Lab 2
Ming Li Liu - 261226740
Guy Khairallah - 261135566
Georges Honein - 261139430

Introduction

In this assignment, we made vhdl code from a schematic rather than vhdl to schematic like last time. We were then introduced to the testbench writer that could generate a template for a given entity. Following the instructions, we implemented nested for loops for exhaustive testing. We implemented the 2-to-1 mux in structural and behavioral vhdl code, which we then used the testbench write to generate a template and completed an exhaustive testing for the two implementations. Using the two mux implementations, we made the 4-bit shifter and tested against an input X = 1110 and every possible shift (00, 01, 10, 11). See figures for more detail.

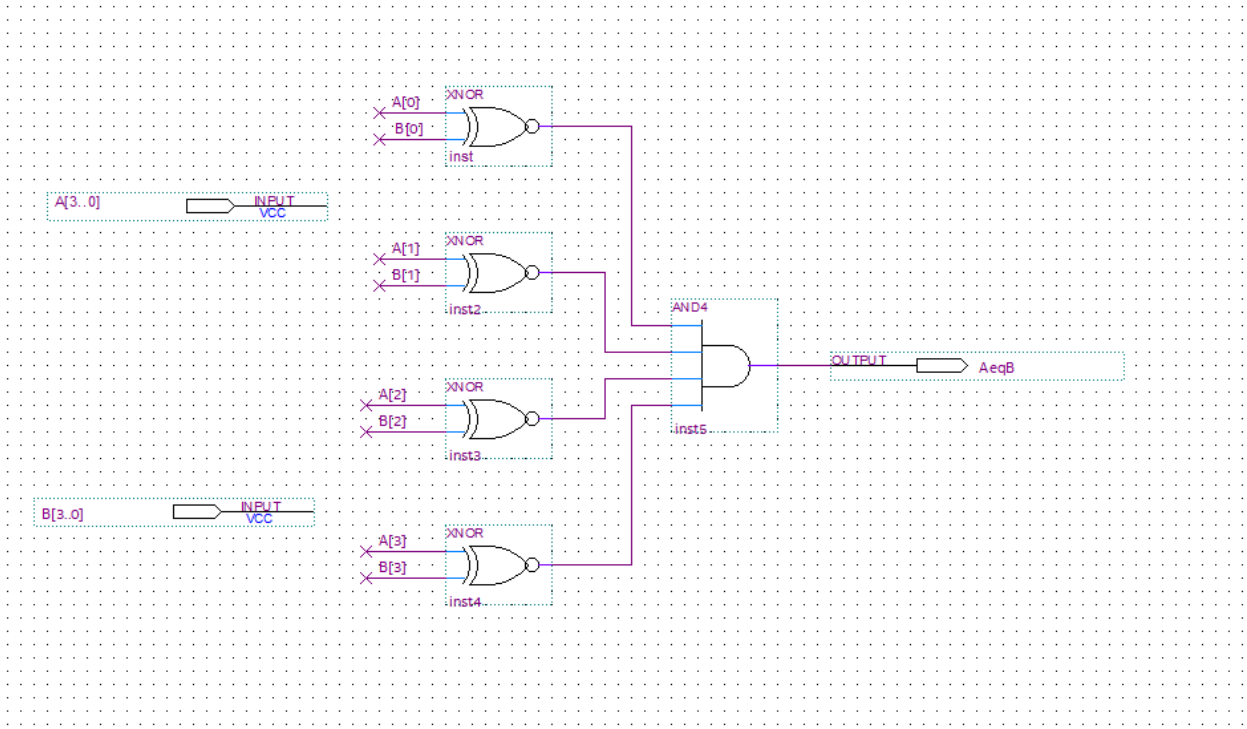


Fig 1. AeqB schematic representation. Inputs were mapped to the inputs of each XOR gate, and the gates's outputs were wired directly to an AND gate which gave the output for the entire function.

```
mingli_liu_2.vhd
1   library ieee;
  1 | use ieee.std_logic_1164.all;
  2
  3 library work;      ■ Library clause not necessary for current working library
  4
  5 entity mingli_liu_2 is
  6 | port (
  7 | |  a    : in    std_logic_vector(3 downto 0);
  8 | |  b    : in    std_logic_vector(3 downto 0);
  9 | |  aeqb : out   std_logic
 10 | );
 11 end entity mingli_liu_2;
 12
 13 architecture bdf_type of mingli_liu_2 is
 14 |
 15 |  signal synthesized_wire_0 : std_logic;
 16 |  signal synthesized_wire_1 : std_logic;
 17 |  signal synthesized_wire_2 : std_logic;
 18 |  signal synthesized_wire_3 : std_logic;
 19 |
 20 begin
 21 |
 22 |  synthesized_wire_0 <= NOT(a(0) xor b(0));
 23 |  synthesized_wire_1 <= NOT(a(1) xor b(1));
 24 |  synthesized_wire_2 <= NOT(a(2) xor b(2));
 25 |  synthesized_wire_3 <= NOT(a(3) xor b(3));
 26 |
 27 |  aeqb <= synthesized_wire_0 and synthesized_wire_1 and synthesized_wire_2 and synthesized_wire_3;
 28 |
 29 end architecture bdf_type;
```

