VHDL Assignment 3

1- Names

Loïc Duchesne - 261052490
Yassine Mimet - 260980175

2- Executive summary

In this lab, for the first part we did a 4-bit circular barrel shifter. We did that by implementing 8 2-to-1 muxes which are connected with each other, where in this part we have 2 inputs X each being a 4 digit input and Select being a 2 digit input(0th digit going to the second layer of the muxes and the first digit going to the first layer of muxes). Finally, we have a 4 digits output Y.

In the second part, the objective was to implement the RCA (Ripple Carry Adder), and use it to implement the BCD Adder. During the process, we had to implement a Half-Adder using a structural description, and then we used this implementation to do the Full-Adder structural description. Afterwards, for the RCA, we implemented it using one Half-Adder and 3 Full-Adders (because we don't need a carry for the first bit). Finally, for the BCD Adder, we used 2 RCA adders, a OR gate & 2 AND gates. For its behavioral counterpart, we used a reference in the book as instructed in the lab manual for the BCD Adder.

3- Questions

1) VHDL code implementation explanations

For the **structural 4-bit circular Barrel Shifter**, we used 8 muxes that we imported (from VHDL 2 lab) with the default library work (which was easier than copying the component code directly into our barrel shifter code). We connected the two layers of the 2-to-1 muxes using four 1-bit signal that we defined in the architecture of the structural.

Structural barrelshifter code:

```vhdl
-- Structural 4-bit Circular Barrel Shifter
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library work;

entity loic_duchesne_barrel_shifter_structural is
    Port (X: in std_logic_vector (3 downto 0);
          sel: in std_logic_vector (1 downto 0);
          Y: out std_logic_vector (3 downto 0));
end loic_duchesne_barrel_shifter_structural;

architecture struct of loic_duchesne_barrel_shifter_structural is
    signal sig0, sig1, sig2, sig3 : std_logic;
begin

--leftmost layer
mux0: entity work.loic_duchesne_MUX_structural port map ( X(0), X(2), sel(1), sig0);
mux1: entity work.loic_duchesne_MUX_structural port map ( X(1), X(3), sel(1), sig1);
mux2: entity work.loic_duchesne_MUX_structural port map ( X(2), X(0), sel(1), sig2);
mux3: entity work.loic_duchesne_MUX_structural port map ( X(3), X(1), sel(1), sig3);
--rightmost layer
mux4: entity work.loic_duchesne_MUX_structural port map ( sig0, sig3, sel(0), Y(0));
mux5: entity work.loic_duchesne_MUX_structural port map ( sig1, sig0, sel(0), Y(1));
mux6: entity work.loic_duchesne_MUX_structural port map ( sig2, sig1, sel(0), Y(2));
mux7: entity work.loic_duchesne_MUX_structural port map ( sig3, sig2, sel(0), Y(3));
 end struct;
```

For the **behavioral 4-bit circular Barrel Shifter**, we just concatenated the X(i) for every single case of the selector. Then, we used the Select values to predict the outputs and assign them to the correct X combinations.

Behavioral barrelshifter code:

```
-- Behavioral 4-bit Circular Barrel Shifter
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity loic_duchesne_barrel_shifter_behavioral is
    Port (X: in std_logic_vector (3 downto 0);
          sel: in std_logic_vector (1 downto 0);
          Y: out std_logic_vector (3 downto 0));
end loic_duchesne_barrel_shifter_behavioral;

architecture behav of loic_duchesne_barrel_shifter_behavioral is
begin
    with sel select
        --Y[3]   Y[2]   Y[1]   Y[0]
       Y <= X(3) & X(2) & X(1) & X(0) when "00",
            X(2) & X(1) & X(0) & X(3) when "01",
            X(1) & X(0) & X(3) & X(2) when "10",
            X(0) & X(3) & X(2) & X(1) when "11",
            "XXXX" when others;
end behav;
```

For the **4-bit circular Barrel Shifter test bench**, we assigned X to the value “1011” and then wrote a for loop so it could iterate over all the different values of the selector. This means we had 4 possible combinations paired with the fixed X value. (We also did a fully comprehensive test using all possible values of X & sel, see part 4 for those tests).

barrelshifter testbench (same test bench for both structural & behavioral except for the naming syntax):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity loic_duchesne_barrel_shifter_structural_tb_2 is
end loic_duchesne_barrel_shifter_structural_tb_2;
architecture tb of loic_duchesne_barrel_shifter_structural_tb_2 is
-- signals
signal X: std_logic_vector (3 downto 0);
signal sel: std_logic_vector (1 downto 0);
signal Y: std_logic_vector (3 downto 0);

component loic_duchesne_barrel_shifter_structural
    Port (X: in std_logic_vector (3 downto 0);
          sel: in std_logic_vector (1 downto 0);
          Y: out std_logic_vector (3 downto 0));
end component;

begin
    i1 : loic_duchesne_barrel_shifter_structural port map (X, sel, Y);

generate_test: process
begin
    --Setting X to "1011"
    X <= "1011";

    for j in 0 to 4 loop
    sel <= std_logic_vector(to_unsigned(j, 2));
    wait for 10 ns;
    end loop;
    wait;
end process generate_test;
end tb;
```

For the **Half-Adder structural** we assigned a XOR output gate with inputs a, b for the s output, and for the c output we assigned AND gate output to it with inputs a and b as well.

Half-Adder structural code:

```vhdl
-- Structural Half-Adder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity half_adder is
    port (a: in std_logic;
          b: in std_logic;
          s: out std_logic;
          c: out std_logic);
end half_adder;

architecture struct of half_adder is
begin
    s <= a XOR b;
    c <= a AND b;

end struct;
```

For the **Half-Adder Test Bench** we could've made all 4 cases, but it was easier to do using the loops so basically it is 2 cases inside if 2 cases so 4 cases overall.

Half-Adder test bench code:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity half_adder_structural_tb is
end half_adder_structural_tb;
architecture tb of half_adder_structural_tb is
-- signals
signal a: std_logic;
signal b: std_logic;
signal s: std_logic;
signal c: std_logic;

component half_adder
    port (a: in std_logic;
          b: in std_logic;
          s: out std_logic;
          c: out std_logic);
end component;

begin
    ha : half_adder port map (a, b, s, c);

generate_test: process
begin
    for i in std_logic range '0' to '1' loop
    a <= i;
        for j in std_logic range '0' to '1' loop
            b <= j;
            wait for 10 ns;

    end loop;
    end loop;
    wait;
end process generate_test;
end tb;
```

For the **Full-Adder structural** we used 2 Half-Adders (that we imported) and an OR gate as it was shown in class. You may refer to the RTL schematic in part 4 for the Full-Adder.

Full-Adder structural code:

```vhdl
-- Structural Full-Adder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library work;

entity full_adder is
    port (a: in std_logic;
          b: in std_logic;
          c_in: in std_logic;
          s: out std_logic;
          c_out: out std_logic);
end full_adder;

architecture struct of full_adder is
    signal sig0, sig1, sig2 : std_logic;
begin
    ha0: entity work.half_adder port map ( a, b, sig0, sig1);
    ha1: entity work.half_adder port map ( c_in, sig0, s, sig2);
    c_out <= sig1 OR sig2;

end struct;
```

For the **Full-Adder Test Bench** We could have used the 8 cases in total but we decided to use the loops to make it easier. We had 3 1 bit inputs, so 2 cases for every input which gives 8 cases in total (see the picture of the tb for more clarity).

