1. Introduction

VHDL supports 3 kinds of modelling styles: dataflow, structural and behavioural. Dataflow and structural modelling style is leveraged for combinatorial circuits while behavioural modelling is used for both combinatorial and sequential. This lab aims to construct a digital 2 to 1 mux using all 3 modelling techniques to observe its feasibility when scaled to a 2 bit, 2 to 1 mux.

2. 2 bit, 2 to 1 mux

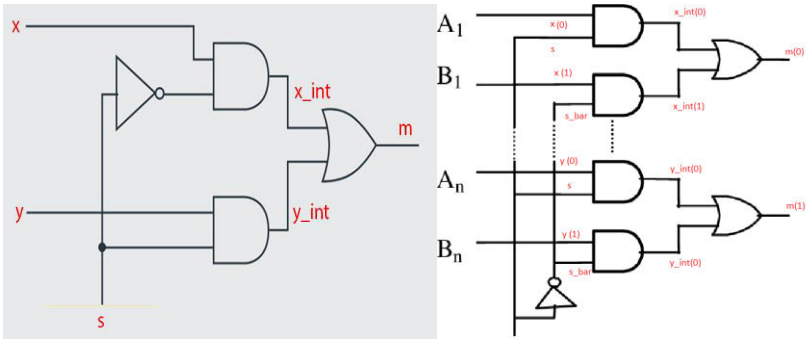


Figure 2.1: 2 to 1 mux logic gate (left), 2 bit 2 to 1 mux logic gate (right)

The purpose of a 2 to 1 mux is to allow a toggle to be implemented between 2 logic inputs. Hence allowing for decisions to be made; if **s** is equal to 0, the output would be **x**, whereas if **s** is equal to 1, the output would be **y**. Additionally, a 2 bit version of a 2 to 1 mux can be scaled as indicated in figure 2.1.

3. Dataflow modelling design

Dataflow modelling of 2 to 1 mux	Assigning inputs and outputs to physical ports of the FPGA	Schematic of the 2 to 1 mux

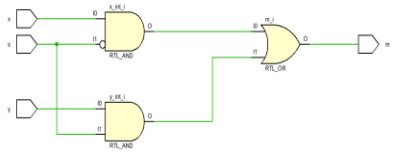
`architecture Behavioral of gates is`
 	
`signal sbar : STD_LOGIC;`
`signal x_int: STD_LOGIC;`
`signal y_int: STD_LOGIC;`
 	
`begin`
 	
`sbar <= not s;`
`x_int <= x and sbar;`
`y_int <= y and s;`
`m <= x_int or y_int;`
 	
`end Behavioral;`
	`set_property PACKAGE_PIN L16 [get_ports x]`
	`set_property IOSTANDARD LVCMOS33 [get_ports x]`
	`set_property PACKAGE_PIN J15 [get_ports y]`
	`set_property IOSTANDARD LVCMOS33 [get_ports y]`
	`set_property PACKAGE_PIN M13 [get_ports s]`
	`set_property IOSTANDARD LVCMOS33 [get_ports s]`
	
	`set_property PACKAGE_PIN H17 [get_ports m]`
	`set_property IOSTANDARD LVCMOS33 [get_ports m]`
		

Table 1: Dataflow modelling of 2 to 1 mux.

Dataflow modelling style's entity is described using concurrent statements (statements which get executed at the same time) without defining the structure of the design. As shown in the first column of Table 1, the concurrent statements will be assigned to signals. Additionally, these signals will create an internal netlist inside the FPGA, routing the AND gate outputs to the OR gate inputs,

allowing concurrent execution to occur. Consequently, the inputs and outputs of the digital 2 to 1 mux will be routed to the switches and LEDs of the FPGA respectively as shown in the second column of Table 1. The circuit can be verified by the RTL schematic as shown in the 3rd column since it matches the mux logic gate in Figure 2.1.

To observe the scalability of dataflow modelling design, a 2 bit, 2 to 1 mux was programmed, routed, and verified as shown in Table 2. This process was similar to Table 1 however, additional routing and signals were required to allow for a 2 bit input.