Fig 2 (mingli_liu_vhd_tst.vhd). Generated vhdl code for AeqB schematic representation. The vhdl code was generated from the schematic representation. It uses AND, NOT, and XOR gates, so it is structural.

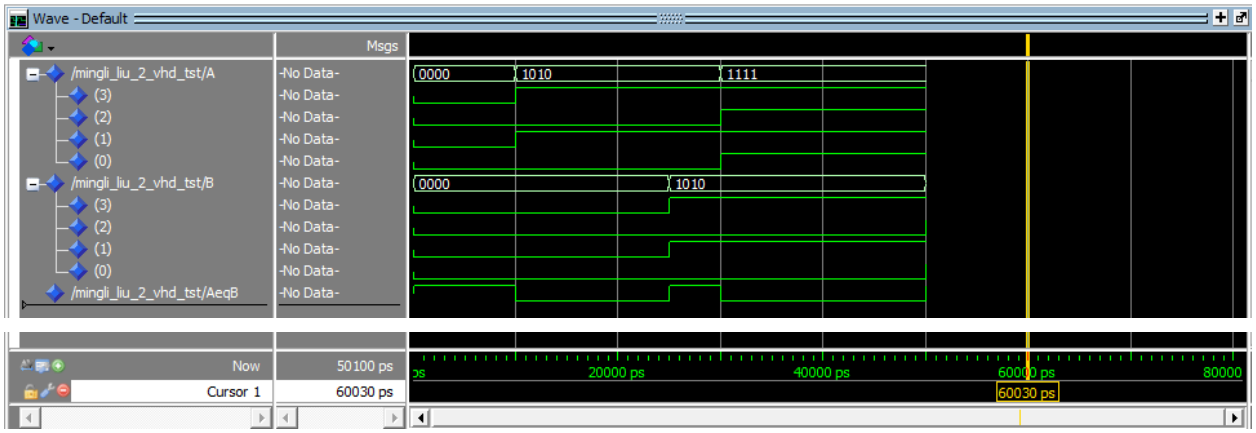


Fig 3. AeqB simple example simulation plot.

```
mingli_liu_2_vhd_…
1   library ieee;
  1 | use ieee.std_logic_1164.all;
  2 | use ieee.numeric_std.all;
  3
  4 entity mingli_liu_2_vhd_tst is
  5 end entity mingli_liu_2_vhd_tst;
  6
  7 architecture mingli_liu_2_arch of mingli_liu_2_vhd_tst is
  8 |
  9 |  -- constants
 10 |  -- signals
 11 |  signal a    : std_logic_vector(3 downto 0);
 12 |  signal aeqb : std_logic;
 13 |  signal b    : std_logic_vector(3 downto 0);
 14 |
 15 |  component mingli_liu_2 is
 16 |  | port (
 17 |  |   a    : in    std_logic_vector(3 downto 0);
 18 |  |   aeqb : out   std_logic;
 19 |  |   b    : in    std_logic_vector(3 downto 0)
 20 |  | );
 21 |  end component mingli_liu_2;
 22 |
 23 begin
 24 |
 25 |  i1 : component mingli_liu_2
 26 |  | port map (
 27 |  |   -- list connections between master ports and signals
 28 |  |   a    => a,
 29 |  |   aeqb => aeqb,
 30 |  |   b    => b
 31 |  | );
 32 |
 33 |  always : process is
 34 |  begin
 35 |  |
 36 |  |  a <= "0000";
 37 |  |  b <= "0000";
 38 |  |  wait for 10 ns;
 39 |  |  a <= "1010";
 40 |  |  wait for 15 ns;
 41 |  |  b <= "1100";
 42 |  |  wait for 5 ns;
 43 |  |  a <= "1111";
 44 |  |  wait for 20 ns;
 45 |  |  b <= "1111";
 46 |  |  wait; -- this waits forever...
 47 |  |
 48 |  end process always;
 49 |
 50 end architecture mingli_liu_2_arch;
```