Full-Adder test bench code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity full_adder_structural_tb is
end full_adder_structural_tb;
architecture tb of full_adder_structural_tb is
-- signals
signal a: std_logic;
signal b: std_logic;
signal c_in: std_logic;
signal s: std_logic;
signal c_out: std_logic;

component full_adder
   port (a: in std_logic;
         b: in std_logic;
         c_in: in std_logic;
         s: out std_logic;
         c_out: out std_logic);
end component;

begin
   fa : full_adder port map (a, b, c_in, s, c_out);

generate_test: process
begin
   for i in std_logic range '0' to '1' loop
   a <= i;
      for j in std_logic range '0' to '1' loop
         b <= j;
            for k in std_logic range '0' to '1' loop
            c_in <= k;
         wait for 10 ns;

   end loop;
   end loop;
   end loop;
   wait;
end process generate_test;
end tb;
```

For the **RCA structural**, we used a Half-Adder at first since we don't have a carry, which makes it so we don't have to use a Full-Adder there. For the last 3 components we used 3 Full-Adders because we're adding three 1-bit variables: the 2 main bits and the carry.

RCA structural code:

```
-- Structural 4-bit Ripple-Carry Adder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library work;

entity rca_structural is
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(4 downto 0));
end rca_structural;

architecture struct of rca_structural is
   signal sig0, sig1, sig2 : std_logic;
begin
   -- Half-Adder
   ha0: entity work.half_adder port map(B(0), A(0), S(0), sig0);
   -- Full-Adders
   fa0: entity work.full_adder port map ( B(1), A(1), sig0, S(1), sig1);
   fa1: entity work.full_adder port map ( B(2), A(2), sig1, S(2), sig2);
   fa2: entity work.full_adder port map ( B(3), A(3), sig2, S(3), S(4));

end struct;
```

For the **RCA structural Test Bench**, we used FOR loops again because we had 2 inputs of 4 bits each, and therefore we have 256 test cases.

RCA structural test bench code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rca_structural_tb is
end rca_structural_tb;
architecture tb of rca_structural_tb is
-- signals
signal A: std_logic_vector (3 downto 0);
signal B: std_logic_vector (3 downto 0);
signal S: std_logic_vector (4 downto 0);

component rca_structural
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(4 downto 0));
end component;

begin
   rca : rca_structural port map (A, B, S);

generate_test: process
begin
   for i in 0 to 16 loop
   A <= std_logic_vector(to_unsigned(i, 4));
      for j in 0 to 16 loop
      B <= std_logic_vector(to_unsigned(j, 4));
      wait for 10 ns;
   end loop;
   end loop;
   wait;
end process generate_test;
end tb;
```

For the **RCA Behavioral**, We defined a signal ‘sig’ to be an integer and then we assigned that signal to_integer(unsigned(A)) + to_integer(unsigned(B)). And assign to the output value the std_numeric_vector(to_unsigned(sig, 5).

RCA behavioral code:

```
-- Behavioral 4-bit Ripple-Carry Adder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rca_behavioral is
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(4 downto 0));
end rca_behavioral;

architecture behav of rca_behavioral is
   signal sig: integer;
begin
      sig <= to_integer(unsigned(A)) + to_integer(unsigned(B));
      S <= std_logic_vector(to_unsigned(sig, 5));

end behav;
```

For the **RCA behavioral Test Bench**, we had the same test bench as the structural one, which the only differences being the naming syntax.

RCA behavioral test bench code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rca_behavioral_tb is
end rca_behavioral_tb;
architecture tb of rca_behavioral_tb is
-- signals
signal A: std_logic_vector(3 downto 0);
signal B: std_logic_vector(3 downto 0);
signal S: std_logic_vector(4 downto 0);

component rca_behavioral
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(4 downto 0));
end component;

begin
   rca : rca_behavioral port map (A, B, S);

generate_test: process
begin
   for i in 0 to 16 loop
   A <= std_logic_vector(to_unsigned(i, 4));
      for j in 0 to 16 loop
      B <= std_logic_vector(to_unsigned(j, 4));
      wait for 10 ns;
   end loop;
   end loop;
   wait;
end process generate_test;
end tb;
```

For the **BCD adder structural**, we tried to implement it using two RCA's and a subpart that contains an OR gate and 2 AND gates. The subpart will take the fifth digit of the output (rc0_output(5)) of the first RCA (rca0), and will take it as an input for the OR gate. The two other inputs are just the outputs of the two AND gates. The first AND gate will take as input the second and the fourth first digits of the first RCA output, and the second AND gate will take as input the third and the fourth digits of the first RCA output. Then, the output of this subpart will be the output carry of the BCD adder. Finally, we get the second output S from the 2nd RCA. It will be only the 4 first digits of that RCA. That RCA takes two inputs, the first one being the first 4 digits of the 1st RCA's output. For the second input, we basically concatenated '0' & 'output_carry' & 'output_carry' & '0', so it will be either '0000' when the output carry is 0 or it will be '0110' if the output carry is 1.

BCD adder structural code:

```
-- Structural one-digit BCD adder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library work;

entity bcd_adder_structural is
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(3 downto 0);
         C: out std_logic);
end bcd_adder_structural;

architecture struct of bcd_adder_structural is
   signal rca0_output, rca1_output : std_logic_vector(4 downto 0);
   signal rca1_A_input, rca1_B_input : std_logic_vector(3 downto 0);
   signal output_carry : std_logic;
begin
   rca0: entity work.rca_structural port map(A, B, rca0_output);

   rca1_A_input <= rca0_output(3) & rca0_output(2) & rca0_output(1) & rca0_output(0);

   output_carry <= rca0_output(4) OR (rca0_output(2) AND rca0_output(3)) OR (rca0_output(1) AND rca0_output(3));

   rca1_B_input <= '0' & output_carry & output_carry & '0';

   rca1: entity work.rca_structural port map(rca1_A_input, rca1_B_input, rca1_output);

   S <= rca1_output(3) & rca1_output(2) & rca1_output(1) & rca1_output(0);
   C <= output_carry;

end struct;
```

For the **BCD adder structural Test Bench**, we had also 256 cases because it was 2 loops of two 4 bits inputs.

BCD adder structural testbench code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity bcd_adder_structural_tb is
end bcd_adder_structural_tb;
architecture tb of bcd_adder_structural_tb is
-- signals
signal A: std_logic_vector(3 downto 0);
signal B: std_logic_vector(3 downto 0);
signal S: std_logic_vector(3 downto 0);
signal C: std_logic;

component bcd_adder_structural
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(3 downto 0);
         C: out std_logic);
end component;

begin
   bcd : bcd_adder_structural port map (A, B, S, C);

generate_test: process
begin
   for i in 0 to 16 loop
   A <= std_logic_vector(to_unsigned(i, 4));
      for j in 0 to 16 loop
         B <= std_logic_vector(to_unsigned(j, 4));
         wait for 10 ns;

   end loop;
   end loop;
   wait;
end process generate_test;
end tb;
```

For the **BCD adder behavioral**, we used the example from the book and adjusted it so that it suits our cases. We had to define 3 signals in our architecture so that we could use them to connect different operations. Firstly, we check if the addition of our two inputs is bigger than 9, which is just (‘1001’). We then put the carry to be equal to 1 otherwise the carry will be ‘0’. If the carry is 0, the output will simply be the addition of our two inputs, otherwise the output will be the addition + 6 which is just ‘0110’. Finally, the C gets the value of the output carry signal and the other output gets the first 4 digits of the full sum.