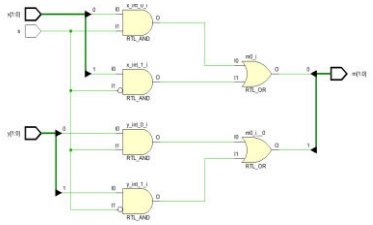
Dataflow modelling of the 2 bit, 2 to 1 mux	Assigning inputs and outputs to physical ports of the FPGA	Schematic of the 2 bit, 2 to 1 mux
50 x_int (0) <= x (0) and s; 51 x_int (1) <= x (1) and sbar; 52 y_int (0) <= y (0) and s; 53 y_int (1) <= y (1) and sbar; 54 55 m (0) <= x_int (0) or y_int (0); 56 57 m (1) <= x_int (1) or y_int (1);	1 set_property PACKAGE_PIN R2 [get_ports {x[1]}] 2 set_property IOSTANDARD LVCMOS33 [get_ports {x[1]}] 3 set_property PACKAGE_PIN T1 [get_ports {x[0]}] 4 set_property IOSTANDARD LVCMOS33 [get_ports {x[0]}] 5 set_property PACKAGE_PIN U1 [get_ports {y[1]}] 6 set_property IOSTANDARD LVCMOS33 [get_ports {y[1]}] 7 set_property PACKAGE_PIN W2 [get_ports {y[0]}] 8 set_property IOSTANDARD LVCMOS33 [get_ports {y[0]}] 9 set_property PACKAGE_PIN R3 [get_ports s] 10 set_property IOSTANDARD LVCMOS33 [get_ports s] 11 12 set_property PACKAGE_PIN L1 [get_ports {m[1]}] 13 set_property IOSTANDARD LVCMOS33 [get_ports {m[1]}] 14 set_property PACKAGE_PIN P1 [get_ports {m[0]}] 15 set_property IOSTANDARD LVCMOS33 [get_ports {m[0]}]	

Table 2: Dataflow modelling of 2 bit, 2 to 1 mux.

```
48  sbar <= not s after 3 ns;
49  x_int (0) <= x (0) and s after 3 ns;
50  x_int (1) <= x (1) and sbar after 3 ns;
51  y_int (0) <= y (0) and s after 3 ns;
52  y_int (1) <= y (1) and sbar after 3 ns;
53
54
55
56  m (0) <= x_int(0) or x_int (1) after 3 ns;
57  m (1) <= y_int(0) or y_int (1) after 3 ns;
```

Figure 3.1: example of a dataflow modelling delay

This style of modelling typically favours simple design or as a function that is part of a structural modelling design's component. Due to the concurrent execution of this modelling style, any delays appended onto each statement (example can be seen in Figure 3.1) will be executed synchronously as shown in figure 3.2, as the delay was 3 ns, it has negligible effect on the timing of the executions.

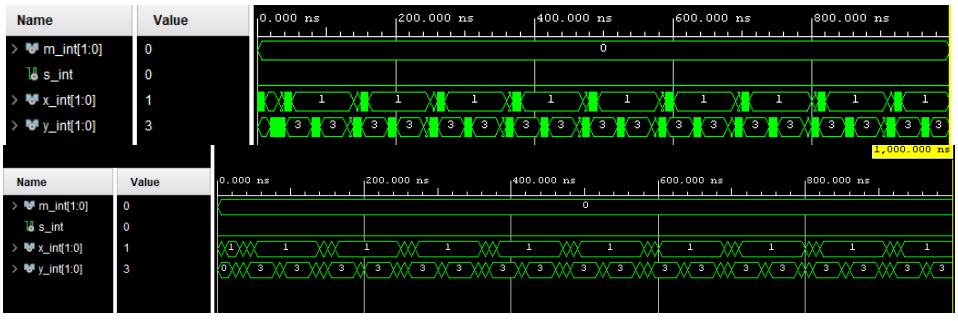


Figure 3.1: dataflow modelling of 2 bit 2 to 1 mux; no delays (top), 3ns delay (bottom)

4. Structural modelling design

Structural modelling uses dataflow modelling style and aims to construct components such as ICs or muxes. This hierarchical modelling style grants organisation when programming in VHDL as suggested in Figure 4.1. Components require declaration between the **Architecture** and the **begin section**. This will allow the components to be instantiated after the **begin section** as shown in Figure 4.2 where a 2 bit, 2 to 1 mux circuit is generated using the (and_or_gate).