Fig 4 (mingli_liu_2_vhd_tst.vhd). Simple introductory example testbench vhdl code for AeqB. We used the testbench template generator and the always block was provided by the instructions.

```
mingli_liu_2_vhd_…
1   library ieee;
  1   use ieee.std_logic_1164.all;
  2   use ieee.numeric_std.all;
  3
  4 entity mingli_liu_2_vhd_tst is
  5 end entity mingli_liu_2_vhd_tst;
  6
  7 architecture mingli_liu_2_arch of mingli_liu_2_vhd_tst is
  8
  9   -- constants
 10   -- signals
 11   signal a    : std_logic_vector(3 downto 0);
 12   signal aeqb : std_logic;
 13   signal b    : std_logic_vector(3 downto 0);
 14
 15   component mingli_liu_2 is
 16     port (
 17       a    : in    std_logic_vector(3 downto 0);
 18       aeqb : out   std_logic;
 19       b    : in    std_logic_vector(3 downto 0)
 20     );
 21   end component mingli_liu_2;
 22
 23 begin
 24
 25   i1 : component mingli_liu_2
 26     port map (
 27       -- list connections between master ports and signals
 28       a    => a,
 29       aeqb => aeqb,
 30       b    => b
 31     );
 32
 33   generate_test : process is
 34   begin
 35
 36     for i in 0 to 16 loop
 37
 38       a <= std_logic_vector(to_unsigned(i, 4));
 39
 40       for j in 0 to 16 loop
 41
 42         b <= std_logic_vector(to_unsigned(j, 4));
 43         wait for 10 ns;
 44
 45       end loop;
 46
 47     end loop;
 48
 49     wait;
 50
 51   end process generate_test;
 52
 53 end architecture mingli_liu_2_arch;
```

Fig 5 (mingli_liu_2_vhd_tst.vhd). Exhaustive vhdl testbench code for AeqB. We used the testbench template generator and the generate_test block was provided by the instructions. Using nested for loops to loop through all possible input values.

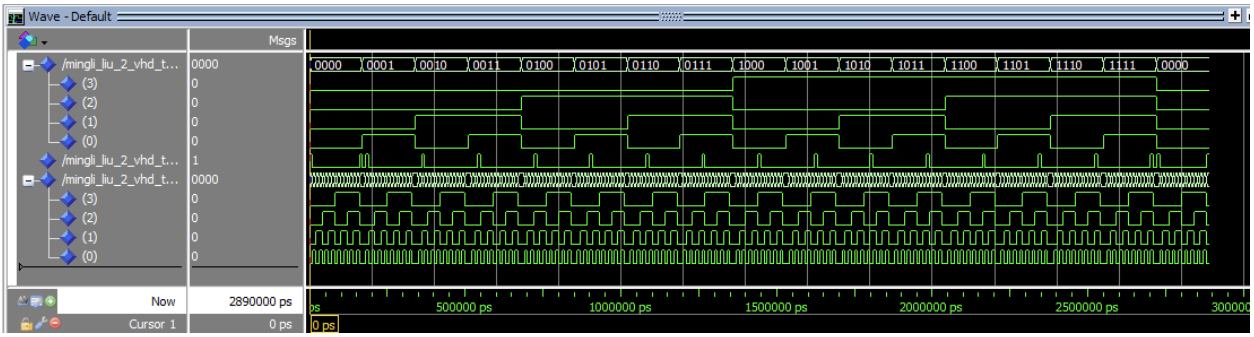
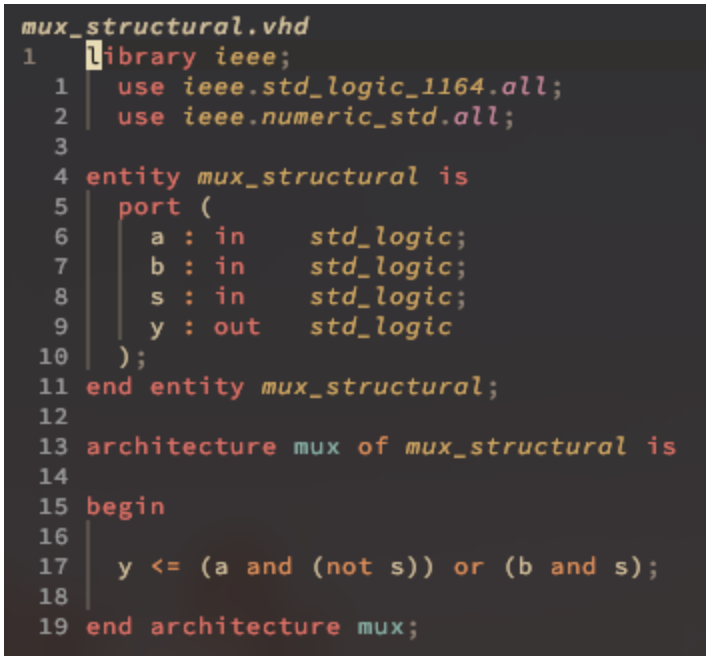