BCD Adder behavioral code:

```
-- Behavioral one-digit BCD adder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bcd_adder_behavioral is
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(3 downto 0);
         C: out std_logic);
end bcd_adder_behavioral;

architecture behav of bcd_adder_behavioral is
   signal Z : std_logic_vector(4 downto 0);
   signal output : std_logic_vector(4 downto 0);
   signal adjust : std_logic;
begin
      Z <= ('0' & A) + B;
      adjust <= '1' when Z>9 else '0';
      output <= Z when (adjust='0') else Z + 6;

      S <= output(3) & output(2) & output(1) & output(0);
      C <= adjust;

end behav;
```

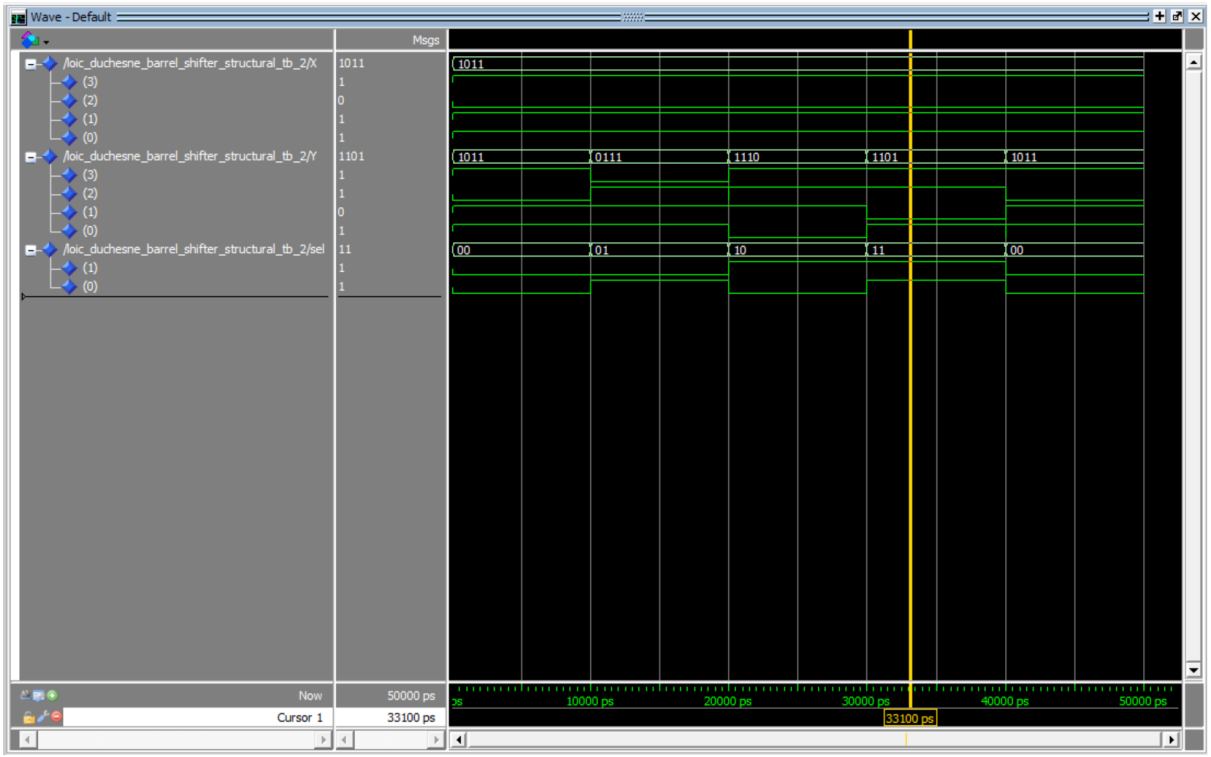
For the **BCD adder behavioral Test Bench**, we had also 256 cases because it was 2 loops of two 4 digits inputs.

BCD Adder behavioral testbench code:

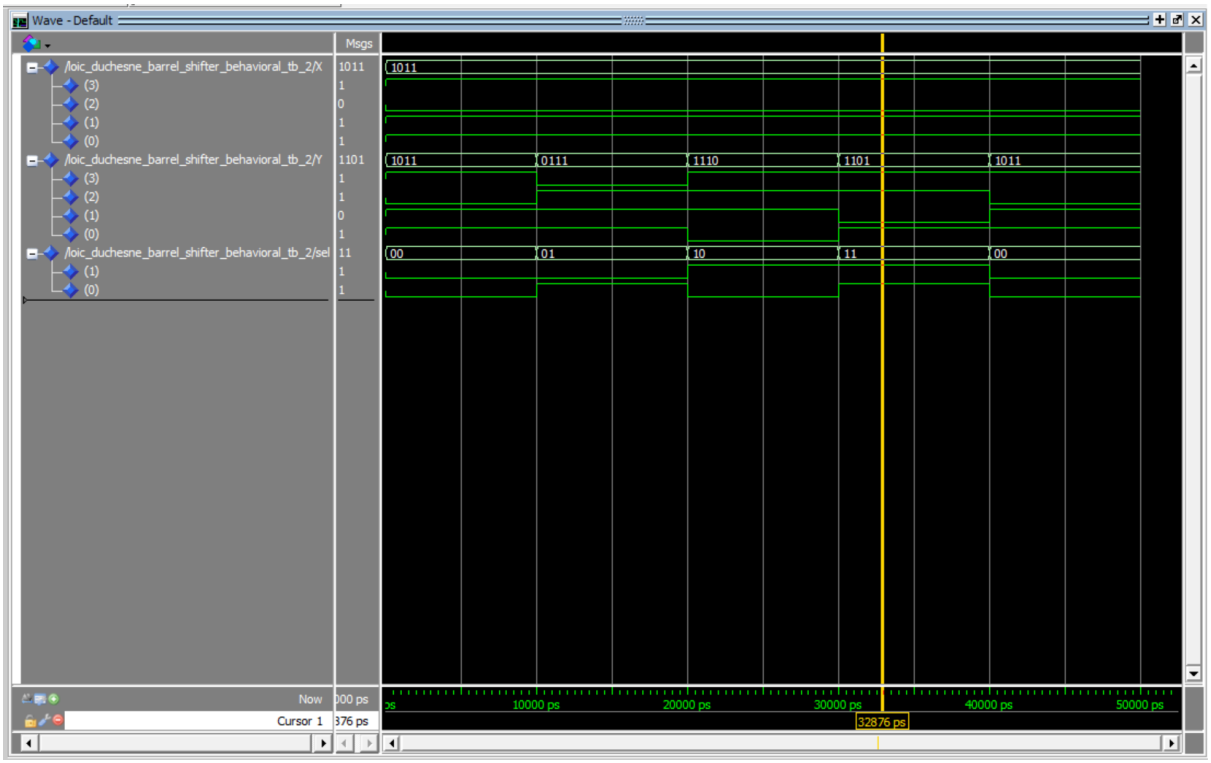
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity bcd_adder_behavioral_tb is
end bcd_adder_behavioral_tb;
architecture tb of bcd_adder_behavioral_tb is
-- signals
signal A: std_logic_vector(3 downto 0);
signal B: std_logic_vector(3 downto 0);
signal S: std_logic_vector(3 downto 0);
signal C: std_logic;

component bcd_adder_behavioral
   port (A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         S: out std_logic_vector(3 downto 0);
         C: out std_logic);
end component;

begin
   bcd : bcd_adder_behavioral port map (A, B, S, C);