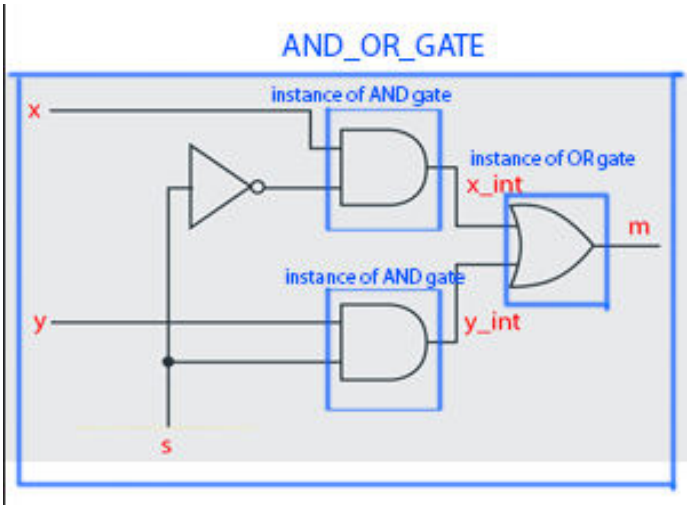


Figure 4.1: visual example of a structurally modelled mux

```
34  entity muxcomp is
35      Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
36             y : in STD_LOGIC_VECTOR (1 downto 0);
37             s : in STD_LOGIC;
38             m : out STD_LOGIC_VECTOR (1 downto 0));
39  end muxcomp;
40
41   architecture gates of muxcomp is
42
43      component and_or_gate --calling "function"
44      Port ( bits_int : in STD_LOGIC_VECTOR(1 downto 0);
45             s_int : in STD_LOGIC;
46             o_int : out STD_LOGIC);end component;
47
48      Signal a_int : STD_LOGIC;
49      Signal b_int : STD_LOGIC;
50      Signal sbar : STD_LOGIC;
51
52  begin
```

```
52  begin
53
54  sbar <= not s;
55
56  and_or_comp_1 : and_or_gate
57  port map (
58      bits_int => x (1 downto 0),
59      s_int => s,
60      o_int => m (0)
61   );
62
63  and_or_comp_2 : and_or_gate
64  port map (
65      bits_int => y (1 downto 0),
66      s_int => s,
67      o_int => m (1)
68   );
69
70
71    ---- 2nd half of mux
```

Figure 4.2: declaration of components between **Architecture** and **begin section** (left), instantiation of components after the **begin section.**

For organisational purposes, components can be declared within components as shown in Table 3. This versatile modelling style allows for a hierarchical approach in which the declared inputs in **entity ports** are fed into a component (and_or_gate) then inputs are processed via internal logic (and gates/or gates). Consequently, these are routed inside the FPGA, binding switches to inputs and LEDs to outputs. Using the RTL simulation, a schematic is produced that resembles a 2 bit, 2 to 1 mux in a hierarchical manner, as illustrated in Figure 4.3.

And or gate code	And gate	Or gate

```vhdl
entity and_or_gate is
    Port ( bits_int : in STD_LOGIC_VECTOR(1 downto 0);
           s_int : in STD_LOGIC;
           o_int : out STD_LOGIC);
end and_or_gate;

architecture Behavioral of and_or_gate is

component andgate --calling "function"
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);end component;
component orgate --calling "function"
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);end component;

Signal a_int : STD_LOGIC;
Signal b_int : STD_LOGIC;
signal c_int : STD_LOGIC;
Signal sbar : STD_LOGIC;
```

```vhdl
begin

sbar <= not s_int;

and_comp_1 : andgate
port map (
    i0 => bits_int(0),
    i1 => s_int,
    o => a_int
);

and_comp_2 : andgate
port map (
    i0 => bits_int(1),
    i1 => sbar,
    o => b_int
);

or_comp_1 : orgate
port map(
    i0 => a_int,
    i1 => b_int,
    o => o_int
    );
```

```vhdl
entity andgate is
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);
end andgate;

architecture Behavioral of andgate is

begin

o <= i0 and i1;

end Behavioral;
```

```vhdl
entity orgate is
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);
end orgate;

architecture Behavioral of orgate is

begin

o <= i0 or i1;

end Behavioral;
```

Table 3: Component structure used in Figure 4.2 to create a 2 bit, 2 to 1 mux.