Fig 6. Simulation plot for exhaustive testing of AeqB.

```
mux_behavioral.vhd
1   library ieee;
  1   use ieee.std_logic_1164.all;
  2   use ieee.numeric_std.all;
  3
  4 entity mux_behavioral is
  5   port (
  6     a : in    std_logic;
  7     b : in    std_logic;
  8     s : in    std_logic;
  9     y : out   std_logic
 10   );
 11 end entity mux_behavioral;
 12
 13 architecture mux of mux_behavioral is
 14
 15 begin
 16
 17   with S select Y <=
 18     A when '0',
 19     B when '1',
 20     'X' when others;
 21
 22 end architecture mux;
```

Fig 7 (mux_behavioral.vhd). Behavioral vhdl code for mux implementation using “with select”.

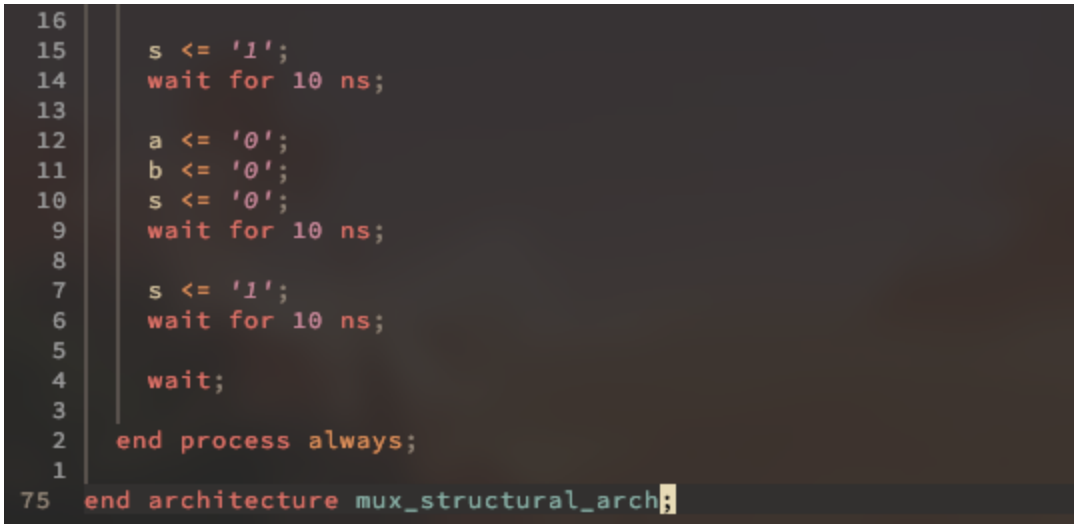


```vhdl
mux_structural.vhd
1   library ieee;
  1   use ieee.std_logic_1164.all;
  2   use ieee.numeric_std.all;
  3
  4 entity mux_structural is
  5   port (
  6     a : in    std_logic;
  7     b : in    std_logic;
  8     s : in    std_logic;
  9     y : out   std_logic
 10   );
 11 end entity mux_structural;
 12
 13 architecture mux of mux_structural is
 14
 15 begin
 16
 17   y <= (a and (not s)) or (b and s);
 18
 19 end architecture mux;
```

Fig 8 (mux_structural.vhd). Structural vhdl code for mux implementation using AND, OR, and NOT gates.