generate_test: process
begin
   for i in 0 to 16 loop
   A <= std_logic_vector(to_unsigned(i, 4));
      for j in 0 to 16 loop
         B <= std_logic_vector(to_unsigned(j, 4));
         wait for 10 ns;

   end loop;
   end loop;
   wait;
end process generate_test;
end tb;
```

2) Structural and behavioral tests for Barrel Shifter (At given input X = “1011”)

Barrel Shifter structural test for all possible values of sel at select input of X = “1011”:

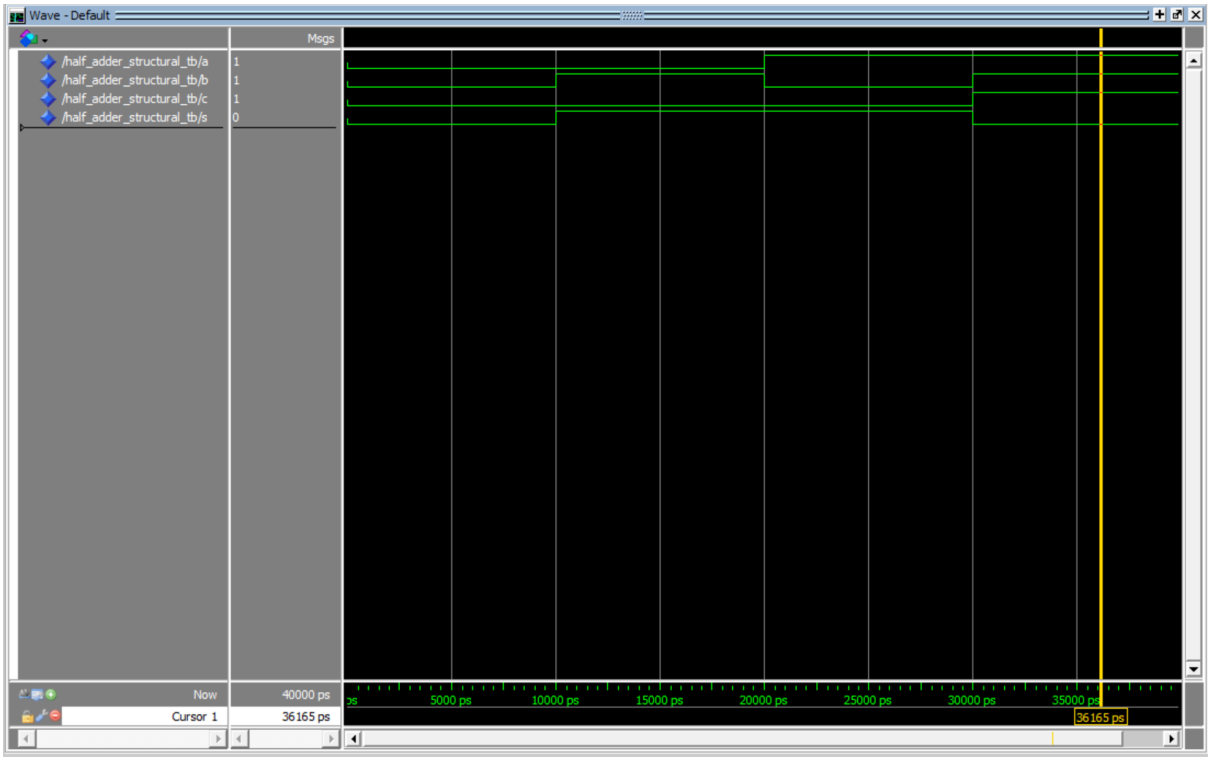


Barrel Shifter behavioral test for all possible values of sel at select input of X = “1011”:



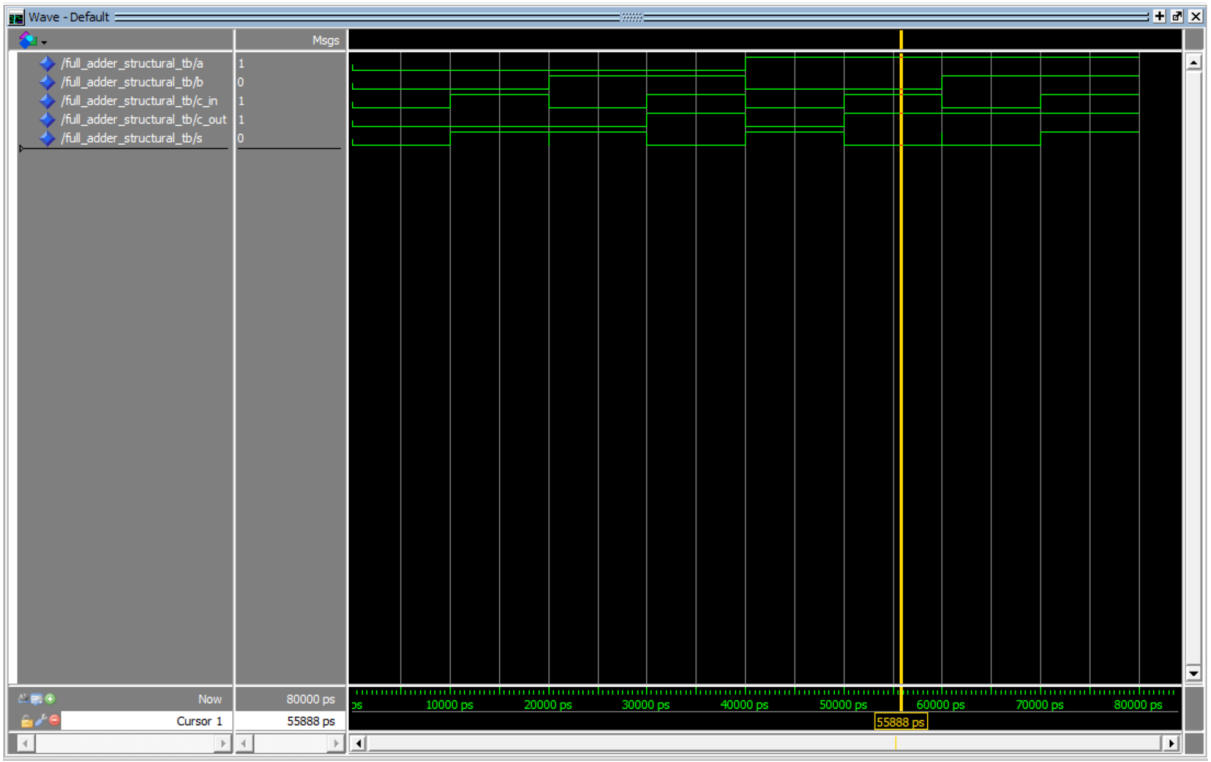
3) Representative simulation plots for half-adder circuits

Half-adder structural test for all possible values:



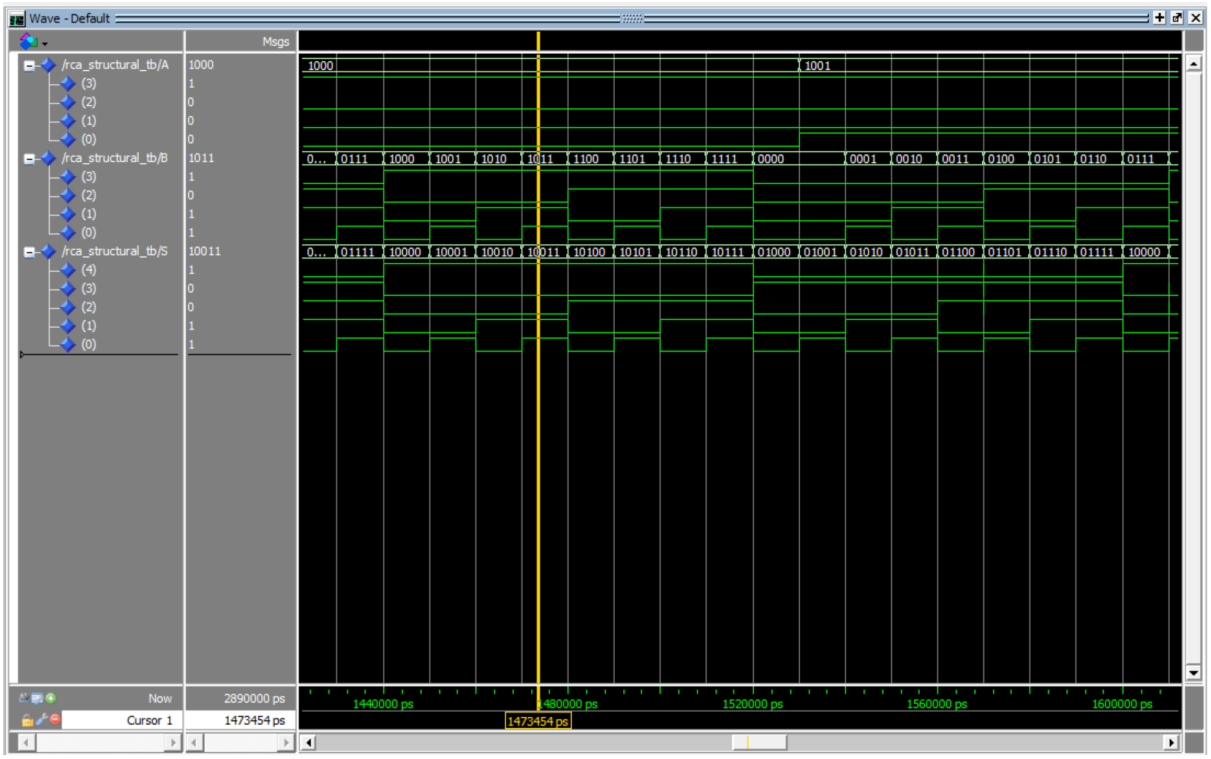
4) Representative simulation plots for full-adder circuits

Full-adder structural test for all possible values:

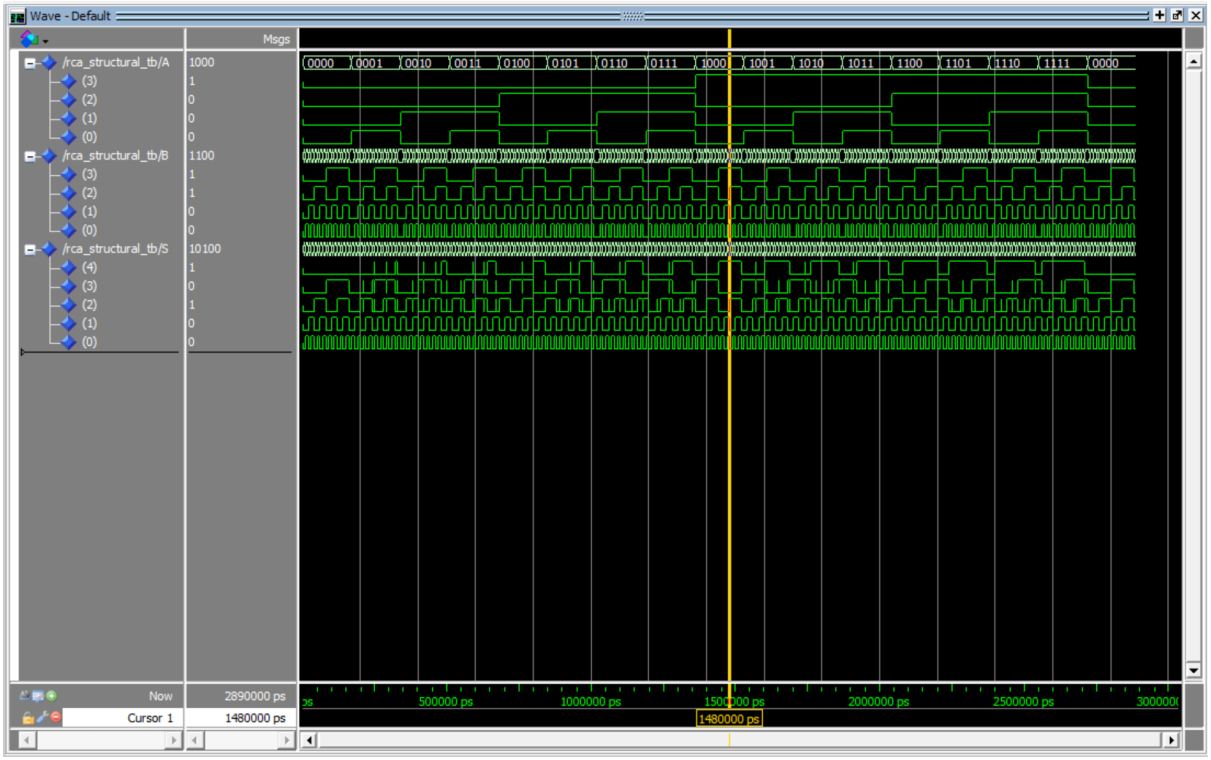


5) Representative simulation plots for the 4-bit RCA

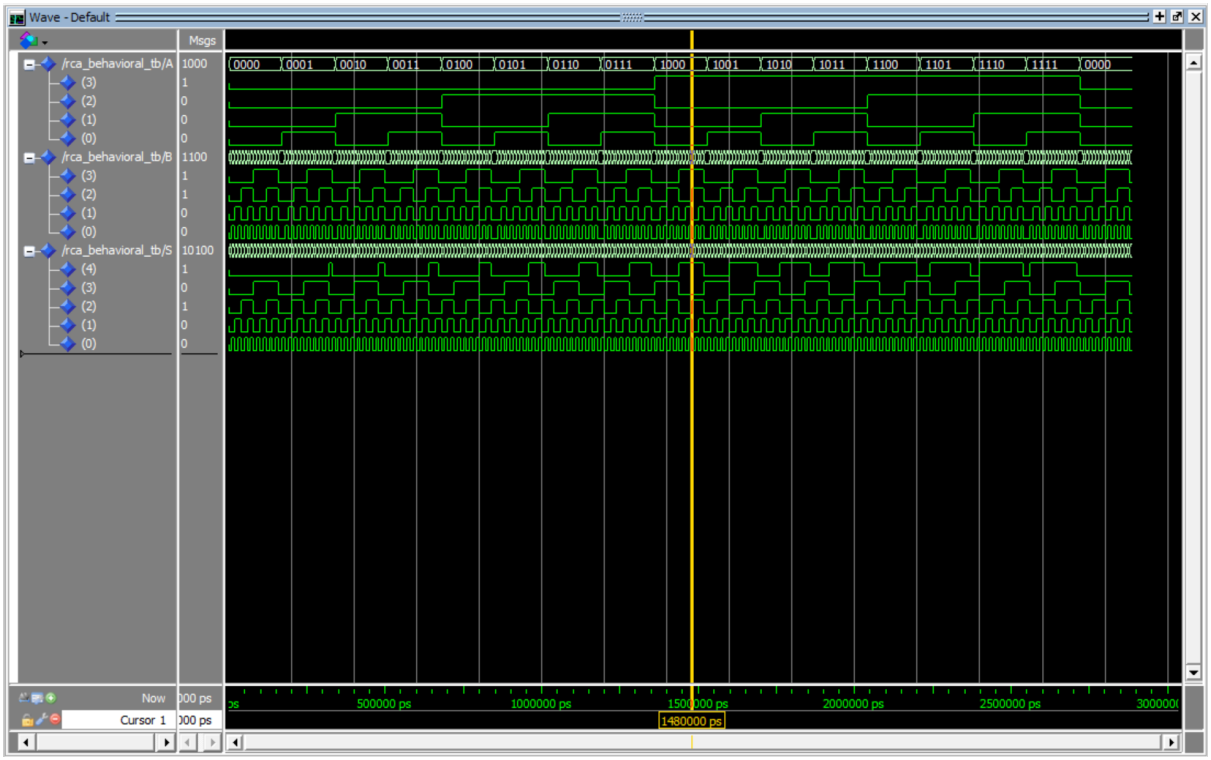
4-bit RCA structural test (zoomed in to observe different input/output cases):



4-bit RCA structural test (full size):

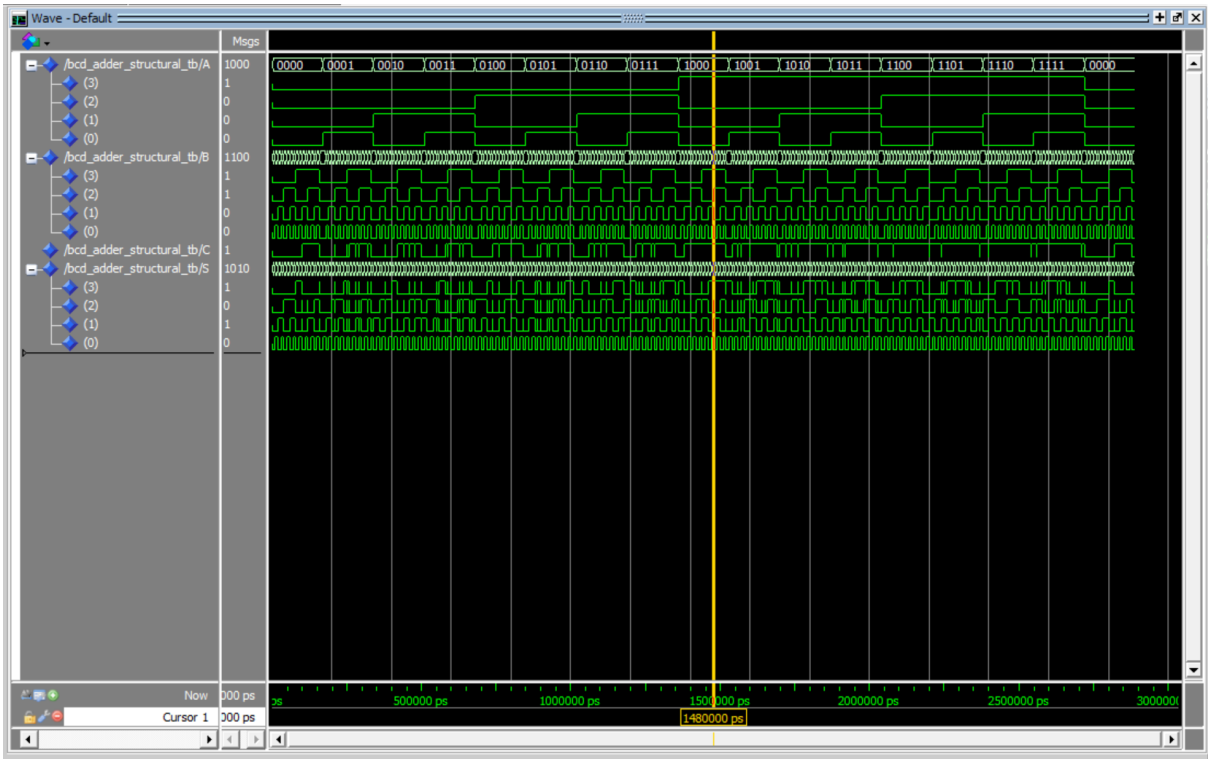


4-bit RCA behavioral test (full size):

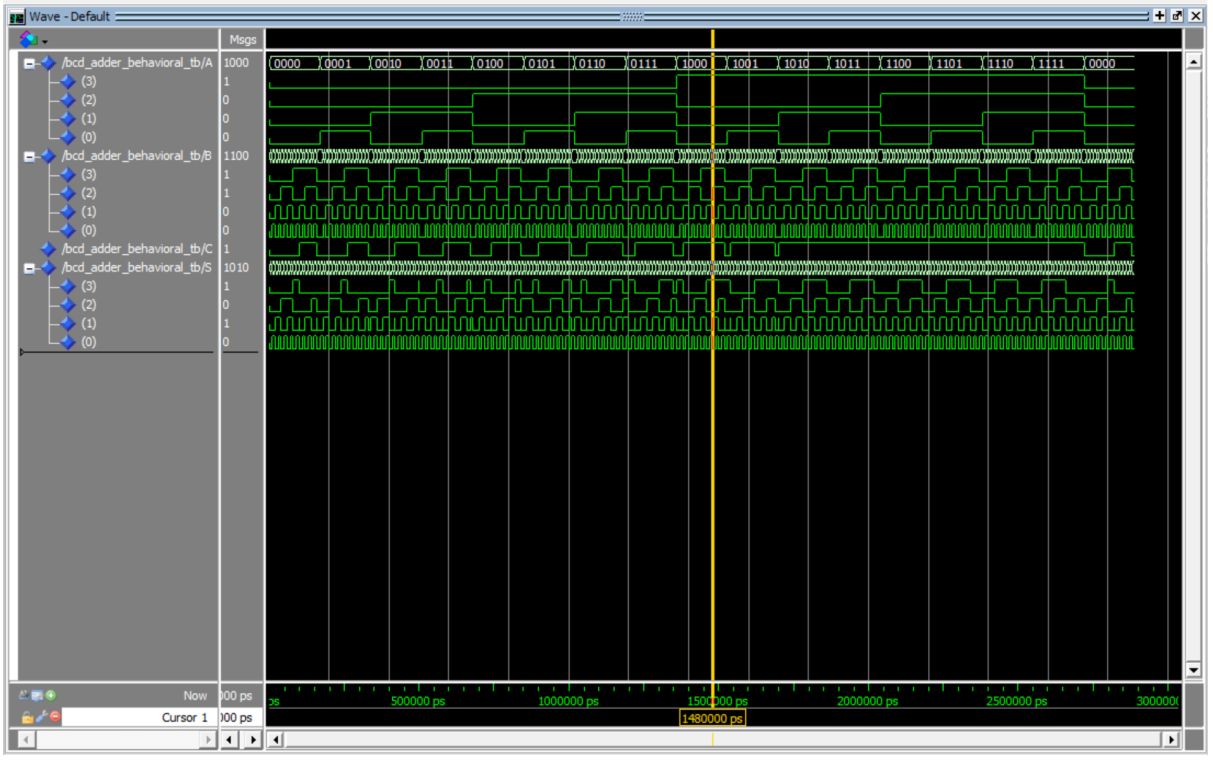


6) Representative simulation plots for the one-digit BCD adder circuits

One-digit BCD adder structural test:



One-digit BCD adder behavioral test:



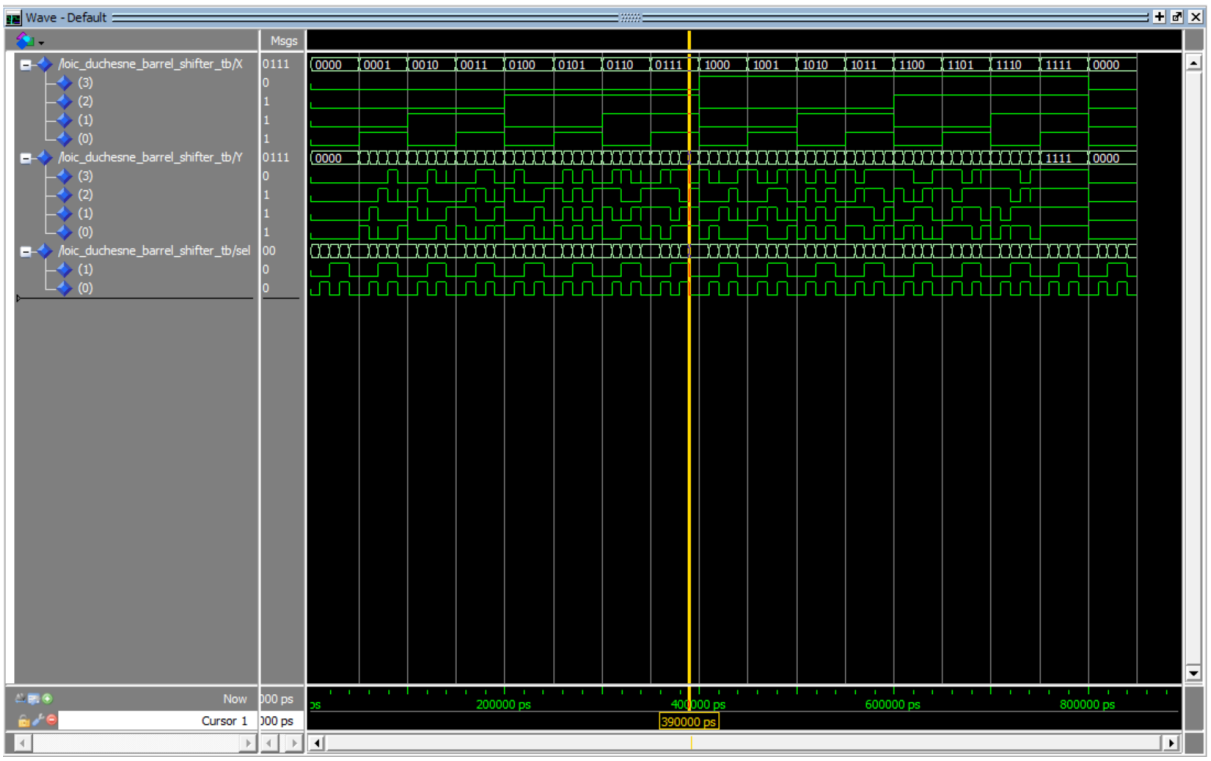
7) Logic & pin utilization

	4-bit circular barrel shifter		RCA		One-digit BCD adder	
	Structural	Behavioral	Structural	Behavioral	Structural	Behavioral
Logic Utilization (in ALMs)	5/32070 (< 1%)	5/32070 (< 1%)	4/32070 (< 1%)	3/32070 (< 1%)	7/32070 (< 1%)	5/32070 (< 1%)
Total pins	10/457 (2%)	10/457 (2%)	13/457 (3%)	13/457 (3%)	13/457 (3%)	13/457 (3%)

