wallace Tree Multiplier Test Banch

```
1. entity wallace_tree_multiplier_tb is
2. end wallace_tree_multiplier_tb;
3.
   architecture beh of wallace_tree_multiplier_tb is
4. signal x_tb, y_tb : std_logic_vector(3 downto 0);
5. signal p_tb : std_logic_vector(7 downto 0);
6.
   component wallace_tree_multiplier is
7. port(
8. x, y : in std_logic_vector(3 downto 0);
9. p : out std_logic_vector(7 downto 0)
10.);
11. end component;
12. begin
13.
   DUT: wallace_tree_multiplier port map(
14. x => x_tb,
15. y => y_tb,
16. p => p_tb
17.);
18.
   stimulus_proc : process
19. begin
20. -- Test case 1: multiply 4 * 5
21. x_tb <= "0100"; -- 4
22. y_tb <= "0101"; -- 5
23. wait for 2 ns;
24.
   -- Test case 2: multiply 1 * 1
25. x_tb <= "0001"; -- 1
26. y_tb <= "0001"; -- 1
27. wait for 2 ns;
28.
   -- Test case 3: multiply 15 * 15
29. x_tb <= "1111"; -- 15
30. y_tb <= "1111"; -- 15
31. wait for 2 ns;
32.
   -- Test case 3: multiply 1 * 15
33. x_tb <= "0001"; -- 1
34. y_tb <= "1111"; -- 15
35. wait for 2 ns;
36.
   -- Test case 4: multiply 0 * 15
37. x_tb <= "0000"; -- 0
38. y_tb <= "1111"; -- 15
39. wait for 2 ns;
40.
   wait;
41. end process;
42. end beh;
```



## 2.4. Simulation

### 2.4.1. Set up

The screenshot displays the EDA Playground web interface. The top navigation bar includes the EDA Playground logo, a 'New' button, a 'Run' button, a 'Save' button, a 'KnowHow WEBINARS' banner, a 'Clock Domain Crossing' banner, a 'REGISTER NOW' button, and links to 'Resources', 'Community', 'Help', 'Playgrounds', and 'Profile'.

The left sidebar contains the 'Languages & Libraries' section, 'Testbench + Design' dropdown, 'Libraries' list (None, OVL, OSVVM), 'Top entity' (wallace\_tree\_multiplier\_tb), 'Tools & Simulators' (Aldec Riviera Pro 2023.04), 'Compile Options' (-2019 -o), 'Run Options' (Run Options), 'Run Time' (10 ms), and 'Examples' (using EDA Playground, VHDL, Verilog/SystemVerilog, UVM, EasierUVM, SVAUnit, SVI Init).

The main editor area shows two Verilog files: `testbench.vhd` and `wallace_tree_multiplier_tb.vhd`. The `testbench.vhd` code defines a testbench for the multiplier, including signal declarations, component instantiation, and a stimulus process. The `wallace_tree_multiplier.vhd` code defines the multiplier architecture using three full adders (FA\_0, FA\_1, FA\_2) and a carry chain.

The bottom panel shows the simulation log, which includes warnings about the Riviera-PRO EDU Edition and a message indicating that the simulation has finished. The log also shows the opening of the EPWave window.

### 2.4.2. Result