mux_structural.vht

```
37 library ieee;
36   use ieee.std_logic_1164.all;
35
34 entity mux_structural_tst is
33 end entity mux_structural_tst;
32
31 architecture mux_structural_arch of mux_structural_tst is
30
29   -- constants
28   -- signals
27   signal a : std_logic;
26   signal b : std_logic;
25   signal s : std_logic;
24   signal y : std_logic;
23
22   component mux_structural is
21     port (
20       a : in    std_logic;
19       b : in    std_logic;
18       s : in    std_logic;
17       y : out   std_logic
16     );
15   end component mux_structural;
14
13 begin
12
11   i1 : component mux_structural
10     port map (
 9       -- list connections between master ports and signals
 8       a => a,
 7       b => b,
 6       s => s,
 5       y => y
 4     );
 3
 2   always : process is
 1   begin
38
 1     a <= '1';
 2     b <= '1';
 3     s <= '0';
 4     wait for 10 ns;
 5
 6     s <= '1';
 7     wait for 10 ns;
 8
 9     a <= '1';
10     b <= '0';
11     s <= '0';
12     wait for 10 ns;
13
14     s <= '1';
15     wait for 10 ns;
16
17     a <= '0';
18     b <= '1';
19     s <= '0';
20     wait for 10 ns;
```



```
16
15      s <= '1';
14      wait for 10 ns;
13
12      a <= '0';
11      b <= '0';
10      s <= '0';
 9      wait for 10 ns;
 8
 7      s <= '1';
 6      wait for 10 ns;
 5
 4      wait;
 3
 2    end process always;
 1
75  end architecture mux_structural_arch;
```

Fig 9 (mux_structural.vht). Testbench vhdl code for structural implementation of mux. The always block goes through all possible input values. By enumerating all “ab” values (11, 10, 01, 00) and for each select 0 and 1.

mux_behavioral.vht

```
1   library ieee;
  1 | use ieee.std_logic_1164.all;
  2
  3 entity mux_behavioral_vhd_tst is
  4 end entity mux_behavioral_vhd_tst;
  5
  6 architecture mux_behavioral_arch of mux_behavioral_vhd_tst is
  7 |
  8 |  -- constants
  9 |  -- signals
 10 |  signal a : std_logic;
 11 |  signal b : std_logic;
 12 |  signal s : std_logic;
 13 |  signal y : std_logic;
 14 |
 15 |  component mux_behavioral is
 16 |    port (
 17 |      a : in    std_logic;
 18 |      b : in    std_logic;
 19 |      s : in    std_logic;
 20 |      y : out   std_logic
 21 |    );
 22 |  end component mux_behavioral;
 23 |
 24 begin
 25 |
 26 |  i1 : component mux_behavioral
 27 |    port map (
 28 |      -- list connections between master ports and signals
 29 |      a => a,
 30 |      b => b,
 31 |      s => s,
 32 |      y => y
 33 |    );
 34 |
 35 |  always : process is
 36 |  begin
 37 |  |
 38 |  |  a <= '1';
 39 |  |  b <= '1';
 40 |  |  s <= '0';
 41 |  |  wait for 10 ns;
 42 |  |
 43 |  |  s <= '1';
 44 |  |  wait for 10 ns;
 45 |  |
 46 |  |  a <= '1';
 47 |  |  b <= '0';
 48 |  |  s <= '0';
 49 |  |  wait for 10 ns;
 50 |  |
 51 |  |  s <= '1';
 52 |  |  wait for 10 ns;
 53 |  |
 54 |  |  a <= '0';
 55 |  |  b <= '1';
 56 |  |  s <= '0';
 57 |  |  wait for 10 ns;
```

```
    s <= '1';
    wait for 10 ns;

    a <= '0';
    b <= '0';
    s <= '0';
    wait for 10 ns;

    s <= '1';
    wait for 10 ns;

    wait;

  end process always;

end architecture mux_behavioral_arch;
```

Fig 10 (mux_behavioral.vht). Testbench code for behavioral implementation of mux. The always block goes through all possible input values. By enumerating all “ab” values (11, 10, 01, 00) and for each select 0 and 1.

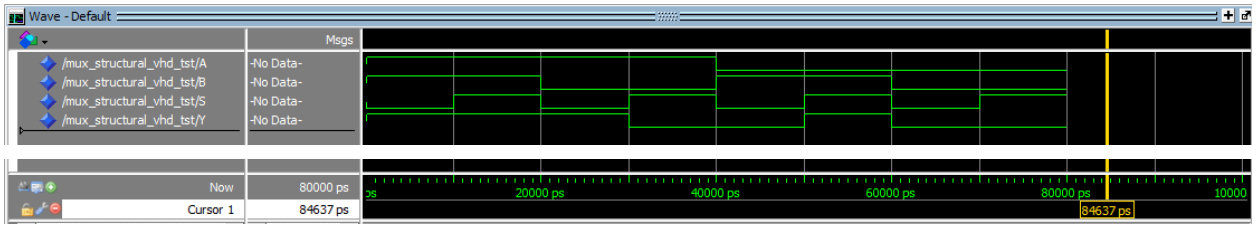


Fig 11. Simulation plot for structural implementation of mux.