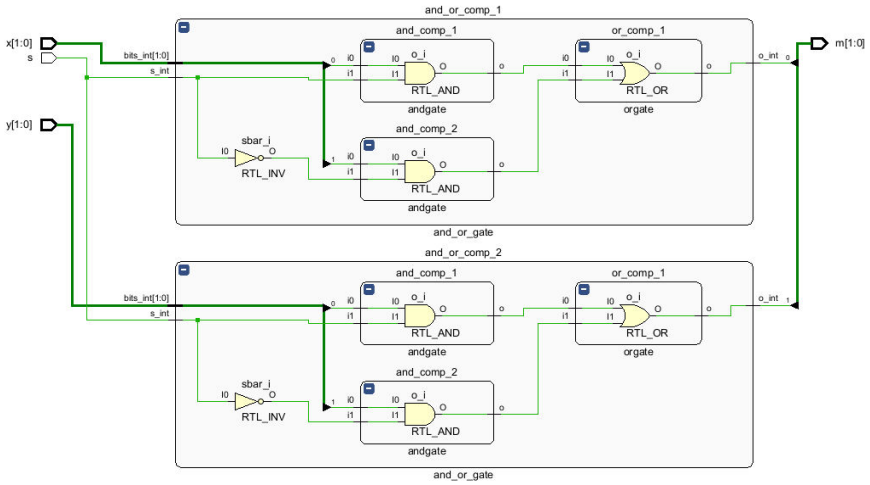


Figure 4.3: RTL schematic of the 2 bit, 2 to 1 mux generated via structural modelling style.

Structural modelling style may localise bugs hence making debugging easier in the future. Additionally, scalability proves to be an advantage for structural modelling style as complex components can be synthesised with an array of logic gates which can be called into the circuit later for efficiency. Furthermore, it also allows multiple components to be made in a large scale project.

5. Behavioural modelling design

Behavioural modelling style is very similar to dataflow modelling, but a behavioural model allows both combinatorial and sequential circuits to be designed. As a result, behavioural modelling designs allow for conditional statements. This modelling technique can be used to create a 2 to 1 mux by assigning conditional statements to determine the output, as shown in Figure 5.1, allowing for sequential execution.

```
entity mux_behave is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : in STD_LOGIC;
           m : out STD_LOGIC);
end mux_behave;

architecture Behavioral of mux_behave is

begin
    process (x,y, s)
    begin
        if s = '0' then
            m <= x;
        else
            m <= y;
        end if;
    end process;
end Behavioral;
```

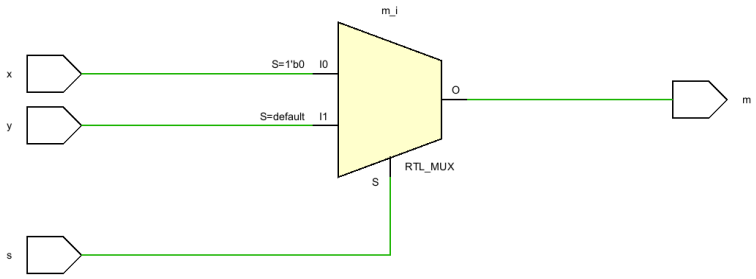


Figure 5.1: RTL schematic of a 2 to 1 mux using behavioural modelling style.

Due to the simplicity of the behavioural modelling style, scaling a 2 to 1 mux to a 2 bit, 2 to 1 mux only requires alteration inside the **entity port** where STD_LOGIC turns to STD_LOGIC_VECTOR as shown in Figure 5.2. As a result, larger inputs will have fewer VHDL code modifications.

```
entity mux_behave is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           y : in STD_LOGIC_VECTOR (1 downto 0);
           s : in STD_LOGIC;
           m : out STD_LOGIC_VECTOR (1 downto 0));
end mux_behave;

architecture Behavioral of mux_behave is

begin
    process(x, y, s)
    begin
        if s = '0' then
            m <= x;
        else
            m <= y;
        end if;
    end process;

end Behavioral;
```

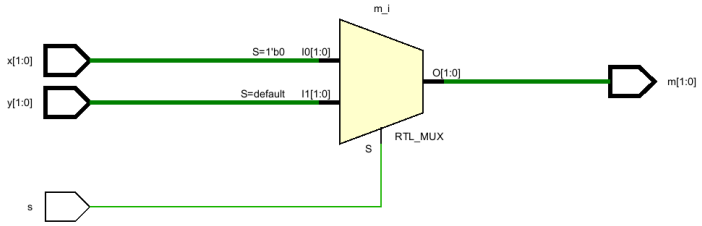


Figure 5.2: RTL schematic of a 2 bit, 2 to 1 mux using behavioural modelling style.