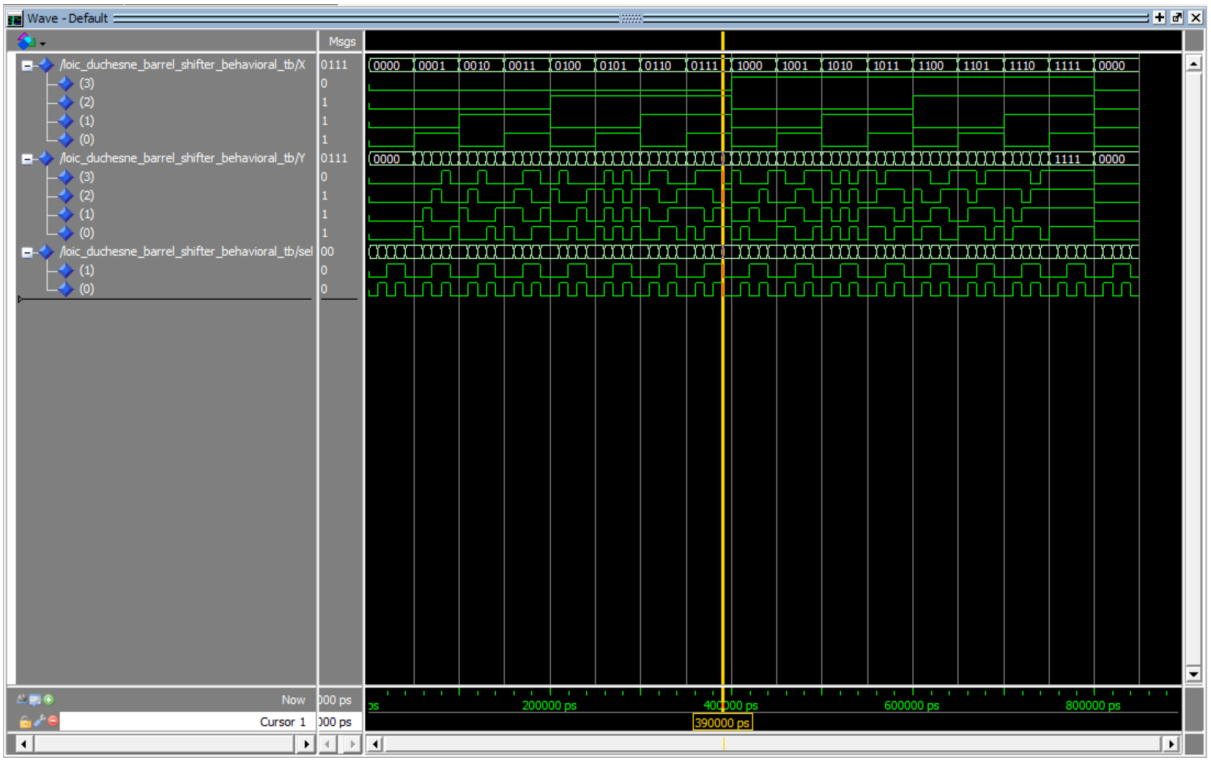
4- Additonal Pictures

*Note: See part 3 for all the other pictures of the results.

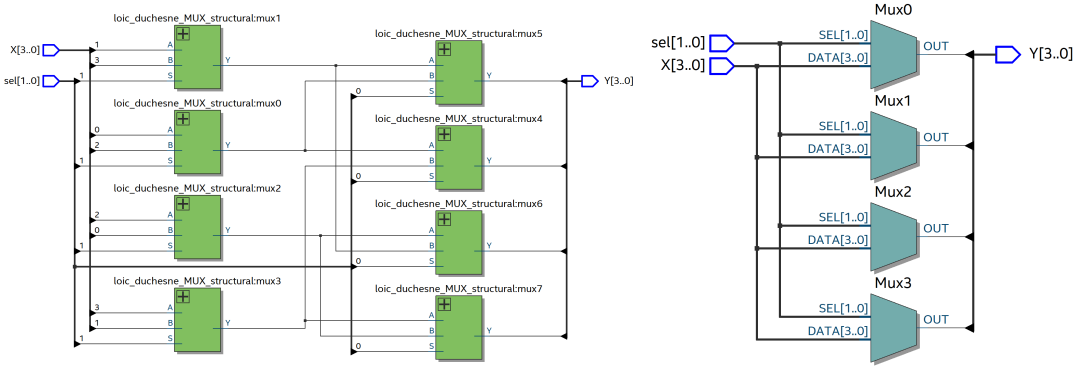
Barrel Shifter structural test for all possible values:



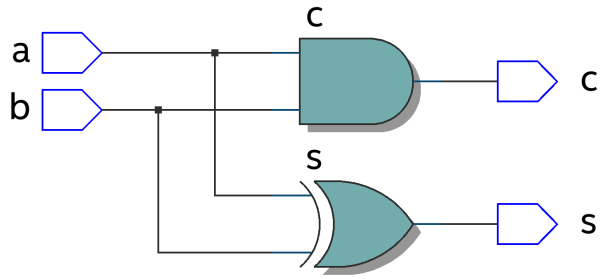
Barrel Shifter behavioral test for all possible values:



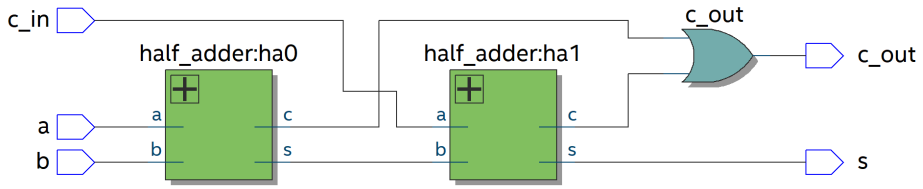
Barrel Shifter (left: Structural, right: Behavioral) RTL view:



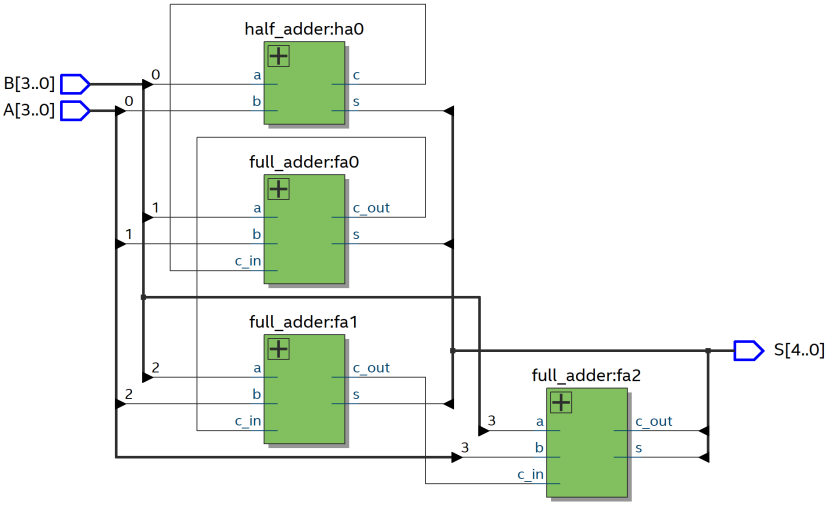
Half-Adder structural RTL view:



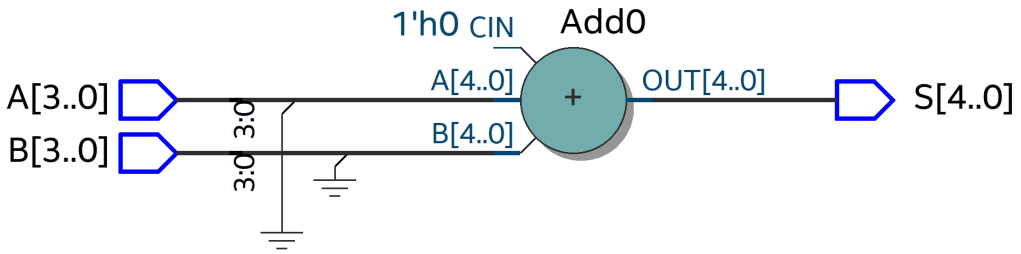
Full-Adder structural RTL view:



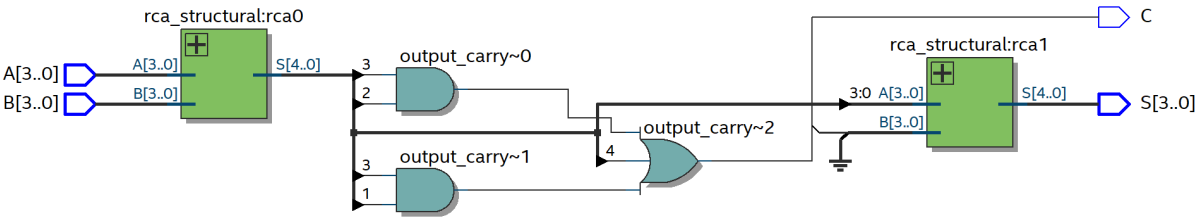
RCA structural RTL view:



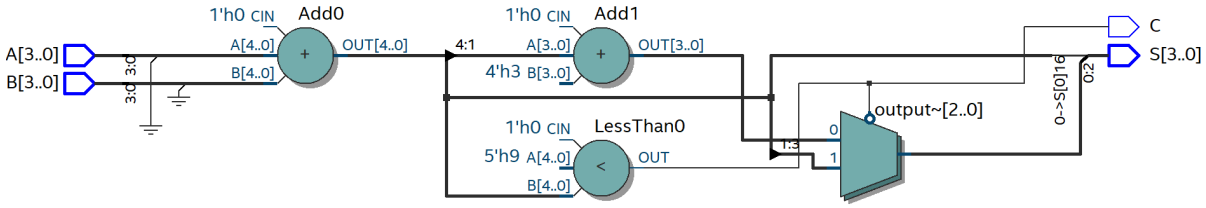
RCA behavioral RTL view:



BCD Adder structural RTL view:



BCD Adder behavioral RTL view:



5- Explanation of the results