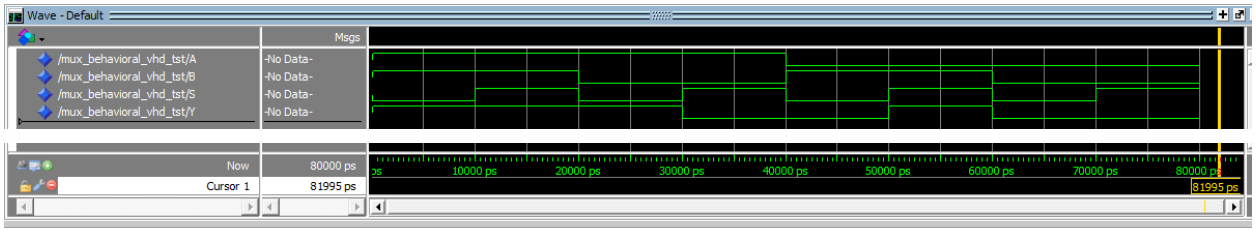


Fig 12. Simulation plot for behavioral implementation of mux.

```
circular_barrel_s…
1   library ieee;
  1 | use ieee.std_logic_1164.all;
  2 | use ieee.numeric_std.all;
  3
  4 entity circular_barrel_shifter_structural is
  5 | port (
  6 | | x   : in    std_logic_vector(3 downto 0);
  7 | | sel : in    std_logic_vector(1 downto 0);
  8 | | y   : out   std_logic_vector(3 downto 0)
  9 | );
 10 end entity circular_barrel_shifter_structural;
 11
 12 architecture circular_barrel_shifter of circular_barrel_shifter_structural is
 13 | component mux_structural is
 14 | | port (
 15 | | | a : in    std_logic;
 16 | | | b : in    std_logic;
 17 | | | s : in    std_logic;
 18 | | | y : out   std_logic
 19 | | );
 20 | end component mux_structural;
 21
 22 | signal m1 : std_logic;
 23 | signal m2 : std_logic;
 24 | signal m3 : std_logic;
 25 | signal m4 : std_logic;
 26
 27 begin
 28 | i1 : component mux_structural port map (x(0), x(2), sel(1), m1);
 29 | i2 : component mux_structural port map (x(1), x(3), sel(1), m2);
 30 | i3 : component mux_structural port map (x(2), x(0), sel(1), m3);
 31 | i4 : component mux_structural port map (x(3), x(1), sel(1), m4);
 32
 33 | i5 : component mux_structural port map (m1, m4, sel(0), y(0));
 34 | i6 : component mux_structural port map (m2, m1, sel(0), y(1));
 35 | i7 : component mux_structural port map (m3, m2, sel(0), y(2));
 36 | i8 : component mux_structural port map (m4, m3, sel(0), y(3));
 37 end architecture circular_barrel_shifter;
```

Fig 13 (circular_barrel_shifter_structural.vhd). Vhdl code for structural implementation of 4 bit shifter. It uses the structural implementation of mux in Fig 8 as a component, then maps out all the inputs to outputs for every mux instance in the 4 bit shifter.

```
circular_barrel_s…
1   library ieee;
  1   use ieee.std_logic_1164.all;
  2   use ieee.numeric_std.all;
  3
  4 entity circular_barrel_shifter_behavioral is
  5   port (
  6     x   : in    std_logic_vector(3 downto 0);
  7     sel : in    std_logic_vector(1 downto 0);
  8     y   : out   std_logic_vector(3 downto 0)
  9   );
 10 end entity circular_barrel_shifter_behavioral;
 11
 12 architecture circular_barrel_shifter of circular_barrel_shifter_behavioral is
 13   component mux_behavioral is
 14     port (
 15       a : in    std_logic;
 16       b : in    std_logic;
 17       s : in    std_logic;
 18       y : out   std_logic
 19     );
 20   end component mux_behavioral;
 21
 22   signal m1 : std_logic;
 23   signal m2 : std_logic;
 24   signal m3 : std_logic;
 25   signal m4 : std_logic;
 26
 27 begin
 28   i1 : component mux_behavioral port map (x(0), x(2), sel(1), m1);
 29   i2 : component mux_behavioral port map (x(1), x(3), sel(1), m2);
 30   i3 : component mux_behavioral port map (x(2), x(0), sel(1), m3);
 31   i4 : component mux_behavioral port map (x(3), x(1), sel(1), m4);
 32
 33   i5 : component mux_behavioral port map (m1, m4, sel(0), y(0));
 34   i6 : component mux_behavioral port map (m2, m1, sel(0), y(1));
 35   i7 : component mux_behavioral port map (m3, m2, sel(0), y(2));
 36   i8 : component mux_behavioral port map (m4, m3, sel(0), y(3));
 37 end architecture circular_barrel_shifter;
```