6. Conclusion

The variety of modelling styles available to VHDL greatly benefits large-scale projects while maintaining simplicity. A structural model would be useful for organising large circuit designs. Additionally, systems that require more sequential execution of instruction would greatly benefit from behavioural modelling. For projects that require concurrent execution, Dataflow modelling would be useful. Using these three main modelling styles in conjunction can increase firmware programming efficiency and productivity while achieving the same results as shown in Figure 6.

Figure 6: result of a 2 bit, 2 to 1 mux, in dataflow, structural and behavioural

Appendix

Part 2

2 to 1 mux dataflow modelling

```
34  entity gates is
35      Port ( x : in STD_LOGIC;
36             y : in STD_LOGIC;
37             s: in STD_LOGIC;
38             m: out STD_LOGIC
39             );
40  end gates;
41
42  architecture Behavioral of gates is
43
44  signal sbar : STD_LOGIC;
45  signal x_int: STD_LOGIC;
46  signal y_int: STD_LOGIC;
47
48
49
50  begin
51
52  sbar <= not s;
53  x_int <= x and sbar;
54  y_int <= y and s;
55  m <= x_int or y_int;
56
57  end Behavioral;
```

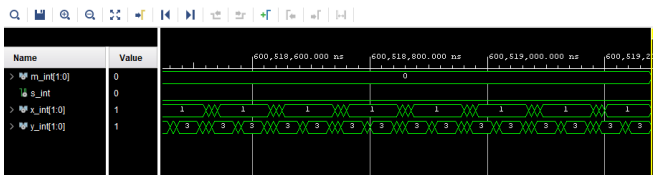
2 bit 2 to 1 mux dataflow modelling

```
34  entity mux is
35      Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
36             y : in STD_LOGIC_VECTOR (1 downto 0);
37             s : in STD_LOGIC;
38             m : out STD_LOGIC_VECTOR (1 downto 0));
39  end mux;
40
41  architecture Behavioral of mux is
42  signal sbar : STD_LOGIC ;
43  signal x_int : STD_LOGIC_VECTOR (1 downto 0);
44  signal y_int : STD_LOGIC_VECTOR (1 downto 0);
45
46  begin
47
48  sbar <= not s;
49  x_int (0) <= x (0) and s;
50  x_int (1) <= x (1) and sbar;
51  y_int (0) <= y (0) and s;
52  y_int (1) <= y (1) and sbar;
53
54
55
56  m (0) <= x_int(0) or x_int (1);
57  m (1) <= y_int(0) or y_int (1);
58
59
60  end Behavioral;
```

2 bit 2 to 1 mux dataflow modelling with delays

Simulation of 2-bit 2 to 1 dataflow modelling with delays

```vhdl
entity mux is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           y : in STD_LOGIC_VECTOR (1 downto 0);
           s : in STD_LOGIC;
           m : out STD_LOGIC_VECTOR (1 downto 0));
end mux;

architecture Behavioral of mux is
signal sbar : STD_LOGIC ;
signal x_int : STD_LOGIC_VECTOR (1 downto 0);
signal y_int : STD_LOGIC_VECTOR (1 downto 0);

begin

sbar <= not s after 3 ns;
x_int (0) <= x (0) and s after 3 ns;
x_int (1) <= x (1) and sbar after 3 ns;
y_int (0) <= y (0) and s after 3 ns;
y_int (1) <= y (1) and sbar after 3 ns;



m (0) <= x_int(0) or x_int (1) after 3 ns;
m (1) <= y_int(0) or y_int (1) after 3 ns;


end Behavioral;
```