The results of the test benches that we had matched our expectations for all cases. All of them were exhaustive tests except for the Barrel Shifter tests. For the Barrel Shifter, we first did tests for only one value of X “1011”, with all possible values of sel. We also did an exhaustive test for the Barrel Shifter which you can see in the additional pictures section.

For all the structural tests, you may notice on the waveforms random spikes that do not appear on the behavioral pictures. These spikes are noise caused by the structural blocks and are normal when building structural programs. They can be ignored when analyzing the results.

Another thing to note is the logic utilization differences for structural and behavioral models (more specifically the RCA & BCD Adders). These differences are likely due to the nature of those structural components requiring extra logic, while the behaviorals don't require as many sub-components processing. To develop, the structural components are using sub-components like rcas, half-adders & full-adders in their design, which in turn will require more logic utilization to process in addition to their main architecture.

For the explanations of how we computed the test benches, you may refer to the explanations in question 3.1).

6- Conclusion

In conclusion, this lab allowed us to get an idea of how the 4-bit circular barrel shifter works as well as the BCD-Adder. It also allowed us to simulate the Test Benches using ModelSim, We also discovered how to implement VHDL codes (& their respective components) in other VHDL files, to use it as a sub-component.

During this lab, we also had the chance to familiarize ourselves with the arithmetic operators for the first time in the behavioral implementation of the RCA. Finally, we learned how to write behavioral representation using the conditional signal assignments.