Fig 14 (circular_barrel_shifter_behavioral.vhd). Vhdl code for behavioral implementation of 4 bit shifter. It uses the behavioral implementation of mux in Fig 7 as a component, then maps out all the inputs to outputs for every mux instance in the 4 bit shifter.

```
circular_barrel_s…
1   library ieee;
  1   use ieee.std_logic_1164.all;
  2   use ieee.numeric_std.all;
  3
  4 entity circular_barrel_shifter_structural_vhd_tst is
  5 end entity circular_barrel_shifter_structural_vhd_tst;
  6
  7 architecture arch of circular_barrel_shifter_structural_vhd_tst is
  8
  9   -- constants
 10   -- signals
 11   signal sel : std_logic_vector(1 downto 0);
 12   signal x   : std_logic_vector(3 downto 0);
 13   signal y   : std_logic_vector(3 downto 0);
 14
 15   component circular_barrel_shifter_structural is
 16     port (
 17       sel : in    std_logic_vector(1 downto 0);
 18       x   : in    std_logic_vector(3 downto 0);
 19       y   : out   std_logic_vector(3 downto 0)
 20     );
 21   end component circular_barrel_shifter_structural;
 22
 23 begin
 24
 25   i1 : component circular_barrel_shifter_structural
 26     port map (
 27       -- list connections between master ports and signals
 28       sel => sel,
 29       x   => x,
 30       y   => y
 31     );
 32
 33   always : process is
 34   begin
 35
 36     x <= "1110";
 37
 38     for j in 0 to 4 loop
 39
 40       sel <= std_logic_vector(to_unsigned(j, 2));
 41       wait for 10 ns;
 42
 43     end loop;
 44
 45     wait;
 46
 47   end process always;
 48
 49 end architecture arch;
```

Fig 15 (circular_barrel_shifter_structural.vht). Testbench code for structural implementation of 4 bit shifter. It uses the structural component defined in Fig 13. The input X is 1110 and it test all possible shifts using a for loop that loops through from 00 to 11.

```
circular_barrel_s…    circular_barrel_s…
1   library ieee;
  1   use ieee.std_logic_1164.all;
  2   use ieee.numeric_std.all;
  3
  4 entity circular_barrel_shifter_behavioral_vhd_tst is
  5 end entity circular_barrel_shifter_behavioral_vhd_tst;
  6
  7 architecture arch of circular_barrel_shifter_behavioral_vhd_tst is
  8
  9   -- constants
 10   -- signals
 11   signal sel : std_logic_vector(1 downto 0);
 12   signal x   : std_logic_vector(3 downto 0);
 13   signal y   : std_logic_vector(3 downto 0);
 14
 15   component circular_barrel_shifter_behavioral is
 16     port (
 17       sel : in    std_logic_vector(1 downto 0);
 18       x   : in    std_logic_vector(3 downto 0);
 19       y   : out   std_logic_vector(3 downto 0)
 20     );
 21   end component circular_barrel_shifter_behavioral;
 22
 23 begin
 24
 25   i1 : component circular_barrel_shifter_behavioral
 26     port map (
 27       -- list connections between master ports and signals
 28       sel => sel,
 29       x   => x,
 30       y   => y
 31     );
 32
 33   always : process is
 34   begin
 35
 36     x <= "1110";
 37
 38     for j in 0 to 4 loop
 39
 40       sel <= std_logic_vector(to_unsigned(j, 2));
 41       wait for 10 ns;
 42
 43     end loop;
 44
 45     wait;
 46
 47   end process always;
 48
 49 end architecture arch;
```

Fig 16 (circular_barrel_shifter_behavioral.vht). Testbench code for behavioral implementation of 4 bit shifter. It uses the behavioral component defined in Fig 14. The input X is 1110 and it test all possible shifts using a for loop that loops through from 00 to 11.