Part 3

Component code	2-bit 2 to 1 mux structural modelling

```vhdl
entity orgate is
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);
end orgate;

architecture Behavioral of orgate is

begin

o <= i0 or i1;

end Behavioral;
```

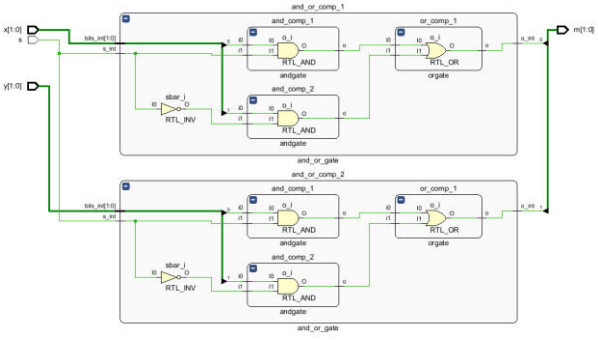
```vhdl
entity muxcomp is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           y : in STD_LOGIC_VECTOR (1 downto 0);
           s : in STD_LOGIC;
           m : out STD_LOGIC_VECTOR (1 downto 0));
end muxcomp;

 architecture gates of muxcomp is

    component and_or_gate --calling "function"
    Port ( bits_int : in STD_LOGIC_VECTOR(1 downto 0);
           s_int : in STD_LOGIC;
           o_int : out STD_LOGIC);end component;

    Signal a_int : STD_LOGIC;
    Signal b_int : STD_LOGIC;
    Signal sbar : STD_LOGIC;
```

```
entity andgate is
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);
end andgate;

architecture Behavioral of andgate is

begin

o <= i0 and i1;

end Behavioral;
```

```
begin

sbar <= not s;

and_or_comp_1 : and_or_gate
port map (
    bits_int => x (1 downto 0),
    s_int => s,
    o_int => m (0)
 );

and_or_comp_2 : and_or_gate
port map (
    bits_int => y (1 downto 0),
    s_int => s,
    o_int => m (1)
 );
```

```
entity and_or_gate is
    Port ( bits_int : in STD_LOGIC_VECTOR(1 downto 0);
           s_int : in STD_LOGIC;
           o_int : out STD_LOGIC);
end and_or_gate;

architecture Behavioral of and_or_gate is

component andgate --calling "function"
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);end component;
component orgate --calling "function"
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           o : out STD_LOGIC);end component;

Signal a_int : STD_LOGIC;
Signal b_int : STD_LOGIC;
signal c_int : STD_LOGIC;
Signal sbar : STD_LOGIC;
```

```
---- 2nd half of mux

 and_comp_3 : andgate
port map (
    i0 => x (1),
    i1 => sbar,
    o => c_int
 );

 and_comp_4 : andgate

 port map (
    i0 => y (1) , --mapping (defining internal connection po
    i1 => s,
    o => d_int --
 );


 or_comp_2 : orgate port map (
    i0 => c_int,
    i1 => d_int,
    o => m (1)
 );
```



```
begin

sbar <= not s_int;

and_comp_1 : andgate
port map (
    i0 => bits_int(0),
    i1 => s_int,
    o => a_int
 );

and_comp_2 : andgate
port map (
    i0 => bits_int(1),
    i1 => sbar,
    o => b_int
 );

 or_comp_1 : orgate
 port map(
    i0 => a_int,
    i1 => b_int,
    o => o_int
    );



end Behavioral;
```

Part 4

2 to 1 mux behavioural modelling

```
entity mux_behave is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : in STD_LOGIC;
           m : out STD_LOGIC);
end mux_behave;

architecture Behavioral of mux_behave is

begin
    process (x,y, s)
    begin
        if s = '0' then
            m <= x;
        else
            m <= y;
        end if;
    end process;
end Behavioral;
```

2-bit 2 to 1 mux behavioural modelling

```
34  entity mux_behave is
35      Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
36             y : in STD_LOGIC_VECTOR (1 downto 0);
37             s : in STD_LOGIC;
38             m : out STD_LOGIC_VECTOR (1 downto 0));
39  end mux_behave;
40
41  architecture Behavioral of mux_behave is
42
43  begin
44      process(x, y, s)
45      begin
46          if s = '0' then
47              m <= x;
48          else
49              m <= y;
50          end if;
51      end process;
52
53  end Behavioral;
```