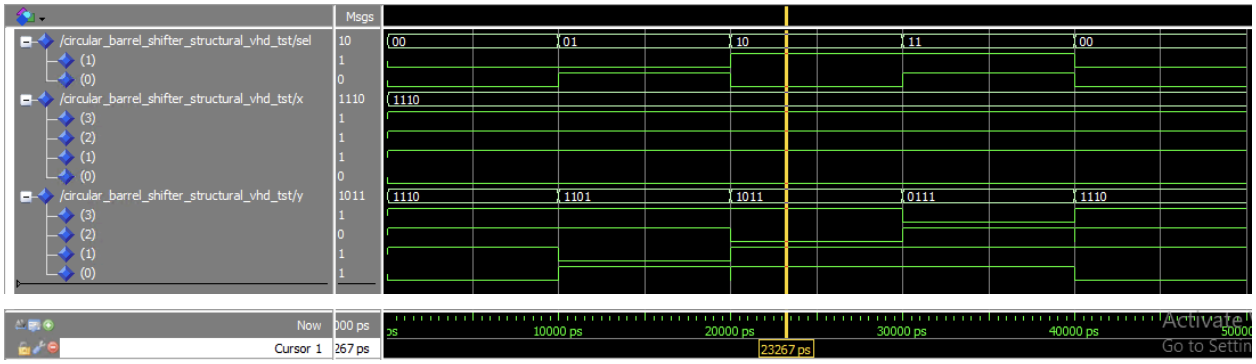


Fig 17. Simulation plot for structural implementation of 4 bit shifter with X = 1110.

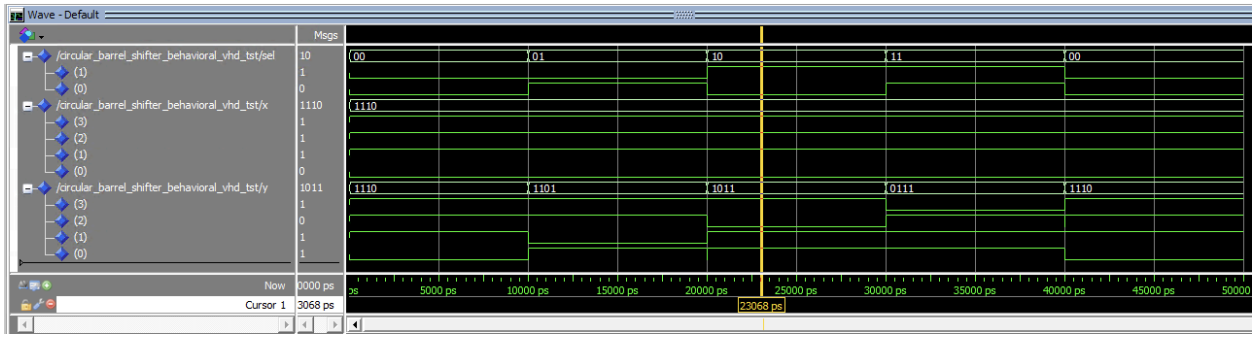


Fig 18. Simulation plat for behavioral implementation of 4 bit shifter with X = 1110.

8 Questions:

1) **Briefly explain your VHDL code implementation of all circuits.**
   See Fig 2, 4, 5, 7, 8, 9, 10 ,13, 14, 15, 16 for vhdl code and explanation.

2) **Report the number of pins and logic modules used.**

	AeqB	**2-1 MUX**		**4-bit circular barrel shifter**	
	schematic	**structural**	**behavioral**	**structural**	**behavioral**
Logic utilization (in ALMs)	2	1	1	5	5
Total pins	9	4	4	10	10

3) **Show a representative simulation plot for the introductory testing example. You can simply include a snapshot from the waveform that you obtained from ModelSim. In order to fully capture all the signals from the waveform, you can adjust the display range using the magnifier icons.**

See Fig 3.

4) **Show representative simulation plots for the exhaustive test.**

   See Fig 6.

5) **Show representative simulation plots of the 2-to-1 MUX circuits for all the possible input values.**

   See Fig 11 and Fig 12.

6) **Show representative simulation plots of the 4-bit circular shift register for a given input sequence.**

   See Fig 17 and Fig 18 for simulation plots of the 4 bit shifter for X = 1110.

Conclusion

We learned that it was possible to convert schematic representation to vhdl code and how to generate a template for testbench code using the Testbench Template Writer. All testbenches confirmed the expected behavior of the code they were responsible for.